Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Un proxy VoIP basato su PJSIP a sostegno della mobilita'

Tesi di Laurea in Sistemi Mobili

Relatore:
Chiar.mo Prof.
Ghini Vittorio

Presentata da:
Ciaramella Marco

II Sessione
Anno Accademico 2013/2014

Dedicata
ad Alice ...

Capitolo 1

Introduzione

I servizi di Voice over IP stanno portando a un radicale cambiamento nel modo di comunicare della gente.
Oggigiorno, a seguito dell'abbattimento dei costi di un abbonamento Internet, molte persone preferiscono telefonare attraverso il servizio VoIP sfruttando la propria connessione per evitare costi aggiuntivi.
Questo cambiamento interessa anche le aziende che permettono ai propri clienti un canale alternativo di comunicazione.
A fronte di una rapida diffusione del VoIP fa seguito tutta una serie di possibili ottimizzazioni legate alla presenza di dispositivi dotati di diverse interfacce di rete che potrebbero essere sfruttate per garantire una migliore qualità nelle comunicazioni over IP durante la mobilità.
Ad esempio si potrebbe pensare di cambiare interfaccia di rete su cui trasmettere in base alla fluttuazione del carico oppure semplicemente perché ci si è disconessi da una rete Wi-Fi e si vuole continuare con quella 3G.
L'infrastruttura ABPS (Always Best Packet Switching) offre un supporto a questo scenario, collocando due proxy, uno locale e un altro esterno, lungo il percorso dell'host mobile.
Con la tesi seguente ho voluto continuare il lavoro svolto da Luca Montanari aggiungendo il supporto per il proxy esterno e il supporto a tutta una serie di meccanismi previsti dai clienti VoIP che mancava nella precedente

implementazione.

Indice

Capitolo 2

Scenario

Il nostro scenario di interesse è quello in cui un nodo mobile, su cui sono installate più interfacce di rete, si sposta durante una comunicazione VoIP. Il nodo è in grado di cambiare eventualmente, in modo dinamico, l'interfaccia su cui trasmette a seconda del carico riscontrato o dell'affidabilità del canale su cui sta trasmettendo. Il monitoraggio di questi paramentri permette ad applicazioni real-time, e in particolare di telefonia su Internet, di sfruttare la migliore connessione possibile tra quelle disponibili sul dispositivo mobile per sperimentare una comunicazione più interattiva con l'altro end-system.

2.1 VOIP sfruttando diverse interfacce di rete: ABPS-SIP/RTP

In letteratura esistono già protocolli che utilizzano le diverse interefacce di rete installate sul nodo mobile, come ad esempio un estensione di MIPv6 [5], chiamata Multiple Care of Address (MCoA) , oppure il Flow Mobility technique (FlowMob). Ma tutti questi protocolli vedono le diverse interfacce di rete come una seconda strada da utilizzare in caso di rottura del canale su cui si sta trasmettendo. Quindi, la decisione di cambiare la tecnologia di comunicazione non è presa sulla base di particolari QoS (Quality Of Service), ma sulla rottura del canale di comunicazione corrente.

Nel nostro scenario di interesse consideriamo l'infrastruttura denomita ABPS che sta per Always Best Packet Switching [2].
ABPS, a differenza dei protocolli menzionati prima, monitora costantemente lo stato dei canali che fanno riferimento alle interfacce di rete, per decidere, pacchetto per pacchetto, il percorso migliore su cui instradare.

2.1.1 Archiettura ABPS

Nella figura 1, che mostra l'architettura del sistema ABPS a livello sessione, notiamo due proxy, i cui ruoli sono:

- Proxy locale: crea un tunnel multipath diretto al Proxy Esterno, attraverso tutte le interfacce disponibili (come ad esempio 3G o WiFi).
  Il proxy locale è in esecuzione sulla stessa macchina del nodo mobile. L'applicazione VoIP sarà quindi configurata in modo da mandare i messaggi al proxy locale. Il proxy quindi inoltrerà i messaggi ricevuti al proxy esterno usando l'interfaccia scelta dall'infrastruttura ABPS.

- Proxy esterno o outbound proxy: ha la funzione di Back-to-back user agent. E' un proxy esterno a qualsiasi rete privata, quindi è riconosciuto sulla rete da un suo indirizzo IP pubblico a cui è possibile fare riferimento per contattarlo direttamente.
  La funzione di questo proxy è di creare un canale logico con il Proxy locale in modo tale da poter riconoscere il mittente dei messaggi ricevuti, indipendentemente dall'interfaccia di rete utilizzata. Il proxy esterno maschererà l'indirizzo del proxy locale e inoltrerà i datagram UDP al Correspondent Node (il destinatario) che vedrà solo la comunicazione con il proxy esterno senza percepire ciò che avviene dall'altra parte.

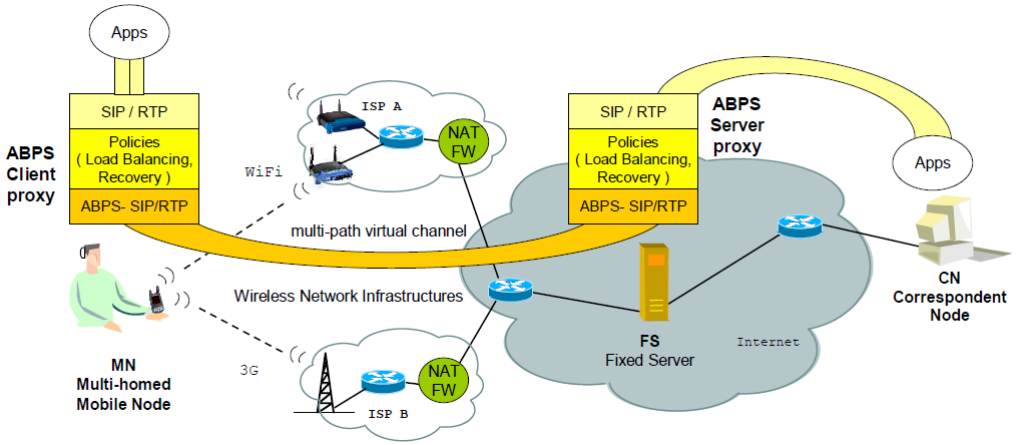

Fig.1 L'architettura di ABPS.

2.2 Problemi nello scenario VoIP

2.2.1 Cambio di indirizzo IP

La problematica relativa al cambio di interfaccia di rete durante un dialogo VoIP è dato dal momento in cui il client mobile decide di cambiare interfaccia. Difatti un'azione di questo tipo porterebbe ad avere in uscita dal nodo mobile pacchetti con un indirizzo IP mittente diverso da quello usato poco prima. L'effetto di questo cambio di indirizzo potrebbe causare sul destinatario l'eliminazione di tutti i pacchetti provenienti dal nuovo indirizzo poiché non riconoscerebbe più l'end-system con cui stava dialogando.

2.2.2 Firewall e NAT System

Il problema che invece affligge una qualsiasi comunicazione VoIP è data dalla presenza dei sistemi NAT e firewall usati per proteggere le reti private e che non permettono ai client VoIP esterni di contattare altri client interni a queste reti.
Un firewall è un sistema di sicurezza che blocca il traffico in ingresso e in

uscita da una rete secondo delle regole. Il firewall quindi stabilisce una barriera tra una rete interna sicura e un'altra rete, ad esempio Internet, che si assume non essere sicura [6].

Il Network Address Translation (NAT) è stato progettato per la conservazioni degli indirizzi IP. Configurando un NAT per una rete privata è infatti possibile usare indirizzi globalmente non unici per i nodi interni che allo stesso tempo possono comunicare con la rete esterna come ad esempio Internet. Questo è possibile perché gli host della rete privata vengono identificati sulla rete esterna da un solo indirizzo IP globalmente univoco.

Il sistema NAT opera su di un router che connette due reti, cioè la rete privata con quella esterna (ad esempio Internet), e traduce gli indirizzi privati dei nodi interni in indirizzi legali, prima che i pacchetti vengano inoltrati all'altra rete. Questo fornisce una sicurezza addizionale andando a nascondere l'intera rete interna dietro quell'indirizzo.

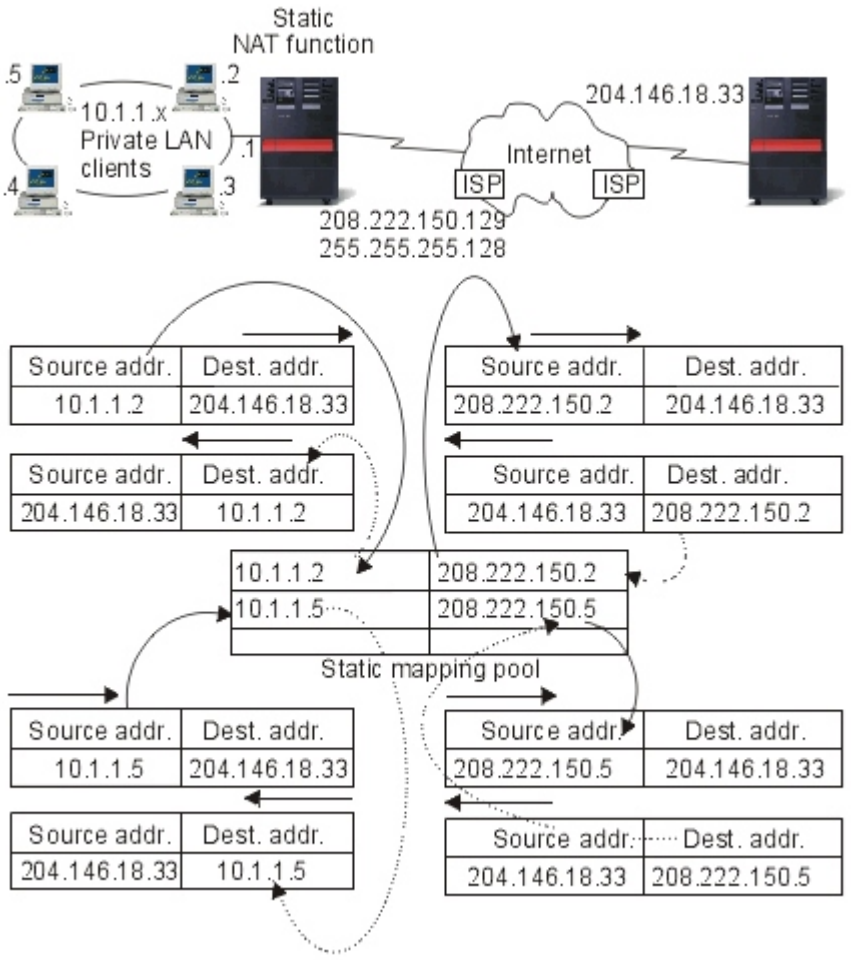

Fig 2.

