Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática

2008

Tiago André Rodrigues e Silva Rés

Sistema de conversação em redes IP

Universidade de Aveiro
2008

Departamento de Electrónica, Telecomunicações e Informática

Tiago André Rodrigues e Silva Rés

Sistema de conversação em redes IP

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e de Telecomunicações, realizada sob a orientação científica do Dr. António Navarro e pelo Dr. Paulo Salvador, ambos Professores Auxiliares do Departamento de Electrónica, Telecomunicações e Informática.

Este trabalho é dedicado à memória da minha Mãe.

O júri

Presidente

Doutor João Nuno Pimentel da Silva Matos
Professor Associado da Universidade de Aveiro

Vogal

Doutor António José Nunes Navarro Rodrigues (Orientador)
Professor Auxiliar da Universidade de Aveiro

Vogal

Doutor Paulo Jorge Salvador Serra Ferreira (Co-orientador)
Professor Auxiliar da Universidade de Aveiro

Vogal

Doutor Joel José Puga Coelho Rodrigues
Professor Auxiliar do Departamento de Informática da Faculdade de Ciências da Engenharia da Universidade da Beira Interior

Agradecimentos

Gostava de agradecer a todas as pessoas que de uma forma directa ou indirecta me ajudaram neste trabalho e no decorrer da minha vida académica.

Agradeço em particular:

- Aos meus orientadores pelo seu apoio e orientação, principalmente na recta final do trabalho.
- Agradeço à minha família pelo seu apoio.
- Agradeço aos meus amigos e namorada por me terem proporcionado momentos inesquecíveis.

palavras-chave

Voz sobre IP, VoIP, SIP, Telefonia, OpenSER, Asterisk, Tarifação, CDRTool.

Resumo

A necessidade da integração de redes telefónicas com redes de dados levou ao aparecimento da tecnologia de voz sobre IP (*Internet Protocol*). Desde então a sua evolução tem sido notável, uma vez que esta tecnologia traz bastantes vantagens a nível económico, de simplicidade de infra-estrutura, escalabilidade e aumento de desempenho.

Este trabalho visa o estudo da voz sobre IP e dos seus protocolos. Foi neste contexto que o protocolo SIP (*Session Initiation Protocol*) foi estudado em maior detalhe, já que as empresas especializadas estão a convergir os seus serviços baseando-se nele, para poder satisfazer as necessidades dos clientes e aumentar os serviços. A popularidade do SIP deve-se sobretudo à sua simplicidade e flexibilidade.

Outra vertente do trabalho foi a implementação de uma rede proprietária de voz sobre IP, onde se inclui o servidor, integração com a linha telefónica pública, gestão de utilizadores, ferramentas de taxação de chamadas e a implementação de um cliente SIP.

Foi idealizada a arquitectura para o sistema, sendo apresentadas algumas capturas de pacotes que demonstram o funcionamento do serviço.

Keywords Voice over IP, VoIP, Telephony, SIP, OpenSER, Asterisk, Billing, CDRTool.

Abstract

The need to integrate telephonic networks with data networks led to the appearance of the voice over IP (Internet Protocol) technology. Since then its evolution has been remarkable, since it brings many advantages in an economic level, infrastructure simplicity, scalability and performance improvement.

This work focuses the study of this technology and its protocols. It was in this context that SIP protocol (Session Initiation Protocol) was studied with more detail, once the service providers are converging their services to meet the costumer's needs and increasing the services. SIP's popularity is mainly due to its simplicity and flexibility.

Another angle of the work was the implementation of a proprietary voice over IP network that includes the server, the public telephone line integration, user management, billing tools and the implementation of a SIP client.

The architecture for the system was idealized, being shown some packet captures that demonstrate the functioning of the service.

Índice

Índice de figuras

Índice de tabelas

Acrónimos

AAA - *Autenthication, Authorization and Accounting.*

ASCII - *American Standard Code for Information Interchange.*

BSD – *Berkeley Software Distribution.*

B2BUA – *Back to Back User Agent.*

CDR – *Call Detail Record.*

Codec – *Coder decoder.*

CSMA/CD – *Carrier Sense/Multiple Access with Collision Detection.*

DNS – *Domain Name Server.*

FTP - *File Transfer Protocol.*

FXO . *Foreign Exchange Office.*

FXS - *Foreign Exchange Subscriber.*

GPL – *General Public License.*

HTTP - *Hypertext Transfer Protocol.*

IETF - *Internet Engineering Task Force.*

IP – *Internet Protocol.*

ISO – *International Standards Organization.*

ITU - *International Telecommunication Union.*

LAN - *Local Area Network.*

LGPL – *Lesser GPL.*

MAC – *Media Access Control.*

MTA - *Message Transfer Agent.*

NAT – *Network Address Translation.*

NFS - *Network File Services.*

OSI – *Open System Interconnection.*

PBX - *Private Branch Exchange.*

PSTN – *Public Switched Telephone Network.*

QoS – *Quality of service.*

RADIUS - *Remote Authentication Dial In User Service.*

RFC – *Request for Comments.*

RTCP – *Real-Time Transport Control Protocol.*

RTP – *Real-Time Transport Protocol*

SIP - *Session Initiation Protocol.*

SDK – *Software Development Kit.*

SDP - *Session Description Protocol.*

TCP – *Transmission Control Protocol.*

UAC – *User Agent Client.*

UAS - *User Agent Server.*

UDP – *User Datagram Protocol.*

URI – *Uniform Resource Identifier.*

VoIP – *Voice over IP.*

YATE – *Yet Another Telephony Engine.*

1 Introdução

1.1 Motivação

A evolução da telefonia já passou por diversas fases. Desde os primeiros telefones, que eram ligações dedicadas ponto a ponto, houve grande evolução e, neste momento, o paradigma da tecnologia telefónica está a mudar das redes telefónicas públicas (PSTN – *Public Switched Telephone Network*) para as redes de voz sobre IP (VoIP).

A principal razão para esta mudança está intimamente ligada com razões económicas, pois os custos do VoIP são bastante menores. Além disso, cada vez surgem mais aparelhos que suportam a tecnologia, como é o exemplo de telefones IP que se podem ligar ao computador, ou directamente à rede local, quer através de uma ligação *wireless*, quer via *Ethernet*. Por outro lado, o número de serviços e a própria qualidade têm vindo a aumentar e melhorar.

Os aspectos citados anteriormente são do ponto de vista do utilizador. Contudo do ponto de vista do operador também há algumas vantagens a enumerar, a redução de custos com utilizações de linhas e algumas facilidades ao nível operacional são factores a ter em conta.

Apesar dos aspectos positivos, a tecnologia tem algumas limitações, principalmente no que concerne à largura de banda e ao atraso na rede, que se for superior a um determinado valor torna a conversa imperceptível. Portanto, é justo afirmar que a evolução deste tipo de tecnologia está intimamente ligada à evolução e utilização massiva de banda larga, que tornou os clientes cada vez mais exigentes do ponto de vista de qualidade e quantidade.

As soluções existentes são bastante diversificadas. As características variam consoante o objectivo mas a mais importante é o preço. Um dos objectivos na implementação de um serviço é o baixo custo, portanto é necessário fazer uma pesquisa a nível de equipamentos e *software* para que a rede cumpra este requisito. Neste contexto, existem diversas ferramentas e bibliotecas de código aberto que são bastante fiáveis e robustas, mas não apresentam custos para quem as queira usar e até alterar. Outro aspecto importante é a gestão dos diversos recursos da rede, desde os utilizadores, domínios, *gateways*, entre outros. Portanto o sistema deve apresentar interfaces gráficos e intuitivos para a configuração destes recursos, uma vez que, na maioria dos casos esta envolve o conhecimento de alguma linguagem de

programação que nem todos os administradores do sistema poderão ter. Outra vertente é a integração com a linha telefónica e os custos inerentes, que terão que ser suportados pelos utilizadores numa perspectiva de utilizador pagador, isto é, os que usufruem do serviço devem pagar uma quantia consoante a sua utilização do mesmo. Neste sentido, deverá ser implementada uma ferramenta de contabilidade e taxação do serviço. Finalmente, tem que existir um cliente onde sejam disponibilizados os serviços básicos de conversação.

1.2 Objectivos

Os objectivos desta dissertação passam pelo estudo da voz sobre IP, podendo dessa forma enumerar e explicar os protocolos associados a esta tecnologia, bem como fazer uma apreciação de algumas soluções presentes no mercado. A vertente mais prática do trabalho está patente na implementação de um serviço proprietário de voz sobre IP, nomeadamente num servidor SIP e num cliente. Alguns requisitos do sistema prendem-se com a possibilidade de efectuar chamadas dentro da própria rede, bem como a possibilidade de efectuar chamadas para telefones exteriores. Além disto, o sistema deve ser capaz de contabilizar e taxar as chamadas.