Funzionamento di un sistema NAT.

Quando un nodo collegato a una rete dietro un NAT o un firewall cerca di collegarsi al web e alle e-mail non riscontra nessun problema. Il problema si pone per le applicazioni di peer-to-peer file sharing e servizi di VoIP per cui un client può svolgere anche la funzione di server, cioè un client in una rete privata diventa un server per un altro client presente sulla rete esterna. Il problema consiste nel fatto che il nodo sulla rete esterna è impossibilitato a contattare il nodo sulla rete privata o perché non ha un IP globale in quanto mascherato dal NAT o perché bloccato da un firewall.

2.3 Soluzione

L'infrastruttura di ABPS è stata progettato allo scopo di supportare le comunicazioni VoIP. Per questo motivo integra una serie di meccanismi per far fronte ai problemi analizzati.

2.3.1 Cambio indirizzo IP

L'approccio adottato da ABPS è quello di usare due proxy server, uno in esecuzione sul nodo mobile e l'altro esterno in esecuzione su una macchina fissa, detto outbound proxy. Ciò che si vuole ottenere è mascherare gli indirizzi del nodo mobile all'altro end-system facendogli credere di comunicare con il proxy server fisso. Poiché l'indirizzo di quest'utlimo rimane sempre lo stesso il destinatario comunicherà sempre con la stessa entità risolvendo così il problema dei pacchetti scartati a causa di un cambio di identità.

2.3.2 Firewall e Sistemi NAT

L'UDP hole punching (foratura) è una tecnica di NAT traversal e firewall traversal comunemente usata per permettere il passaggio del flusso di datagram UDP. Con questa tecnica si stabilisce la connettività tra i due host comunicanti stabilendo lo stato della porta sul server in cui è in esecuzione un NAT o un firewall.
Le porte possono essere attivate dagli host comunicando con un server esterno con indirizzo pubblico, quindi non protetto da NAT e firewall. La comunicazione verso un IP pubblico e una porta valida fa attivare nel server con NAT o firewall la porta e si vedrà quindi costretto ad accettre tutti i pacchetti provenienti da quell'indirizzo IP pubblico e su quella porta.
Una volta impostata la porta sul NAT, lo stato della porta può essere mantenuto attivo, o dal traffico di dati scambiato dalle due entità, oppure in assenza di traffico, dai così detti pacchetti di keep-alive, che di solito sono pacchetti UDP vuoti o con poche informazioni. Nel caso di ABPS i due proxy hanno la responsabilità di mantenere le porte attive sul sistema NAT

o firewall anche in assenza di traffico mentre è in corso una comunicazione VoIP.

Capitolo 3

Protocolli per il VoIP

SIP, Session Initation Protocol [7], è il protocollo usato per l'instaurazione e gestione di un intero ciclo di vita di una sessione VoIP.
Per alcuni aspetti della sessione, SIP si appoggia ad altri protcolli come SDP e RTP/RTCP.

3.1 SIP

Session Initation Protocol [7] serve per iniziare, modificare e terminare sessioni tra uno o più partecipanti.
SIP permette ai partecipanti di essere individuati attraverso il loro indirizzo IP e numero di porta dall'altro end-system. Inoltre consente ai partner di una comunicazione di concordare le modalità con cui scambiarsi i dati, il numero di sessioni multimediali, impostare una chat ecc.
Per alcune delle attività descritte SIP si appongia ad altri standard IETF:

- SDP, Session Description Protocol [3], viene usato per concordare le modalità con cui scambiarsi i dati. Le informazioni SDP vengono trasportate all'interno dei messaggi SIP come corpo del messaggio.

- URI [1], è lo standard per gli identificativi (indirizzi) dei partecipanti ad una sessione.

- RTP/RTCP [8] [4] sono i due protocolli usati per la gestione del traffico multimediale come voce, video, immagini, ecc.

3.1.1 Come funziona SIP

SIP è un protocollo di sessione di tipo testuale in cui le informazioni sono codificate in ASCII e in modo comprensibile all'uomo. Il suo funzionamento è basato sul meccanismo handshake di Request/Response chiamato transazione.
Le entità coinvolte in un dialogo SIP sono dette **User Agent (UA)** che interagiscono attraverso le transazioni.
Ogni transazione inizia con una Request inviata da uno **User Agent Client (UAC)** a uno **User Agent Server (UAS)** e termina con una Final Response inviata nell'altro senso.
Il messaggio di Request specifica nella prima riga il nome del metodo usato nella transazione, come **INVITE, CANCEL, ACK, BYE, REGISTER, OPTIONS ecc**.
Ad esempio il messaggio seguente è una richiesta di BYE.

```
BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
Max-Forwards: 70
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

Mentre nella prima riga di una Response è presente un codice di stato con il seguente significato

- codici 1xx usati per le risposte provvisorie;
- codici 2xx,3xx,4xx,5xx,6xx usati per le Final Response.

Ad esempio una final Response alla Request “BYE” potrebbe essere:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0
```

Come si può vedere in figura 3, dopo la prima riga con il metodo è presente un certo numero di headers seguito infine da una riga vuota, dopo la quale può essere presente il corpo del messaggio, ad esempio SDP.

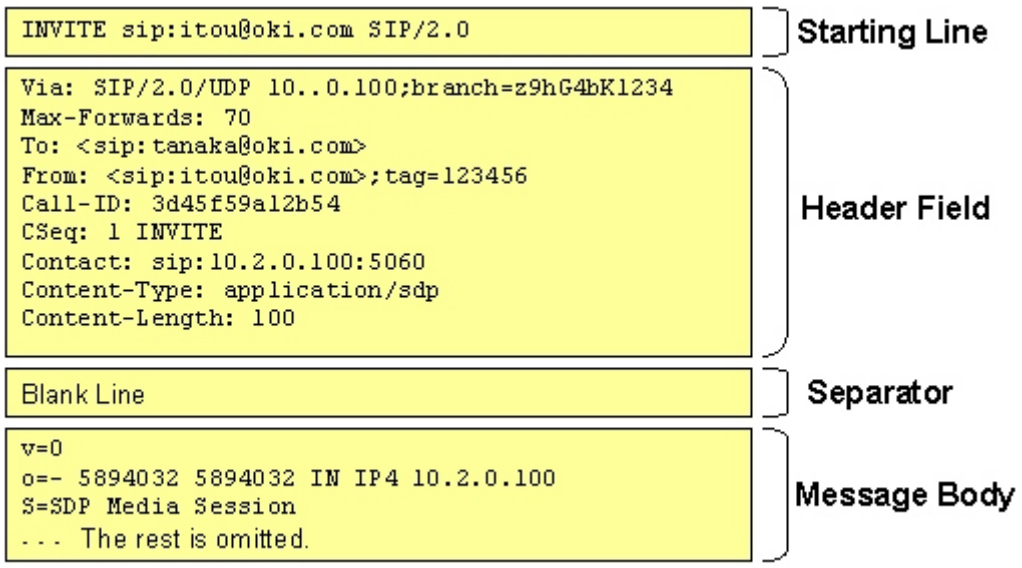

Fig. 3 L’header SIP.

A supporto della comunicazione tra **UA** è presente un Registrar server presso cui i client **UA** si registrano specificando la loro posizione attuale nella rete attraverso una REGISTER Request.
Il Registrar server memorizza le informazioni contenute nel messaggio in un

location service che consulterà quando un UAC farà richiesta di INVITE. Oltre al Registrar, necessario per poter localizzare l'utente con cui si vuole comunicare, può essere aggiunto un Proxy Server che instrada le Request verso gli **UAS** e le Response verso gli **UAC**. Può essere configurato per rispondere direttamente ad una Request operando quindi come UAS. Oppure può rimpiazzare la Request URI con una nuova.

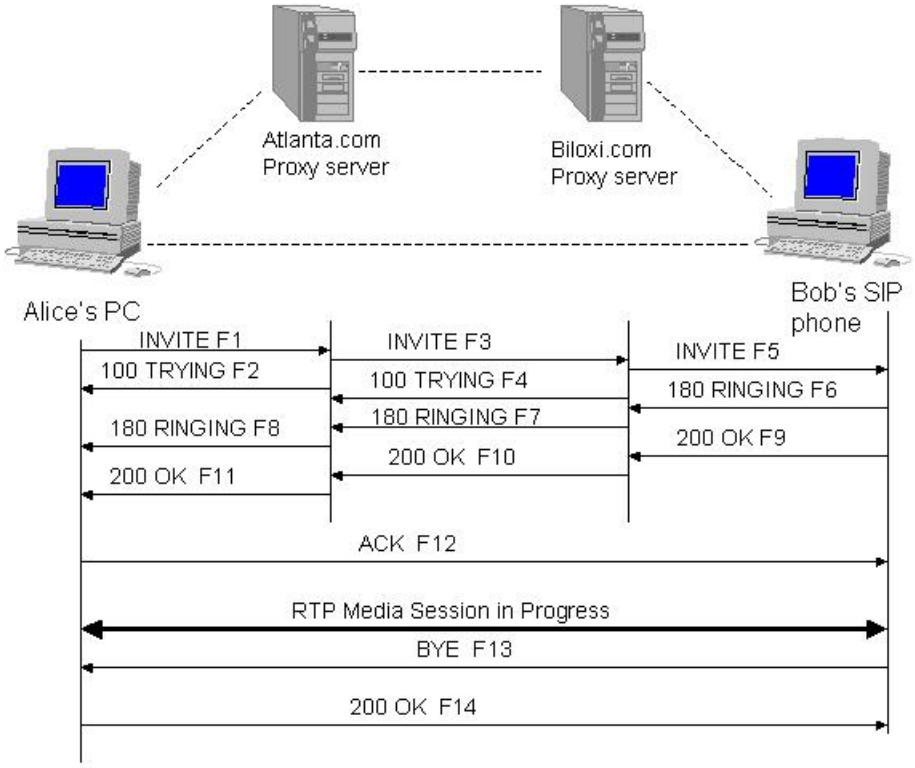

Fig. 4 Interazione tra due client VoIP.