1.3 Estrutura da dissertação

A dissertação está dividida em quatro blocos, que são os seguintes:

- O capítulo 2 faz um pequeno resumo do que é a telefonia clássica, uma introdução às redes de dados e fala com algum detalhe do que é, de facto, a tecnologia de voz sobre IP.
- O capítulo 3 trata da implementação de um serviço na prática, desde a instalação de um servidor até à criação de um cliente, tudo isto usando *software* de código aberto e bibliotecas livres.
- O capítulo 4 demonstra o funcionamento do serviço principalmente através da captura de pacotes na rede.
- O capítulo 5 apresenta as conclusões finais sobre o trabalho e fala do que pode ser o trabalho futuro a desenvolver.

2 Voz sobre IP

Este capítulo pretende fazer uma pequena resenha do que é a telefonia em geral, uma breve descrição do funcionamento de redes de dados, nomeadamente o protocolo IP (*Internet protocol* – Protocolo de internet) e uma visão esclarecedora do que é voz sobre IP. Neste último ponto são tratadas as motivações para o aparecimento da tecnologia, as suas limitações e os seus protocolos. Dentro dos protocolos, o mais retratado será o SIP (*Session Initiation Protocol*), nomeadamente as suas características, vantagens em relação aos outros e potencialidades.

2.1 Rede telefónica (PSTN)

A rede telefónica, tal como a conhecemos hoje em dia, sofreu diversas transformações ao longo dos tempos. A sua descoberta foi atribuída a *Alexander Bell* (1876) [1], que conseguiu reproduzir num receptor o som enviado num emissor através de um par de fios de cobre. Nesta altura apenas era possível a comunicação entre dois pontos específicos, o que não era um modelo muito escalável.

Visto isto, impunha-se a criação de uma central telefónica, para que se pudesse efectuar a chamada entre vários clientes sem que fosse necessário ter um número de terminais igual ao número de possíveis pessoas a ligar. Inicialmente, esta interacção era efectuada por um operador humano numa central, que fazia a comutação de linhas de uma forma manual, consoante o destino da chamada, indicado pela pessoa que a efectuava. Este modelo melhorou a escalabilidade do sistema, contudo continuava limitado devido à necessidade de intervenção humana. A evolução tecnológica permitiu a substituição destes operadores por máquinas (*switches*) que permitiram que a velocidade da comutação fosse maior, bem como levou a um aumento de capacidade da rede. Deste modo, temos uma rede telefónica com comutação de circuitos, como se pode ver na Figura 2.1. O conceito base é a hierarquia, uma vez que a comutação será efectuada consoante o destino e se este for mais afastado, maior número de *switches* são usados, bem como a sua complexidade será maior. O cliente tem um telefone com uma ligação dedicada à central local, assim como outros clientes na mesma área geográfica. Deste modo, se clientes na mesma área pretendem comunicar entre si, a chamada é efectuada apenas na mesma rede de acesso. Caso sejam áreas geográficas distintas, têm que se ligar a centrais de longa distância (também chamados como *switches* de classe 4 [1]), que estão ligadas com outras centrais idênticas através de canais com grandes

capacidades. Para chamadas internacionais é necessário outro nível de hierarquia que vai ligar as centrais dos diferentes países. Neste momento apenas se está a falar das portadoras do sinal de *Media*, contudo a PSTN consiste de duas redes distintas. São elas a rede de sinalização e a rede Media em si. Esta sinalização é responsável pelos tons ouvidos quando se pega no telefone, chamada é terminada, entre outros processos, assegurando o bom funcionamento da rede telefónica, mas não será tratada em pormenor.

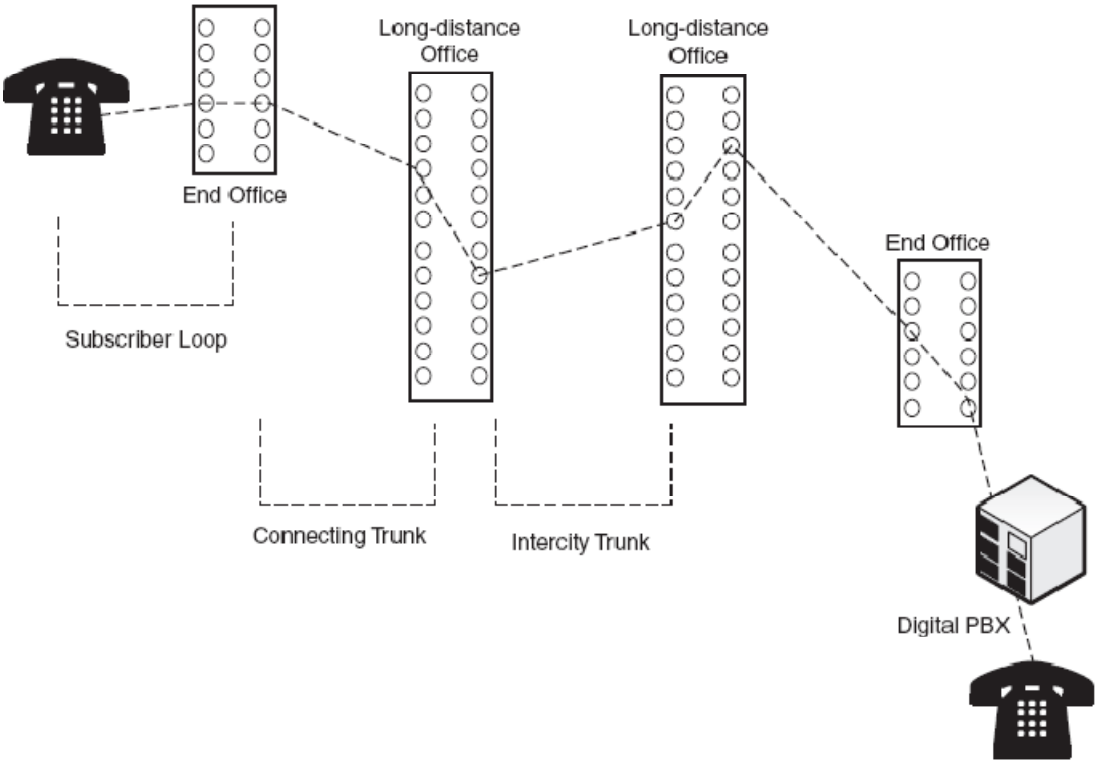


Figura 2.1- Arquitectura de alto nível da PSTN [1]

Como se pode ver na Figura 2.1, é criado um circuito físico entre os dois utilizadores, usando várias centrais telefónicas para o conseguir. É por esta razão que este tipo de rede é chamada rede de comutação de circuitos. Em oposição a isto, temos as redes de comutação de dados por pacotes, que será tratada de seguida.

2.2 Redes de dados

As redes de dados têm um funcionamento diferente das redes de comutação de circuitos. São, como referido anteriormente, comutadas por pacotes. Isto significa que a informação a transmitir é segmentada em partes mais pequenas, cada um devidamente endereçado. Cada parte é enviada e encontra de uma forma dinâmica o seu destino. Isto significa que diferentes pacotes podem tomar diferentes caminhos, sendo que a informação será reconstruída no receptor. Os parâmetros da escolha dos caminhos são configurados nas máquinas responsáveis pelo seu encaminhamento,

tipicamente *routers* ou *switches* e podem ser, a título de exemplo, a latência ou o custo de utilização da linha.

Para descrever as redes de dados por pacotes, a ISO (*International Standarts Organization*) propôs um modelo por camadas que descreve o modo como se deve transmitir dados entre dois pontos numa rede, qualquer que seja o hardware usado. Esse modelo foi chamado OSI (*Open Systems Interconnection*) e está ilustrado na Figura 2.2.