La figura 4 mostra l'interazione tra le componeti descritte dell'infrattura SIP durante una sessione.

SIP URI

La URI di SIP identifica le entità SIP

```
{sip|sips}:[user-part@]domain-part[:port][;transport={UDP|TCP|TLS|...}]
```

le indicazioni su porta e/o trasporto sono consentite solo se la domain-part identifica un host particolare, ad esempio se espressio da un indirzzo IPv4 o IPv6.
La URI può indicare un indirizzo reale (contact-address), cioè l'indirizzo dell'entità SIP, nel caso in cui sia possibile ricostruire tale indirizzo. Oppure un indirzzo pubblico (address-of-record), se permette di ricostruire gli indirizzi di trasporto di un server in grado di locallizare l'entità SIP.

Metodi SIP

L'RFC 3261 definisce i seguenti metodi:

- **REGISTER**: comunica l'indirizzo IP dell'entità SIP al Registrar.

```
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
\textbf{To: Bob <sip:bob@biloxi.com>}
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
\textbf{Contact: <sip:bob@192.0.2.4>}
Expires: 7200
Content-Length: 0
```

Sulla ricezione di questa Register Request, il SIP Registrar associa la URI presente nell'header **Contact** all'indirizzo presente nell'header **To**.

L'immagine seguente mostra la fase di Request/Response della REGISTER.

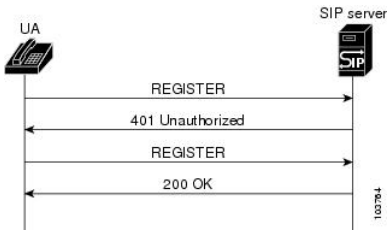

Fig. 5 Le fasi di una richiesta di REGISTER.

Come si può notare nella figura, in un primo momento il Registrar SIP risponde allo UA con una Response di tipo *401 Unauthorized*. In questo messaggio, l'header WWW-Authenticate inserito dal Registrar istruisce il client per l'autenticazione usando la digest authentication. Il client dunque combinerà il parametro *nonce* con la password dell'utente e calcola il valore MD5 per la stringa risultante.
A questo punto trasmette di nuovo una REGISTER Request, questa volta inserendo l'MD5 appena calcolato. Il Registrar, quindi, cofronta il valore MD5 del client con quello calcolato da lui e se coincide risponde con Response 200 OK e inserisce il client nel location database.

- **INVITE**: Inizia un dialogo con le relative sessioni oppure modifica le sessioni di un dialogo già iniziato.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

La prima riga contiene il nome del metodo (INVITE), seguito da una serie di header descritti di seguito:

- **Via** Questo campo rappresenta l'indirizzo da cui proviene la Request e sul quale l'utente si aspetta di ricevere la Response.
- **To** Contiene la SIP URI del destinatario a cui è rivolta la Request originariamente.
- **From** Contiene la SIP URI di colui che ha originariamente generato la Request. E' usato per l'identificazione.
- **Call-ID** Contiene un identificatore globalmente univoco. La combinazione del header To, From e Call-ID definisce in maniera univoca la sessione SIP tra i due utenti (nell'esempio Alice e Bob), detta dialogo.
- **CSeq** La Command Sequence contiene un intero e un nome di metodo. Il valore di CSeq è incrementato per ciascuna nuova richiesta di INVITE dentro lo stesso dialogo.
- **Contact** Contiene un SIP URI che rappresenta la rotta diretta per conttatare l'utente, di solito è composto di un username e un nome di domnio. Ma sono permessi anche indirizzi IP al posto di nomi di dominio.
- **Max-Forwards** Limita il numero di hop che la Request può fare per giungere alla sua destinazione.
- **Content-Type** Contiene una descrizione del corpo del messaggio.
- **Content-Length** Contiene la lunghezza in byte del corpo del messaggio.

- **re-INVITE**: è un'ulteriore INVITE inoltrata per modificare i paramentri della sessione impostati con la precedente INVITE.
  Il caso della RE-INVITE è molto comune, in quanto capita che un utente VoIP voglia aggiungere alla conversazinoe vocale anche il video.

Un'operazione di questo tipo implica l'inoltro di una INVITE quando il dialogo è già attivo.

- **ACK** Segue immediatamente la transazione INVITE, coè lo UAC che riceve la Response inoltra una seconda Request con metodo ACK a cui non fa seguito nessuna Response.

```
ACK sip:bob@186.48.249.1:5060 SIP/2.0
Via: SIP/2.0/UDP 120.146.22.11:5060;....
To: sip:bob@grossaditta.it;tag=bbb...
From: sip:alice@grossaditta.it;tag=aaa...
Call-ID: ccc...
CSeq: 1 ACK
```

- **BYE** La BYE Request termina il dialogo e tutte le sessioni associate.

Attivazione di un dialogo SIP

Dopo che gli utenti si sono registrati presso il loro Registrar presente sul Proxy Server (in realtà basta che solo il destinatario sia registrato), l'UAC che vuole aprire una comunicazione con uno dei suoi contatti SIP (UAS) invierà una INVITE Request al Proxy server dello UAS.
Il SIP Proxy dunque cercherà nel location database i contact address associati allo URI di chi si vuole contattare. Se presente almeno un contact-address, il SIP proxy inoltra l'INVITE all'indirizzo individuato.
Quando l'utente destinatario vede la chiamata e risponde, lo UAS trasmette una Response 200 OK che raggiunge lo UAC mittente transitando negli hop presenti nell'header Via. A questo punto lo UAC è a conoscenza del contact address dello UAS, poiché è presente nell'header Contact della Reponse, e invia un ACK direttamente allo UAS.
A questo punto può iniziare la sessione audio/video/ecc come concordato nella negoziazione SDP.

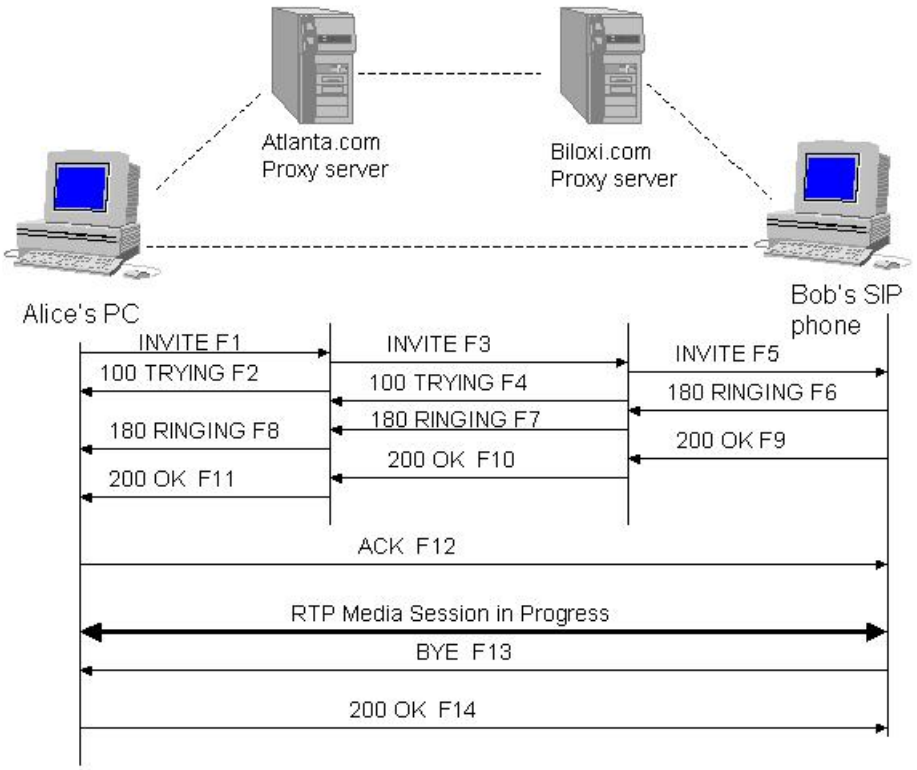

Fig. 6 Fasi di una sessione SIP.

Nella figura sono mostrate le fasi di una sessione SIP dopo che entrambi gli UA si sono registrati presso il loro Registrar di dominio.

3.2 SDP

SDP, Session Description Protocol, è il protocollo usato da SIP per negoziare i parametri dei flussi mulimediali, come numero di flussi, tipo, porta, codec, ecc.
In particolare il messaggio SDP viene trasportato come corpo del messaggio SIP e contiene:

- il tipo di media (video, audio, etc.)

- il protocollo di trasporto (RTP/UDP/IP, H.320, etc.)
- il formato del media (video H.261, video MPEG, etc.)
- indrizzo IP remoto per la trasmissione multimediale
- porta remota per la trasmissione multimediale

Un messaggio SDP consiste di un insieme di righe di testo della forma *type=value* e includono

```
v=  (versione del protcollo)
o=  (identificatore di sessione)
s=  (nome sessione)
i=* (informazioni sulla sessione)
u=* (URI)
e=* (indirizzo email)
c=* (informazioni sulla connessione)
b=* (zero o più linee sulla bandwidth)
k=* (chiave crittografica)
a=* (zero o più linee di attributi di sessione)
m= (nome del media e indirizzo di trasporto)
```

Ad esempio il contenuto seguente

```
v=0
o=marco.ciaramella 1714 638 IN IP4 151.42.227.124
s=Talk
c=IN IP4 151.42.227.124
b=AS:380
t=0 0
m=audio 64494 RTP/AVP 111 110 0 8 104 100 18 3 101
a=rtpmap:111 speex/16000
a=fmtp:111 vbr=on
```

```
a=rtpmap:110 speex/8000
a=fmtp:110 vbr=on
a=rtpmap:104 AMR/8000
a=fmtp:104 octet-align=1
a=rtpmap:100 iLBC/8000
a=fmtp:100 mode=30
a=rtpmap:18 G729/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11
```

specifica uno streming di tipo audio verso l'indrizzo 151.42.227.124 e porta 64494. Il protocollo definisce i campi SDP non estendibili, dunque non sarebbe ammesso aggiungere campi personalizzati.

3.3 RTP/RTCP

Il Real-Time Protocol fornisce i servizi di inoltro per i dati con caratteristiche real-time, come ad esempio audio e video interattivi.
RTP si appoggia a un secondo protocollo, RTCP, Real-Time Control Protocol che monitora la qualità del servizio trasportando informazioni riguardanti i partecipanti in una sessione attiva.
RTP è un protocollo di tipo framework, cioè è volutamente incompleto. Per cui può essere esteso secondo le esisgenze richiesta dall'applicazione.