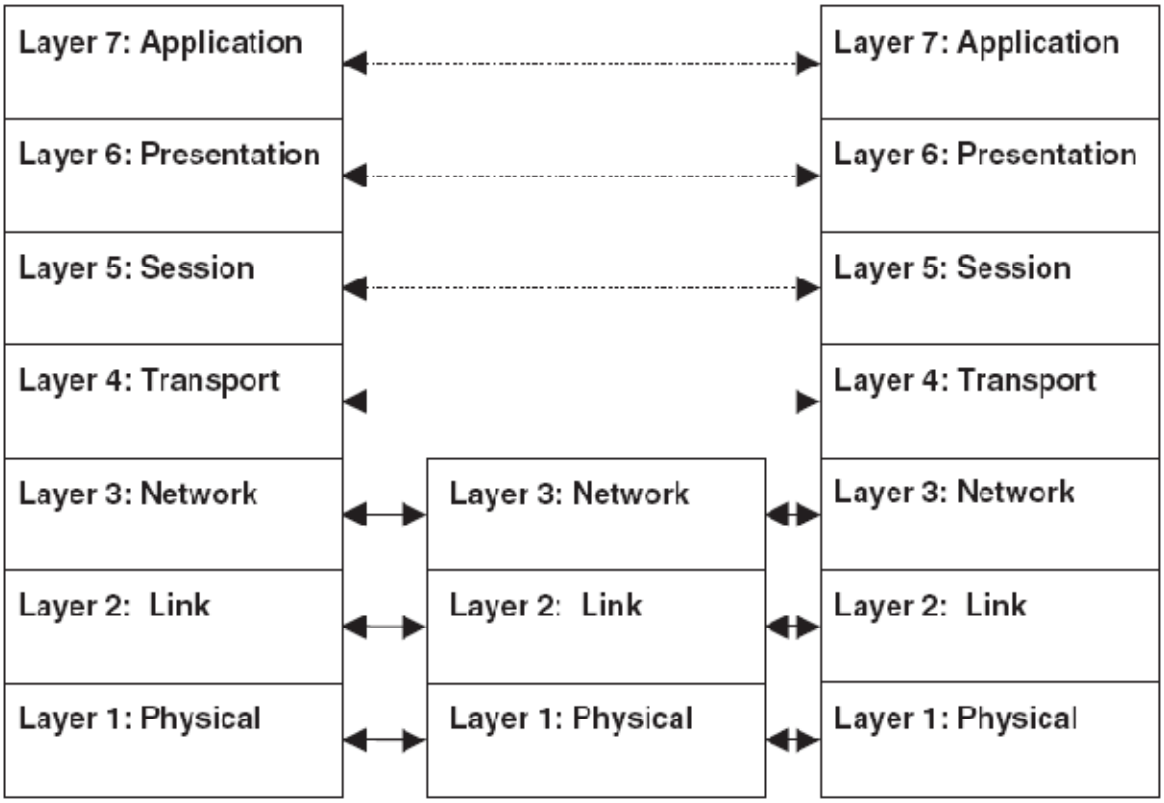


Figura 2.2 - Modelo OSI de 7 camadas [1]

Nesta imagem, além da ilustração do modelo por camadas OSI, está ilustrado o modo de operação em rede, onde se vê 2 clientes e um dispositivo (*router*) entre eles. Este dispositivo é de camada 3, isto é, opera na camada de rede. O modelo pressupõe que protocolos de cada nível comuniquem com o mesmo nível no outro *host*, de uma forma hierárquica.

De seguida será feita uma breve apresentação de cada camada.

2.2.1 Camada física

É responsável por gerir a transferência física da informação sobre os meios disponíveis de transmissão, definindo os mecanismos para o fazer.

Exemplo: O protocolo *Ethernet* define tipos de modulação, ordenação de dados, velocidades de transmissão, etc.

2.2.2 Camada de ligação lógica

Também conhecida como camada de enlace de dados, tem como função gerir a transferência da informação através do canal de transmissão, nomeadamente como acede e partilha a camada física. Esta gestão inclui o controlo do fluxo da informação na rede, a sua sincronização e garantia da entrega. É de destacar a detecção e correcção de erros de transmissão. Esta camada tem uma subcamada, chamada de MAC (*Media Access Control* – Controlo de Acesso ao Meio)

Exemplo: CSMA/CD (*Carrier Sense/Multiple Access with Collision Detection*), que é um protocolo usado em *Ethernet* para controlar o acesso ao meio evitando colisões de dados.

2.2.3 Camada de rede

Esta camada vai ser responsável pelo encaminhamento e endereçamento da informação, quer na estação que emite os dados, quer no respectivo destino.

Exemplo: Protocolo IP, que será estudado com mais detalhe.

2.2.4 Camada de transporte

A camada de sessão proporciona o interface entre as três camadas superiores e as três inferiores. Na transmissão, aceita os dados de camadas acima, divide em blocos mais pequenos e transmite para a camada abaixo, podendo o utilizador abstrair-se dos aspectos funcionais e físicos da rede. Na recepção, vai ocorrer o contrário: vai receber os segmentos de dados da camada de rede, vai juntá-los e vai entregar à camada de sessão. Garante, ainda, o controlo de congestão, isto é, verifica que não estão a ser enviados mais dados num instante de tempo do que a rede pode suportar.