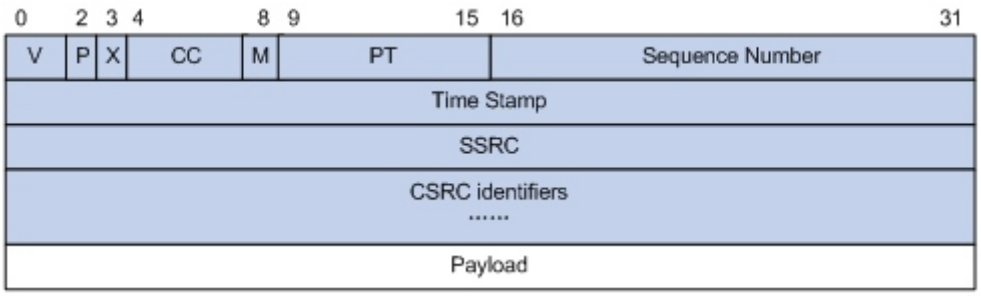

Fig. 7 La struttura di un pacchetto RTP.

Il formato dei messaggi RTP è visualizzato nella figura 7. In particoalre il campo Sequence Number è incrementato di 1 ad ogni pacchetto RTP trasmesso, e può essere usato dal ricevente per determinare la sequenza di pacchetti persi o per riordinare la sequenza.
Mentre il campo TimeStamp di 32 bit contiene l'istante esatto in cui è stato campionato l'audio/video secondo un orolgio definito dall'utente, e serve per permettere la sincronizzazione e il calcolo del jitter.
Il campo SSRC è un identificatore univoco per la sorgente che sta generando i messaggi RTP. Il suo valore dovrebbe essere scelto casualmente in modo da ridurre il rischio di avere lo stesso identificativo per mittente e destinatario durante la stessa sessione RTP.
Il campo CSRC rappresenta una lista di 0 fino a 15 elementi di 32 bit, dove ogni elemento identifica una sorgente che contribuisce alla costruzione del payload contenuto nel messaggio. Ad esempio, per i pacchetti audio, nella lista CSRC saranno presenti gli identificativi SSRC di tutte le sorgenti che verranno mixate per ricostruire il payload da riprodurre.

Capitolo 4

Strumenti

Il proxy da cui sono partito, è un proxy stateless scritto con la libreria PJSIP, incluso tra gli esempi della suite PJ e modificato precedentemente da alcuni ragazzi, Luca Montanari prima e Andrea Monzali e Marco Melletti poi.
PJ è una suite di librerie open source (licenza GPL 2.0) scritte in C che implementa uno stack SIP e uno stack multimediale di supporto al VoIP, instant messaging e comunicazioni multimediali.
La scelta di adoperare tali librerie è dovuta principalmente alle sua portabilità. PJ infatti funziona su architetture a 32 bit, 64 bit, big endian e little endian. Questa caratteristica fa di PJ uno strumento utile per scrivere una sola volta applicazioni destinate a un gran numero di dispositivi.

4.1 La suite PJ

La suite è costituita di diverse librerie:

- **PJSIP**: rappresenta uno stack SIP che supporta un insieme di features ed estensioni del protocollo SIP.
- **PJLIB**: rappresentante la libreria a cui le restanti si appoggiano. Si oc- cupa di garantire funzionalità di base (es. l'astrazione rispetto

al sistema operativo sottostante). Può essere considerata una replica della libreria libc con l'aggiunta di alcune features come la gestione dei socket, funzionalità di logging, gestione thread, mutua esclusione, semafori, critical section, fun- zioni di timing, gestione eccezzioni e definizione di strutture dati di base (liste, stringhe, tabelle, ecc...).

- **PJLIB-UTIL**: anch'essa come PJLIB è una libreria di appoggio, ma a differenza della prima implementa funzioni più complesse utili per la crittografia come gli algoritmi: SHA1, MD5, HMAC, CRC32. Ed anche funzioni per il parsing e la manipolazione di testi.
- **PJMEDIA**: questa libreria si occupa del trasferimento e della gestione dei dati multimediali. Tra le sue caratteristiche principali vi è la gestione degli stack RTP/RTCP.
- **PJSUA**: rappresenta il livello più alto fungendo da wrapper verso le altre librerie. Inoltre agevola notevolmente la scrittura di applicazioni.

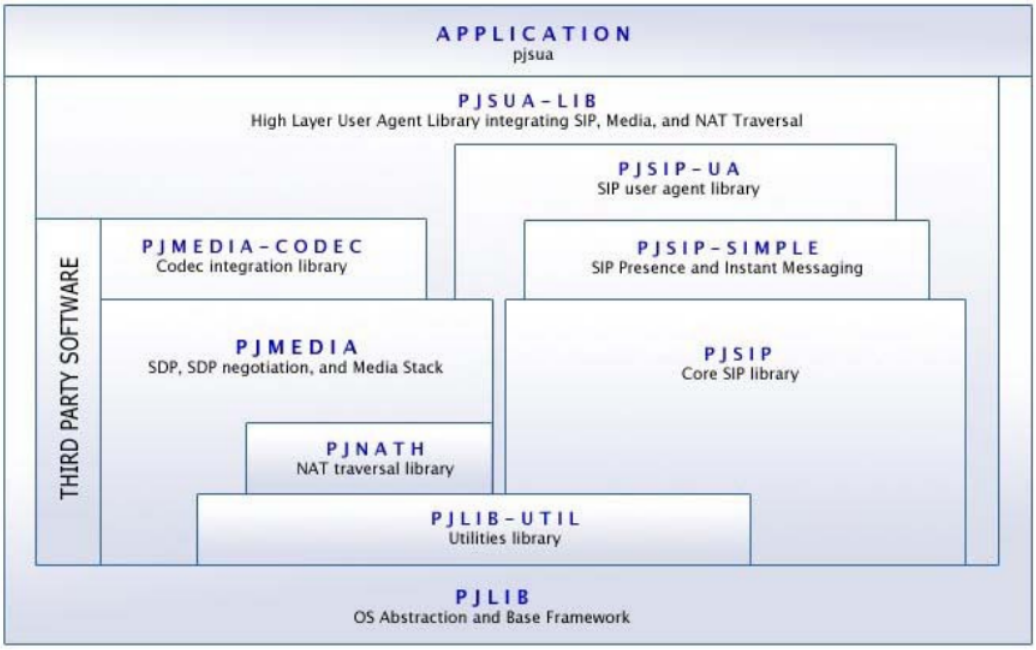

Fig. 8 Lo stack di PJ.

D'ora in poi mi riferirò con PJSIP all'intera suite PJ.

4.2 Architettura PJSIP

PJSIP nonostante sia scritta in C, che non è un linguaggio a oggetti, ha un'architettura fortemente modulare. Tutte le componenti di PJSIP sono implementate come moduli. Il nucleo di questa architettura è costituito da SIP_ENDPOINT che si occupa di:

- allocare la memoria per i moduli;
- temporizza lo heap e schedula gli eventi da notificare ai moduli;
- gestisce i moduli di trasporto e controlla il parsing dei messaggi;
- inoltra i messaggi SIP dal livello trasporto ai moduli interessati.

Ecco come si presenta la struttura di un modulo:

```
struct pjsip_module
{
   PJ_DECL_LIST_MEMBER(struct pjsip_module);              // For internal list mgmt.
   pj_str_t            name;                              // Module name.
   int                 id;                                // Module ID, set by endpt
   int                 priority;                          // Priority

   pj_status_t (*load)          (pjsip_endpoint *endpt);  // Called to load the mod.
   pj_status_t (*start)         (void);                   // Called to start.
   pj_status_t (*stop)          (void);                   // Called top stop.
   pj_status_t (*unload)        (void);                   // Called before unload
   pj_bool_t   (*on_rx_request) (pjsip_rx_data *rdata);   // Called on rx request
   pj_bool_t   (*on_rx_response)(pjsip_rx_data *rdata);   // Called on rx response
   pj_status_t (*on_tx_request) (pjsip_tx_data *tdata);   // Called on tx request
   pj_status_t (*on_tx_response)(pjsip_tx_data *tdata);   // Called on tx request
   void        (*on_tsx_state)  (pjsip_transaction *tsx,  // Called on transaction
                                 pjsip_event *event);     // state changed
};
```

Fig. 9 La struttura di un modulo di PJSIP.

All'interno della struttura del modulo, troviamo dei puntatori a funzione che sono tutti opzionali: se non vengono specificati, il valore di ritorno sarà SUCCESSFUL, altrimenti andranno a svolgere i loro rispettivi compiti, descritti qui di seguito:

- *ON_RX_REQUEST()* e *ON_RX_RESPONSE()* sono i mezzi principali per ricevere i messaggi dall'endpoint SIP o da altri moduli, dove il valore di ritorno è estremamente importante. Se un callback torna non zero (cioe true), significa che il modulo ha provveduto al messaggio e l'endpoint terminerà la distribuzione del messaggio ad altri moduli.

- *ON_TX_REQUEST()* e *ON_TX_RESPONSE()* sono chiamati dal gestore dei trasporti prima che un messaggio venga trasmesso: in questo modo si da la possibilità ad alcuni moduli (ad esempio quello dedito alla firma dei pacchetti) di fare un' ultima modifica al messaggio. Tutti i moduli devono poi restituire come valore di ritorno PJ_SUCCESS, altrimenti la connessione verrà annullata.

- *ON_TSX_STATE()* viene utilizzato per ricevere una notifica ogni volta che lo stato della trasmissione viene cambiato: per esempio a causa del ricevimento di un messaggio, oppure dalla scadenza di alcuni timer o da errori dovuti al trasporto.

PJSIP si basa sul concetto di gerarchia, in cui ogni modulo è identificato da un valore che ne indica la priorità. Questo valore specifica l'ordine con la quale i moduli vengono chiamati: più sarà basso il valore, maggiore priorità avrà il modulo.
Una volta chiamato un modulo, questo invocherà le funzioni *on_rx_request()* e *on_rx_response()* per gestire i messaggi in entrata di tipo Request e Response rispettivamente, e le funzioni *on_tx_request()* e *on_tx_response()* per i messaggi in uscita.
Se un modulo invece desidera accedere alla struttura del messaggio, prima del livello trasporto, dovrà essere impostata una priorità maggiore.
Quando arriva un messaggio, questo viene memorizzato all'interno della struttura *pjsip_rx_data* e passato all'endpoint. Quest'ultimo distribuisce poi il messaggio a ciascun modulo registrato, chiamando *on_rx_request()* oppure *on_rx_response()* a partire dal modulo con priorità maggiore, cioè col valore di priorità più basso, finchè uno di loro non restituisce un valore diverso da

zero (cioè true). In tal caso, l'endpoint interrompe la distribuzione del messaggio verso altri moduli, in quanto presuppone che il modulo sia occupato nel trattamento del messaggio.
Il modulo che restituisce true a sua volta può inoltrare ulteriormente il messaggio ad altri moduli.
Per quanto riguarda i messaggi in uscita, anche questi vengono rappresentati all'interno di una struttura, che prende il nome di *pjsip_tx_data*, la quale contiene anche un buffer contiguo, un pool per la memoria e informazioni di trasporto.
Quando viene eseguita la funzione *pjsip_transport_send()* per inoltrare un messaggio, il gestore dei trasporti, chiama le funzioni *on_tx_request()* oppure *on_tx_response()*, per tutti i moduli, a partire ovviamente da quello con priorità maggiore.

Capitolo 5

Precedente implementazione

Il mio lavoro di tesi è partito dal proxy locale implementato da Luca Montanari e successivamente modificato da Andrea Monzali e Marco Melletti, per l'architettura ABPS, chiamato *pj_relay*.
Il proxy client implementato ha il compito di intercettare i messaggi SIP invitati dal client UA in esecuzione sulla stessa macchina e modificare i campi di intestazione in modo che facciano riferimenti a lui e non allo UA.
Come client VoIP è stato usato *pjsua* facente parte della suite PJ che può essere configurato attraverso un file di configurazione in cui specificare la porta per i messaggi SIP e l'indirizzo e porta del proxy a cui inoltrare i messaggi.

5.1 Funzionamento del proxy pj_relay

Il compito del proxy locale, secondo l'infrastruttura ABPS, è quello di modificare i messaggi in uscita provenienti dallo UA locale, sosituendo la porta con la propria, prima di inoltrarli verso il proxy esterno.
Le informazioni modificate devono essere salvate in modo tale da reinserile per i messaggi in ingresso da inoltrare allo UA locale.
Le informazioni modificate riguardano i campi Via e Contact di SIP visti nel

capitolo sui protocolli del VoIP.

5.1.1 Esempio di funzionamento

Come esempio si considera il caso in cui il *pjsua* sul nodo mobile voglia fare richiesta di INVITE al pjsua esterno.
Secondo il protocollo SIP il messaggio INVITE dovrà essere inoltrato al proxy server del dominio su cui è registrato il pjsua mobile, o in caso non si fosse registrato, direttamente al proxy server su cui è registrato il correspondent node
Nel nostro scenario il comportamento atteso sarà quello di inoltrare il messaggio al proxy locale il quale inserirà se stesso come mittente della INVITE in modo da poter riceve successivamente la Response.
Nella configurazione che ho usato, il client VoIP *pjsua* usa la porta *4445* per le comunicazioni SIP e come account URI *marco.ciaramella.sip1@ekiga.net* che ho creato su ekiga.net.
Mentre il proxy client *pj_relay* è stato configurato per usare la porta *5060*.
Nella figura viene mostrato l'header di una INVITE Request così come ricevuto dal proxy locale.

```
INVITE sip:marco.ciaramella.sip2@ekiga.net SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:44445;rport;branch=z9hG4bKPjxFr-
PKZKhaCOlbe8.HU9I9wU38pZy9fq
Max-Forwards: 70
From: sip:marco.ciaramella.sip1@ekiga.net;tag=uEiC-AJyCN-P0-XXUMY.19tBETcxS1qO
To: sip:marco.ciaramella.sip2@ekiga.net
Contact: <sip:marco.ciaramella.sip1@127.0.0.1:44445;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Route: <sip:127.0.0.1:5060;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER,
MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Content-Type: application/sdp
Content-Length:   472

v=0
o=- 3619546100 3619546100 IN IP4 192.168.1.2
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 60010 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 192.168.1.2
b=TIAS:64000
a=rtcp:60011 IN IP4 192.168.1.2
a=sendrecv
a=rtpmap:98 speex/16000
a=rtpmap:97 speex/8000
a=rtpmap:99 speex/32000
a=rtpmap:104 iLBC/8000
a=fmtp:104 mode=30
a=rtpmap:3 GSM/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:9 G722/8000
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15
```

Fig. 10 Header SIP della INVITE ricevuta dal proxy locale.

L'header Route evidenziato indica che il messaggio era destinato a lui. Dunque il proxy sostituisce le informazioni relative al *pjsua* e inserisce se stesso come mittente della INVITE Request. Infine inoltra la INVITE al proxy server ekiga.net secondo le specifiche del protocollo SIP.
Nella figura possiamo vedere l'header della INVITE Request in uscita, cioè poco prima di essere inoltrata al correspondent node e dopo le modifiche fatte dal proxy.

```
INVITE sip:130.136.4.240:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:5060;rport;branch=z9hG4bKPjPKcMBzp6yhIZQG-
du1TsYkW1MPmX6L5V
Via: SIP/2.0/UDP 192.168.1.2:44445;rport=44445;received=127.0.0.1;branch=z9hG4bKPjxFr-
PKZKhaCOlbe8.HU9I9wU38pZy9fq
Max-Forwards: 69
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-XXUMY.19tBETcxS1qO
To: <sip:marco.ciaramella.sip2@ekiga.net>
Contact: <sip:marco.ciaramella.sip1@127.0.0.1:44445;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER,
MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Route: <sip:marco.ciaramella.sip2@ekiga.net>
Content-Type: application/sdp
Content-Length:   472

v=0
o=- 3619546100 3619546100 IN IP4 192.168.1.2
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 60010 RTP/AVP 98 97 99 104 3 0 8 9 96
c=IN IP4 192.168.1.2
b=TIAS:64000
a=rtcp:60011 IN IP4 192.168.1.2
a=sendrecv
a=rtpmap:98 speex/16000
a=rtpmap:97 speex/8000
a=rtpmap:99 speex/32000
a=rtpmap:104 iLBC/8000
a=fmtp:104 mode=30
a=rtpmap:3 GSM/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:9 G722/8000
a=rtpmap:96 telephone-event/8000
a=fmtp:96 0-15
```

Fig. 11 Header SIP della INVITE dopo le modifiche del proxy locale.

L'header *Via* evidenziato mostra come il proxy client abbia aggiunto se stesso nella lista dei *Via*. Mentre l'header *Route* mostra che il messaggio è direzionato al proxy server *ekiga.net*, cioè verso l'esterno.

Le informazioni relative allo UA interno vengono memorizzate nella struttura PJSIP denominata *src_info* e definita nel modo seguente:

```
struct src_info_struct {                /* struttura per la
    destinazione originle dei pacchetti */

  DECL_LIST_MEMBER(struct src_info_struct);    /* pointer to the
    next/preceding */
                                                /* structure
    for dynamic allocation */
   pj_str_t username;
   pj_sockaddr src_addr;

   int active_calls;
   int src_type;                          /* source type: PERM_HOST
    or TEMP_HOST */

   pj_timestamp touch;
   pj_pool_t* pool;
};
```

la struttara memorizza nel campo *src_addr* l'indirizzo IPv4 o IPv6 del client e in *username* l'URI del nome utente SIP.
Le informazioni salvate in questa struttura verrano usate dal proxy sulla ricezione della INVITE Response, facendo quindi il lavoro inverso a quello descritto per la Request in modo che resti trasparente al client VoIP. Questo è necessario poiché il *pjsua* deve riconoscere se stesso negli header SIP altrimenti potrebbe scartare tutti i messaggi SIP.

Capitolo 6

Obiettivi

6.1 Limiti della precedente implementazione

L'implementazione descritta nel precedente capitolo fa riferimento al solo proxy locale dell'architettura ABPS.
Come è chiaro dalla descrizione dello scenario in cui stiamo lavorando, manca la parte di proxy esterno fisso con funzione di Back-to-back user agent che nasconde il nodo mobile. Infatti senza la controparte di server fisso con indirizzo IP pubblico non è possibile superare sistemi NAT e firewall come descritto nelle tecniche di NAT e firewall traversal. Inoltre, si vorrebbe permettere a un utente che non vuole essere contattato, e dunque non visibile sulla rete, di contattare tramite un INVITE un'altro utente precedentemente registrato.
Per come è stato implementato il proxy *pj_relay*, però, una comunicazione dallo UA sul nodo mobile può essere instaurata solo dopo essersi registrato presso il suo Registrar. Se l'UA interno volesse fare INVITE senza prima aver fatto la REGISTER, il proxy sulla ricezione della INVITE Request non sarebbe in grado di capire da che direzione provenga, se da locale (interno) oppure da un UA esterno che sta cercando di contattare il client mobile. Come vedremo più avanti il problema di poter fare INVITE senza esserci registrati sul nostro Registrar si limita al problema di capire da dove proviene

la INVITE, se dallo UA del nostro nodo mobile o dallo UA esterno.
Relativamente alla mancaza del secondo proxy server descritto nel nostro scenario, nella precedente implementazione manca il supporto ai keep-alive.

6.2 Obiettivi

Nella mia tesi ho cercato di far fronte ai limiti analizzati, aggiungendo quella parte dell'infrastruttura ABPS che abbiamo visto mancare nella precedente implementazione.
L'obiettivo della mia tesi è stato dunque:

- aggiugere il supporto al proxy esterno descritto nel capitolo 2.
- permettere al nodo mobile di poter contattare con una INVITE il correspondent node.
- una volta aggiunto il Back-to-back user agent, aggiungere la gestione dei keep-alive tra proxy locale e proxy esterno in modo da poter bucare eventuali firewall o sistemi NAT tenendo attiva la comunicazione .
- relativamente all'instaurare un dialogo con un UA esterno senza esserci registrati, capire se è necessario o no usare una seconda porta sui due proxy.

6.2.1 Proxy esterno di ABPS

La parte di proxy esterno dovrà essere in grado di accettare più di un utente che voglia utilizzare ABPS mentre è in mobilità e dovrà far comparire se stesso nei messaggi SIP in uscita sulla rete esterna rimanendo trasparente ai client mobili.