Exemplo: TCP (*Transmission Control Protocol*), preocupado principalmente com a entrega fiável dos dados, e UDP (*User Datagram Protocol*) que não se preocupa com essa entrega fiável, deixando isso para protocolos de camadas superiores.

2.2.5 Camada de sessão

Tem como função apresentar ao utilizador um *interface* genérico da rede. Ela trata da forma como as outras camadas têm acesso à rede, como são autenticadas, como descobrem outros utilizadores e como a sessão é configurada e gerida.

Exemplo: NFS *(Network File System*) é um protocolo de partilha de ficheiros.

2.2.6 Camada de apresentação

A sua preocupação principal é a sintaxe, ou seja, o modo como a informação a transmitir é representada. Outras funções incluem a compressão de dados e encriptação.

Exemplo: Formato de dados ASCII (*American Standard Code for Information Interchange*) que é um formato definido e conhecido, permitindo que os dados sejam apresentados do mesmo modo em ambos os intervenientes da transmissão.

2.2.7 Camada de aplicação

A camada mais alta do modelo é responsável pelo suporte das aplicações do utilizador. Além disto, define a semântica da informação a transmitir/receber, ou seja, o seu significado.

Exemplo: FTP (*File Transfer Protocol*) é um protocolo para transferência de dados.

2.3 Protocolo IP

O Protocolo IP (*Internet Protocol*) é definido na norma RFC791 do IETF (*Internet Engineering Task Force*) e pode designar-se como a espinha dorsal do que chamamos internet. Todos os pacotes que são encaminhados pela internet usam-no. É um protocolo não orientado à ligação onde cada pacote IP é tratado independentemente, não havendo qualquer relação com outro. Ele é responsável pela comunicação entre cada elemento da rede para permitir o transporte de uma mensagem de um *host* de origem até a um *host* de destino, podendo o datagrama passar por várias sub-redes (a origem e o destino são hosts identificados por endereços IP)[2].

Este protocolo é não confiável, cabendo essa responsabilidade aos protocolos das camadas superiores, nomeadamente do TCP. Deste modo, nenhum mecanismo de controlo de fluxo ou de controlo de erros de dados são usados, verificando-se apenas a integridade do cabeçalho através de um *checksum* (soma de controlo), de modo a assegurar que os datagramas são encaminhados devidamente.

O protocolo IP tem como funções mais importantes a de atribuição de um esquema de endereçamento independente do endereçamento da rede utilizada e independente da topologia da rede. Além disso, devem ter a capacidade de encaminhar e tomar as respectivas decisões de encaminhamento para o transporte das mensagens entre os elementos que interligam as redes.

Aos dados é acrescentado um cabeçalho IP, resultando um datagrama IP. O cabeçalho terá no mínimo 20 *bytes*. A Figura 2.3 mostra a constituição do datagrama IP, mas este não será estudado em detalhe.

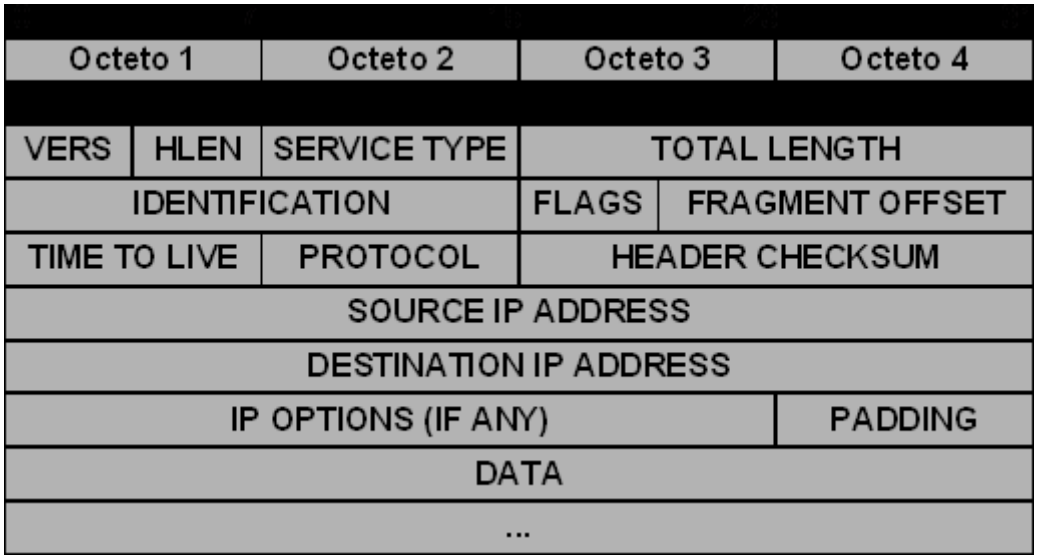


Figura 2.3 - Formato de um datagrama IP [2]

Este datagrama não é alterado desde a origem até ao destino podendo, quanto muito, sofrer fragmentação.

2.3.1 TCP e UDP

Uma vez que o protocolo IP por si só não oferece garantias, controlo de fluxo, entre outras, estão a si associados dois protocolos de transporte: TCP e UDP. A sua descrição está adiante.

2.3.1.1 TCP

É orientado à conexão e confiável, pois existe a garantia da transmissão integral dos dados para os *hosts* de destino correctos e na sequência correcta. Quando ocorrem erros, caso a informação seja impossível de recuperar (através do *checksum*) ou caso o pacote TCP/IP se tenha perdido durante a transmissão, TCP garante a retransmissão do mesmo. Esta informação é possível mediante o envio por parte do destino de uma mensagem de "*acknowledgement*". Para que seja possível identificar a que serviço um determinado datagrama pertence, o TCP utiliza o conceito de portas. A cada porta está associado um serviço. Após determinada a porta, toda a comunicação com a aplicação é realizada e endereçada através dela [2].

2.3.1.2 UDP

É um protocolo mais leve e acrescenta muito pouco ao protocolo IP. Basicamente pega nos dados e coloca o cabeçalho UDP antes de os passar para a camada

seguinte. A principal vantagem deste protocolo é que, em determinadas aplicações, é mais proveitoso o uso de um protocolo mais leve que permita que os dados cheguem ao seu destino de uma forma muito mais rápida, podendo-se tolerar alguma perda. Esta é a razão para que a voz sobre IP use UDP como transporte.

2.3.2 SDP

SDP (*Session Description Protocol*) está definido na RFC 2327 e é um método de codificação baseado em texto para definir sessões Multimédia. Para os protocolos VoIP que o usam, serve para definir tipos de *stream* áudio que se podem estabelecer entre os intervenientes da chamada e por onde e como vão ser enviados. Isto inclui a negociação do tipo de *codec* a utilizar, endereços IP, protocolo de transporte e porta.

2.3.3 RTP

RTP (*Real-Time Transport Protocol*) está definido na RFC 3550. As aplicações VoIP habitualmente usam RTP e RTCP (*Real-Time Transport Control Protocol*) sobre UDP. RTP providencia um serviço de entrega fim-a-fim, para dados *Media* com características de tempo real. Define informação útil, tal como *timestamp* (define o tempo), número de sequência e marcador, para permitir ao receptor manter a ordem correcta dos pacotes e para a reproduzir correctamente. Isto é especialmente interessante uma vez que, como será visto adiante, existe ruído introduzido em redes IP que pode alterar os pacotes e/ou a sua ordem de chegada.

Este protocolo por si só não tem nenhum mecanismo que garanta a entrega dos pacotes no tempo correcto ou qualquer outro tipo de qualidade de serviço. Para tal é usado o RTCP em conjunto com RTP, de forma a monitorizar a qualidade dos dados entregues.

2.4 Voz sobre IP (VoIP)

Nos pontos anteriores foi falado da rede telefónica PSTN e da rede IP, contudo são duas redes separadas lógica e fisicamente, que foram concebidas com diferentes propósitos e durante bastante tempo existiram separadamente, não sendo permitido nenhum tipo de interacção entre elas, como se pode ver na Figura 2.4. Pode-se falar em convergência destas redes na medida em que os utilizadores podem usar um modem e a sua linha telefónica para ligar à internet, contudo é uma convergência diferente de VoIP já que as linhas telefónicas são usadas para transportar dados. O objectivo desta tecnologia é o transporte de voz e sinalização usando o(s) protocolo(s) IP.