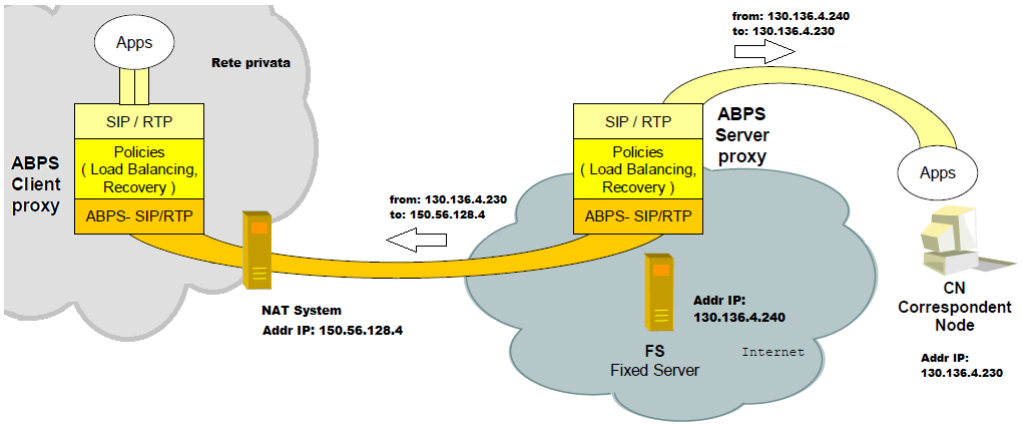

Fig.12 La figura mostra il comportamento trasparente che dovrebbe avere il proxy esterno. Al Correspondent Node deve sembrare di comunicare con il proxy esterno, mentre al nodo mobile di comunicare con il proxy locale.

6.2.2 Da dove proviene l'INVITE e a chi inoltrarla

Il problema di poter fare un INVITE senza che lo UAC si sia precedentemente registrato è stato uno dei principali obiettivi di questa tesi. Questo perché se i proxy inizialmente non vengono attraversati dalla REGISTER dell' UAC mobile sucessivamente, sulla ricezione di una richiesta di INVITE, si troveranno a dover gestire un messaggio di cui non hanno alcuna informazione sugli utenti interessati.
Abbiamo visto nel capitolo precedente che quando i proxy vengono attraversati dalla richiesta di REGISTER del nodo interno, conservano l'username del mittente nella struttura *src_info*. La lista di *src_info* su un proxy come abbiamo detto mantiene tutti gli username passati attraverso lui.
Nel caso in cui l'utente sul nodo mobile si registra, il suo usename verrà catturato dai due proxy lungo la rotta.
Quando si presenta in ingresso una richiesta di INVITE questa può provenire o dall'esterno, e cioè dal proxy server del dominio su cui è registrato un' utente che ci sta contattando, oppure dall'interno, e cioè dal client VoIP sul nodo mobile che sta contattando un host esterno. In entrambi i casi se

è stata effettuata la richiesta di REGISTER sarà presente all'interno della lista di *src_info* lo username del destinatario, nel primo caso, o mittente nel secondo. Quindi i proxy sono in grado di capire da dove proviene facendo una ricerca per username nella lista di *src_info*.
Dunque se la ricerca dello username del mittente presente nel campo *msg_info.from* della stuttura *pjsip_rx_data* va a buon fine, il proxy può concludere con certezza che la richiesta di INVITE proviene dall'interno, cioè dallo UA mobile del nostro scenario e quindi può inoltrarla all'esterno, cioè al Proxy server del dominio dell'utente mittente. Se invece la ricerca dello username del destinatario presente nel campo *msg_info.to* della struttura *pjsip_rx_data* va a buona fine, il proxy può concludere con certezza che la richiesta di INVITE proviene dall'esterno, e può inoltrarla all'interno verso l'UA mobile.
Quindi il comportamento dei due proxy sarà il seguente:

- Se la INVITE viene ricevuta dall'outbound proxy, cioè il proxy esterno, allora:
  - se proviene dal proxy sul nodo mobile, lo inoltra verso il Registrar su cui è iscritto l'utente e specificato nel campo .....
  - se proviene dallo UA esterno allora lo inoltra al proxy sul nodo mobile usando l'informazione memorizzata durante la fase di registrazione.
- Se l'INVITE viene ricevuta dal proxy locale, cioè dal proxy sul nodo mobile, allora:
  - se proviene dallo UA interno, lo inoltra all'outbound proxy utilizzando l'indirizzo e la porta specificati
  - se proviene dallo UA esterno e quindi dall'outbound proxy, lo inoltra allo UA locale.

I client VoIP in genere consento agli utenti di non registrarsi presso un Registrar al fine di evitare di essere visibili sulla rete e di essere contattati.

Nel nostro scenario in cui sono presenti i due proxy di ABPS è necessario prevedere questo caso in modo da renderlo compatibile con tutti i client in circolazione.

Capitolo 7

Progettazione

Sulla base degli obiettivi discussi, il mio compito, dunque, è stato quello di aggiungere l'outbound proxy e tutte quelle funzionalità tra i due proxy discusse nello scenario.

7.1 Progettazione dell'outbound proxy

Come ho già detto, la funzionalità dell'outbound proxy è quella di Back-to-back user agent, in grado di creare un canale logico con il Proxy locale e mascherare il suo indirizzo inoltrando i messaggi SIP modificati in maniera tale che dall'esterno appai lui come entità SIP. Poiché il proxy locale come abbiamo visto svolge già la funzione di mascherare le informazioni del client VoIP inserendo se stesso nei messaggi SIP, ho deciso di riutilizzare l'implementazione del *pj_relay* senza apportare modifiche per la gestione delle entità SIP.

7.2 Implementazione dell'outbound proxy

La distinzione tra proxy interno e outbound viene fatta a seconda dei parametri passati. Al *pj_relay* viene specificata la modalità di proxy interno specificando come parametri l'indirizzo del secondo *pj_relay* con funzione di

outbound. Mentre il *pj_relay* sarà in modalità proxy server se non viene specificato l'indirizzo di un altro *pj_relay*.
Quando il proxy interno riceve un messaggio dal client VoIP locale, lo inoltra all'outbound proxy specificato, mentre quando l'outbound proxy riceve un messaggio dal proxy sul nodo mobile, lo inoltra alla destinazione effettiva.
La funzionalità dei due proxy, come vedremo, cambia solamente nella gestione dei messaggi di keep-alive.
Quindi come faceva già il proxy interno, l'outbound proxy avrà il seguente comportamento:

- sulla ricezione di una SIP Request, sostituirà le informazioni dei campi *Via* e *Contact* inserendo se stesso.

- sulla ricezione di una SIP Response, sostituirà le informazioni che fanno riferimento a lui presenti nel campo *Via* con l'indirizzo del proxy client. Mentre nel campo *Contact* rimane il suo indrizzo, poiché le specifiche SIP dicono che l'indirizzo nel *Contact* sarà quello a cui verrano inoltrati i pacchetti RTP/RTCP.

7.3 Progettazione del meccanismo di inoltro della INVITE

Il problema di dove inoltrare una INVITE senza che l'utente sul nodo mobile sia prima passato attraverso i proxy tramite una REGISTER, può essere affrontato analizzando le informazioni in possesso del proxy e del fatto che un nodo esterno non può inviare una INVITE al nodo mobile se questo non si è prima registrato.
Infatti l'outbound proxy, a monte della rotta che collega il nodo mobile col mondo esterno, non è raggiungibile da nessuna richiesta esterna se il suo indirizzo non è stato memorizzato sul Registrar tramite una REGISTER dal nodo mobile.
Questo ci permette di capire che una INVITE Request i cui username *From*

e *To* non sono presenti nella lista di *src_info* può provenire solamente dall'interno. Infatti se provenisse dall'esterno allora il nodo mobile si sarebbe dovuto registrare precedentemente per il motivo descritto prima. Infatti registrandosi sarebbe passato per i due proxy che avrebbero fissato l'username nelle liste e sul registrar sarebbe stato salvato l'indirizzo IP pubblico dell'outbound proxy. Dunque se sulla ricezione della INVITE Request i due proxy non vedono né mittente né destinatario nelle loro liste, possono concludere che certamente il messaggio proviene dal nodo mobile e che quindi va mandato all'esterno.
Gli altri casi in cui è presente almeno un'username *From* e/o *To* nei proxy possono essere trattati nel modo seguente:

- se è presente l'username *From* nelle liste dei proxy allora significa che proviene dal nodo mobile e quindi va inoltrata all'esterno.

- se è presente l'username *To* nelle liste dei proxy allora significa che proviene dall'esterno e quindi va inoltrata all'interno.

- il caso in cui sono presenti entrambi gli username al momento non è gestito, poiché ci si aspetta che sulla chiusura di un dialogo VoIP venga inoltrata una BYE. Il comportamento dei proxy sulla ricezione della BYE è di eliminare le informazioni sulla chiamata in corso, identificato dal call-id, rimuovendo gli username coinvolti dalla lista di *src_info*. Quindi su una seconda INVITE si ricomincerebbe da capo.
  Diverso è invece il comportamento in caso di una seconda INVITE (re-INVITE) a seguito di un cambio nei paramentri di comunicazione o dopo che la sessione è stata interrotta per un guasto. In questo caso il proxy avrebbe entrambi gli username ma per come è attualmente implementato non sarebbe in grado di decidere nulla a riguardo.

7.4 Implementazione dell'algoritmo di inoltro della INVITE

Possiamo scrivere lo pseudocodice dell'algoritmo di decisione sulla direzione della INVITE Request:

```
if( mittente non appartiene alla lista src_info && destinatario
    non appartiene alla lista src_info){
  inoltra INVITE Request verso esterno ;
}
else if ( mittente appartiene alla lista src_info){
  inoltra INVITE Request verso esterno ;
}
else if ( destinatario appartiene alla lista src_info){
  inoltra INVITE Reqeust verso interno ;
}
else{
  non gestito ;
}
```

La direzione viene stabilita nell'header *Route* leggendo il un campo nei dati ausiliari che il proxy scrive indicando la direzione. Prima dell'inoltro il proxy crea l'header *Route* in cui speciica l'indirizzo e porta del prossimo hop lungo la rotta.

7.5 Progettazione del supporto ai keep-alive

Nel nostro scenario mobile è indispensabile trattare anche il caso di ritrovarsi in una rete protetta dai NAT o firewall e che impediscono utenti esterni di contattare il nodo dietro questi sistemi.
Come visto una tecnica di NAT e firewall traversal consiste nel forare il sistema di protezione comunicando con un host esterno e con indirizzo pubblico

il quale potrà successivamente dialogare con l'host protetto.
Il nostro oubound proxy dell'infrastruttura ABPS ha anche questo scopo.
Il meccanismo di keep-alive inoltre deve interessare solo i due proxy e solo lungo la rotta dei messaggi SIP. Questo perché le comunicazioni RTP/RTCP avvengono su due canali distinti da quello SIP per i quali non è strettamente necessario mantenere le porte attive poiché in genere i client VoIP generano un flusso costante di messaggi anche nel caso in cui i due interlocutori smettono di parlare. Mentre è necessario mantenere aperte le porte per il passaggio dei messaggi SIP in quanto dopo l'avvio dello streming sulle porte SIP può non venire trasmesso più nulla per molto tempo. Questo causa un problema nel momento in cui l'UA esterno voglia fare una richiesta di re-INVITE o di BYE poiché il suo messaggio verrebbe scartato dal NAT o firewall.

7.6 Implementazione del supporto ai keep-alive

Il meccanismo prevede l'attivazione della trasmissione di datagram UDP vuoti nel momento in cui viene instaurata una comunicazione dopo la ricezione di una INVITE Response. Il comportamento dei due proxy è differente ed è il seguente:

- proxy locale: trasmette i keep-alive verso il proxy server pubblicco così che possa ricevere le trasmissioni da lui;

- proxy esterno: trasmette i keep-alive verso il proxy locale del nodo mobile così che in caso in cui il proxy sul nodo mobile si dovesse disconttere dalla rete per un certo periodo di tempo, il proxy fisso continua a tenere il canale SIP aperto sul sistema NAT o firewall.

L'implementazione prevede anche un numero massimo di trasmissioni di keep-alive quando non è più attiva la sessione a seguito di un guasto e che

mai più si riattiverà. Può succedere, infatti, che una sessione si disattivi involontariamente poco prima che un utente abbia fatto BYE e che quindi non venga più riattivata da nessuno degli interlocutori. In questo caso i due proxy credendo che lo stream RTP/RTCP sia ancora attivo continuano con il loro indefinito scambio di keep-alive.

L'algoritmo di inoltro dei keep-alive, dunque, prevede di trasmettere il prossimo keep-alive se si sta ascoltando del traffico RTP/RTCP in ingresso o in uscita. Dal momento in cui non è più presente del traffico, l'algoritmo conta il numero di keep-alive trasmessi senza che ci sia in sottofondo lo stream RTP/RTCP. Quando raggiunge un numero massimo di trasmissioni decide che i due utenti si sono disconnessi in modo anomalo e quindi smette di trasmettere.

Capitolo 8

Valutazione

Il test che ho condotto e riportato in questo capitolo riguarda il nostro scenario di interesse.
Ho, dunque, eseguito un *pjsua* e il *pj_relay* locale su una macchina connessa a una rete privata e protetta da un sistema NAT. L'outbound proxy, invece, l'ho lanciato su una macchina del laboratorio Ercolani con indirizzo 130.136.4.240, quindi su una rete pubblica. Mentre il secondo *pjsua* su una macchina sempre del laboratorio, con indirizzo 130.136.4.230. Di seguito ho riportato gli header SIP che mostrano come variano i campi prima e dopo il passaggio attraverso i proxy.
Nel test riportato il nodo mobile non ha eseguito la REGISTER così che possa essere verificato il corretto funzionamento dell'algoritmo di decisione della rotta e allo stesso tempo il comportamento dell'outbound proxy secondo le specifiche di ABPS.

8.1 Proxy locale

```
INVITE sip:marco.ciaramella.sip2@ekiga.net SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:44445;rport;branch=z9hG4bKPjxFr-
PKZKhaCOlbe8.HU9I9wU38pZy9fq
Max-Forwards: 70
From: sip:marco.ciaramella.sip1@ekiga.net;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To: sip:marco.ciaramella.sip2@ekiga.net
Contact: <sip:marco.ciaramella.sip1@127.0.0.1:44445;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Route: <sip:127.0.0.1:5060;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Content-Type: application/sdp
Content-Length:   472
```

Questa è la richiesta di INVITE ricevuta dal proxy locale. Si nota come il Via e il Contact facciano riferimento al client pjsua.

```
INVITE sip:130.136.4.240:5060 SIP/2.0
Via: SIP/2.0/UDP
192.168.1.2:5060;rport;branch=z9hG4bKPjPKcMBzp6yhIZQG-
du1TsYkW1MPmX6L5V
Via: SIP/2.0/UDP
192.168.1.2:44445;rport=44445;received=127.0.0.1;branch=z9hG4b
KPjxFr-PKZKhaCOlbe8.HU9I9wU38pZy9fq
Max-Forwards: 69
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To: <sip:marco.ciaramella.sip2@ekiga.net>
Contact: <sip:marco.ciaramella.sip1@127.0.0.1:44445;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Route: <sip:marco.ciaramella.sip2@ekiga.net>
Content-Type: application/sdp
Content-Length:   472
```

Questa è la richiesta di INVITE dopo la modifica da parte del proxy locale. Si nota come il proxy ha modificato l'header Via inserendo se stesso in modo da poter ricevere la Response in seguito.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
192.168.1.2:5060;rport=61708;received=192.168.1.2;branch=z9hG4
bKPjz0KD2mdIf81exOTBIMg2pPgKOBdDB3So
Record-Route: <sip:86.64.162.35;lr;did=1e1.1d09469>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To:
<sip:marco.ciaramella.sip2@ekiga.net>;tag=ekfUIk00ObK0yUy964Jy
QOfAP9MAqTz3
CSeq: 786 INVITE
Contact: <sip:marco.ciaramella.sip2@130.136.4.230:44445;ob>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800;refresher=uac
Require: timer
Content-Type: application/sdp
Content-Length:280
```

La Response INVITE che riceve il proxy interno presenta nell'header Via il riferimento a lui, mentre nell'header Contact il riferimento all'utente che si vuol contattare.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
192.168.1.2:44445;rport=44445;received=127.0.0.1;branch=z9hG4b
KPjxFr-PKZKhaCOlbe8.HU9I9wU38pZy9fq
Record-Route: <sip:86.64.162.35;lr;did=1e1.1d09469>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To:
<sip:marco.ciaramella.sip2@ekiga.net>;tag=ekfUIk00ObK0yUy964Jy
QOfAP9MAqTz3
CSeq: 786 INVITE
Contact: <sip:marco.ciaramella.sip2@192.168.1.2:5060;ob>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800;refresher=uac
Require: timer
Content-Type: application/sdp
Content-Length:   272
```

In uscita la Response INVITE si presenta con un Via che contiene il riferimento al *pjsua* e il Contact con il riferimento a se stesso, in modo da ricevere il flusso RTP/RTCP.

8.2 Outbound proxy

```
INVITE sip:130.136.4.240:5060 SIP/2.0
Via: SIP/2.0/UDP
151.42.10.136:61708;rport;branch=z9hG4bKPjPKcMBzp6yhIZQG-
du1TsYkW1MPmX6L5V
Via: SIP/2.0/UDP
192.168.1.2:44445;rport=44445;received=127.0.0.1;branch=z9hG4b
KPjxFr-PKZKhaCOlbe8.HU9I9wU38pZy9fq
Max-Forwards: 69
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To: <sip:marco.ciaramella.sip2@ekiga.net>
Contact: <sip:marco.ciaramella.sip1@127.0.0.1:44445;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Route: <sip:marco.ciaramella.sip2@ekiga.net>
Content-Type: application/sdp
Content-Length:   478
```

In ingresso al proxy esterno la richiesta di INVITE si presenta, con un header Via che contiene l'indirizzo pubblico con cui esce la rete privata. Mentre nel Contact è presente il riferimento al pjsua mobile.

```
INVITE sip:marco.ciaramella.sip2@ekiga.net SIP/2.0
Via: SIP/2.0/UDP
130.136.4.240:5060;rport;branch=z9hG4bKPjPKcMBzp6yhIZQG-
du1TsYkW1MPmX6L5V
Max-Forwards: 68
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To: <sip:marco.ciaramella.sip2@ekiga.net>
Contact: <sip:marco.ciaramella.sip1@130.136.4.240:5060;ob>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
CSeq: 786 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: PJSUA v2.1 Linux-3.13.0.35/x86_64/glibc-2.15
Content-Type: application/sdp
Content-Length:   476
```

In uscite la richiesta di INVITE presenta negli header Via e Contact il riferimento a se stesso. Aver il riferimento al proxy esterno nell'header Contact vuol dire specificare al pjsua esterno che si vuole il traffico RTP/RTCP su questo proxy, come volevamo ottenere.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
130.136.4.240:5060;rport=5060;received=130.136.4.240;branch=z9
hG4bKPjPKcMBzp6yhIZQG-du1TsYkW1MPmX6L5V
Record-Route: <sip:86.64.162.35;lr;did=1e1.1d09469>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To:
<sip:marco.ciaramella.sip2@ekiga.net>;tag=ekfUIk00ObK0yUy964Jy
QOfAP9MAqTz3
CSeq: 786 INVITE
Contact: <sip:marco.ciaramella.sip2@130.136.4.230:44445;ob>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800;refresher=uac
Require: timer
Content-Type: application/sdp
Content-Length:   280
```

Per quando riguarda la Response INVITE notiamo che il campo Via presenta il riferimento a se stesso, mentre il campo Contact il riferimento al pjsua esterno.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP
151.42.10.136:61708;rport=61708;received=151.42.10.136;branch=
z9hG4bKPjz0KD2mdIf8lexOTBIMg2pPgKOBdDB3So
Record-Route: <sip:86.64.162.35;lr;did=1e1.1d09469>
Call-ID: k3w32SZKTPPYnh1IYQsOkCiDb5min28x
From: <sip:marco.ciaramella.sip1@ekiga.net>;tag=uEiC-AJyCN-P0-
XXUMY.19tBETcxS1qO
To:
<sip:marco.ciaramella.sip2@ekiga.net>;tag=ekfUIk00ObK0yUy964Jy
QOfAP9MAqTz3
CSeq: 786 INVITE
Contact: <sip:marco.ciaramella.sip2@130.136.4.240:5060;ob>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO,
SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800;refresher=uac
Require: timer
Content-Type: application/sdp
Content-Length:   278
```

Infine in uscita dal proxy esterno la INVITE Response si presenterà con il Via che fa riferimento al pjsua mobile e il Contact che fa riferimento a se stesso in modo che il proxy mobile inoltri a lui lo stream RTP/RTCP.

Capitolo 9

Conclusioni e sviluppi futuri

9.1 Discussione sui risultati

Dai risultati che ho mostrato emerge che il comportamento ottenuto rispecchia quello voluto solo in uno scenario di in cui il nodo mobile si muove all'interno di reti pubbliche.
Di fatti quello dai vari test condotti utilizzando il *pjsua* in una rete privata è emerso che i pjsua non utilizzano i proxy per le trasmissioni RTP/RTCP nonostante siano state specificate nei messaggi SDP gli indirizzi su cui inoltrare lo stream RTP/RTCP. Questo comportamento è poco chiaro e si è presentato anche con altri client VoIP.
Il comportamento descritto invece non si è presentato con il *pjsua* in esecuzione in una rete pubblica come quella del laboratorio Ercolani.
Prima di continuare con lo sviluppo dei proxy, sarebbe il caso di verificare il motivo del comportamento anomalo riscontrato nelle reti private.

9.2 Sviluppi futuri

Come ho già accennato, l'algoritmo di decisione per la direzione della INVITE sarebbe da migliorare per supportare le re-INVITE.

Una soluzione potrebbe essere quella di aggiungere nella struttura *src_info* l'informazione sulla posizione dell'utente rispetto al proxy, cioè se interno o esterno.
Tramite questa informazione l'algoritmo quando verificherà la presenza di entrambi gli user *From* e *To*, controllerà anche la posizione di questi in modo da capire in che direzione è diretta la INVITE che ha appena ricevuto.
Inoltre si consiglia di provare i due proxy insieme a client VoIP diversi da quelli della libreria PJSIP in moda da verificare l'affidabilità con software sviluppati da terzi e non strettamente connessi alla suite PJ.

Queste sono le conclusioni.
In queste conclusioni voglio fare un riferimento alla bibliografia: questo è il mio riferimento [**?**, **?**].

Appendice A

Appendice

A.1 *pjsip_rx_data*

```
struct pjsip_rx_data
{

    /**
     * tp_info is part of rdata that remains static for the
    duration of the
     * buffer. It is initialized when the buffer was created by
    transport.
     */
    struct
    {
  /** Memory pool for this buffer. */
  pj_pool_t    *pool;

  /** The transport object which received this packet. */
  pjsip_transport    *transport;

  /** Other transport specific data to be attached to this
    buffer. */
  void       *tp_data;

  /** Ioqueue key. */
```

```
pjsip_rx_data_op_key    op_key;

  } tp_info;


  /**
   * pkt_info is initialized by transport when it receives an
 incoming
   * packet.
   */
  struct
  {
/** Time when the message was received. */
pj_time_val     timestamp;

/** Pointer to the original packet. */
char        packet[PJSIP_MAX_PKT_LEN];

/** Zero termination for the packet. */
pj_uint32_t     zero;

/** The length of the packet received. */
pj_ssize_t      len;

/** The source address from which the packet was received. */
pj_sockaddr     src_addr;

/** The length of the source address. */
int       src_addr_len;

/** The IP source address string (NULL terminated). */
char        src_name[PJ_INET6_ADDRSTRLEN];

/** The IP source port number. */
int       src_port;

  } pkt_info;
```

```
  /**
   * msg_info is initialized by transport mgr (tpmgr) before
  this buffer
   * is passed to endpoint.
   */
  struct
  {
/** Start of msg buffer. */
char      *msg_buf;

/** Length fo message. */
int       len;

/** The parsed message, if any. */
pjsip_msg    *msg;

/** Short description about the message.
 *  Application should use #pjsip_rx_data_get_info() instead.
 */
char      *info;

/** The Call-ID header as found in the message. */
pjsip_cid_hdr    *cid;

/** The From header as found in the message. */
pjsip_from_hdr     *from;

/** The To header as found in the message. */
pjsip_to_hdr     *to;

/** The topmost Via header as found in the message. */
pjsip_via_hdr    *via;

/** The CSeq header as found in the message. */
pjsip_cseq_hdr     *cseq;

/** Max forwards header. */
```

```
pjsip_max_fwd_hdr *max_fwd;

/** The first route header. */
pjsip_route_hdr   *route;

/** The first record-route header. */
pjsip_rr_hdr    *record_route;

/** Content-type header. */
pjsip_ctype_hdr   *ctype;

/** Content-length header. */
pjsip_clen_hdr    *clen;

/** "Require" header containing aggregates of all Require
 *  headers found in the message, or NULL.
 */
pjsip_require_hdr *require;

/** "Supported" header containing aggregates of all Supported
 *  headers found in the message, or NULL.
 */
pjsip_supported_hdr *supported;

/** The list of error generated by the parser when parsing
    this message.
 */
pjsip_parser_err_report parse_err;

  } msg_info;


  /**
   * endpt_info is initialized by endpoint after this buffer
 reaches
   * endpoint.
   */
  struct
```

```
    {
  /**
   * Data attached by modules to this message.
   */
  void  *mod_data[PJSIP_MAX_MODULE];

    } endpt_info;

    /* VIC auxiliary info used by our proxies */
    struct
    {
        int     direction; /* 0 unknown, -1 dall'esterno, 1
    dall'interno */
    } auxiliary_pkt_info;

};
```

A.2 *pjsip_tx_data*

```
struct pjsip_tx_data
{
    /** This is for transmission queue; it's managed by
    transports. */
    PJ_DECL_LIST_MEMBER(struct pjsip_tx_data);

    /** Memory pool for this buffer. */
    pj_pool_t   *pool;

    /** A name to identify this buffer. */
    char      obj_name[PJ_MAX_OBJ_NAME];

    /** Short information describing this buffer and the message
    in it.
     *  Application should use #pjsip_tx_data_get_info() instead
    of
```

```
     *  directly accessing this member.
     */
    char    *info;

    /** For response message, this contains the reference to
    timestamp when
     *  the original request message was received. The value of
    this field
     *  is set when application creates response message to a
    request by
     *  calling #pjsip_endpt_create_response.
     */
    pj_time_val    rx_timestamp;

    /** The transport manager for this buffer. */
    pjsip_tpmgr   *mgr;

    /** Ioqueue asynchronous operation key. */
    pjsip_tx_data_op_key op_key;

    /** Lock object. */
    pj_lock_t   *lock;

    /** The message in this buffer. */
    pjsip_msg     *msg;

    /** Strict route header saved by #pjsip_process_route_set(),
    to be
     *  restored by #pjsip_restore_strict_route_set().
     */
    pjsip_route_hdr *saved_strict_route;

    /** Buffer to the printed text representation of the message
    . When the
     *  content of this buffer is set, then the transport will
    send the content
     *  of this buffer instead of re-printing the message
    structure. If the
```

```
     *  message structure has changed, then application must
    invalidate this
     *  buffer by calling #pjsip_tx_data_invalidate_msg.
     */
    pjsip_buffer    buf;

    /** Reference counter. */
    pj_atomic_t    *ref_cnt;

    /** Being processed by transport? */
    int         is_pending;

    /** Transport manager internal. */
    void     *token;

    /** Callback to be called when this tx_data has been
    transmitted. */
    void          (*cb)(void*, pjsip_tx_data*, pj_ssize_t);

    /** Destination information, to be used to determine the
    network address
     *  of the message. For a request, this information is
    initialized when
     *  the request is sent with #
    pjsip_endpt_send_request_stateless() and
     *  network address is resolved. For CANCEL request, this
    information
     *  will be copied from the original INVITE to make sure
    that the CANCEL
     *  request goes to the same physical network address as the
     INVITE
     *  request.
     */
    struct
    {
/** Server name.
 */
pj_str_t      name;
```

```
/** Server addresses resolved.
 */
pjsip_server_addresses   addr;

/** Current server address being tried.
 */
unsigned cur_addr;

  } dest_info;

  /** Transport information, only valid during on_tx_request()
  and
   *  on_tx_response() callback.
   */
  struct
  {
pjsip_transport     *transport;     /**< Transport being used.
   */
pj_sockaddr       dst_addr;       /**< Destination address. */
int         dst_addr_len;  /**< Length of address. */
char          dst_name[PJ_INET6_ADDRSTRLEN]; /**< Destination
 address. */
int         dst_port;       /**< Destination port.  */
  } tp_info;

  /**
   * Transport selector, to specify which transport to be used
 .
   * The value here must be set with
 pjsip_tx_data_set_transport(),
   * to allow reference counter to be set properly.
   */
  pjsip_tpselector      tp_sel;

  /**
   * Special flag to indicate that this transmit data is a
 request that has
```

```
     * been updated with proper authentication response and is
    ready to be
     * sent for retry.
     */
    pj_bool_t        auth_retry;

    /**
     * Arbitrary data attached by PJSIP modules.
     */
    void         *mod_data[PJSIP_MAX_MODULE];

    /**
     * If via_addr is set, it will be used as the "sent-by"
    field of the
     * Via header for outgoing requests as long as the request
    uses via_tp
     * transport. Normally application should not use or access
    these fields.
     */
    pjsip_host_port           via_addr;       /**< Via address.
         */
    const void               *via_tp;         /**< Via transport.
          */
};
```

Bibliografia

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 3986, uniform resource identifier (uri): Generic syntax, 2005.

[2] V. Ghini, S. Ferretti, and F. Panzieri. always best packet switching for sip services. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:805–810, 2012.

[3] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[4] C. Huitema. Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP). RFC 3605 (Proposed Standard), October 2003.

[5] D. Johnson, C. Perkins, and J. Arkko. Mobility support in ipv6. RFC 3775 (Proposed Standard), June 2004.

[6] Rolf Oppliger. Internet security: Firewalls and beyond. *Commun. ACM*, 40(5):92–102, May 1997.

[7] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol. Technical report, IETF, 2002.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 3550: RTP: A Transport Protocol for Real-Time Applications. Technical report, IETF, 2003.

Ringraziamenti

Ringrazio i miei genitori che hanno permesso la mia formazione nelle discipline informatiche.