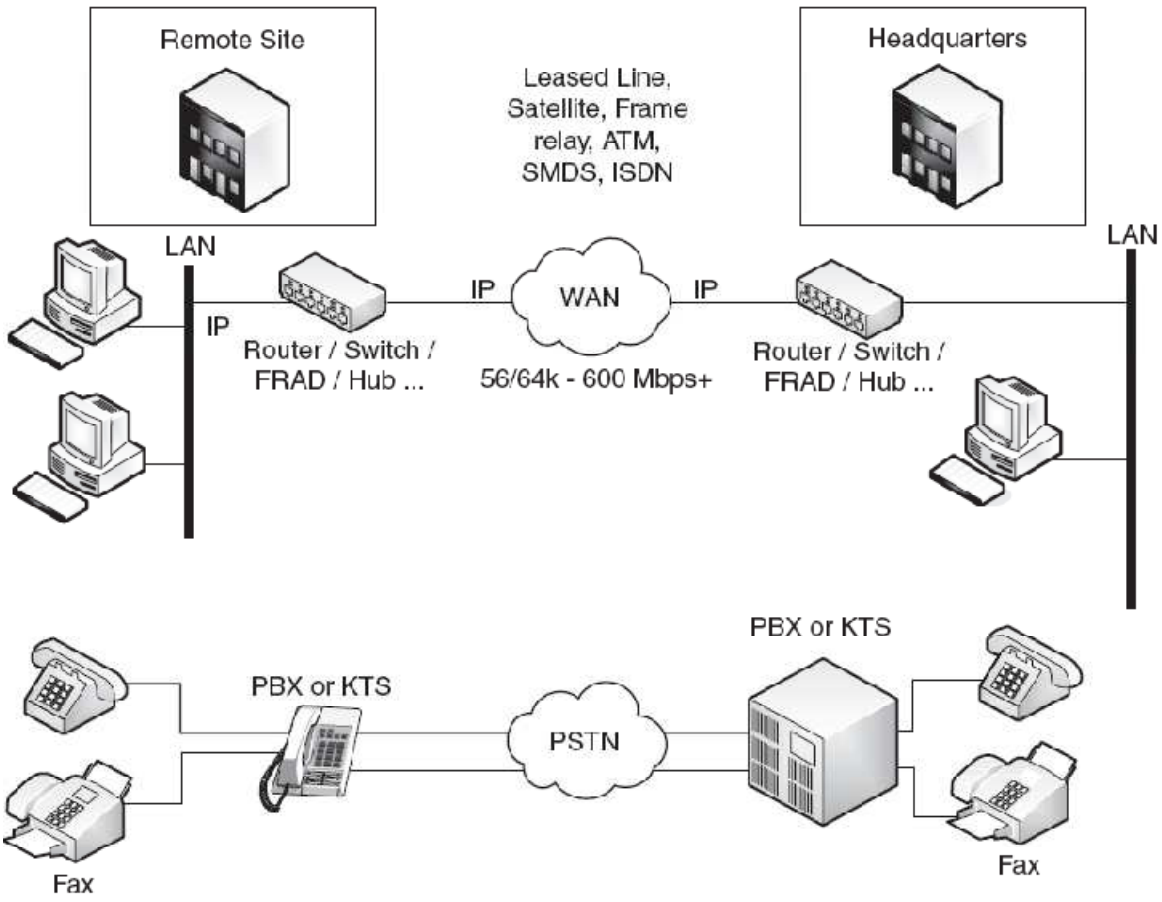


Figura 2.4 - Arquitectura tradicional: Redes de dados e voz separadas [1]

2.4.1 Objectivos do aparecimento da tecnologia

Os objectivos que levaram ao aparecimento da voz sobre IP são, acima de tudo, económicos. Havendo a possibilidade de ultrapassar o custo associado ao uso dos *switches* das centrais telefónicas, o custo das chamadas reduz significativamente. Isto faz mais sentido principalmente com chamadas de longa distância nomeadamente as internacionais. Estes interesses não são apenas do ponto de vista do utilizador final, uma vez que os provedores de serviços telefónicos também podem usufruir de custos menores, podendo aumentar as margens de lucro, contudo é de notar que parte das poupanças que advém de VoIP, são derivadas da falta de legislação em alguns países. Isto permite que os provedores operem usando esta tecnologia sem pagar licenças aos governos.

Outro factor de poupança é a largura de banda. Usando *codecs* de baixa taxa de bit, a voz pode ser comprimida para usar muito menos largura de banda (tipicamente menor que 20 kbps) do que a usada pela PSTN (64kbps) [1]. Usando um método de transporte não orientado à conexão é possível optimizá-la para que não seja necessário reservar largura de banda como no orientadas à conexão, tal como a

PSTN. Assim resulta uma convergência entre as duas redes, como se pode ver na Figura 2.5.

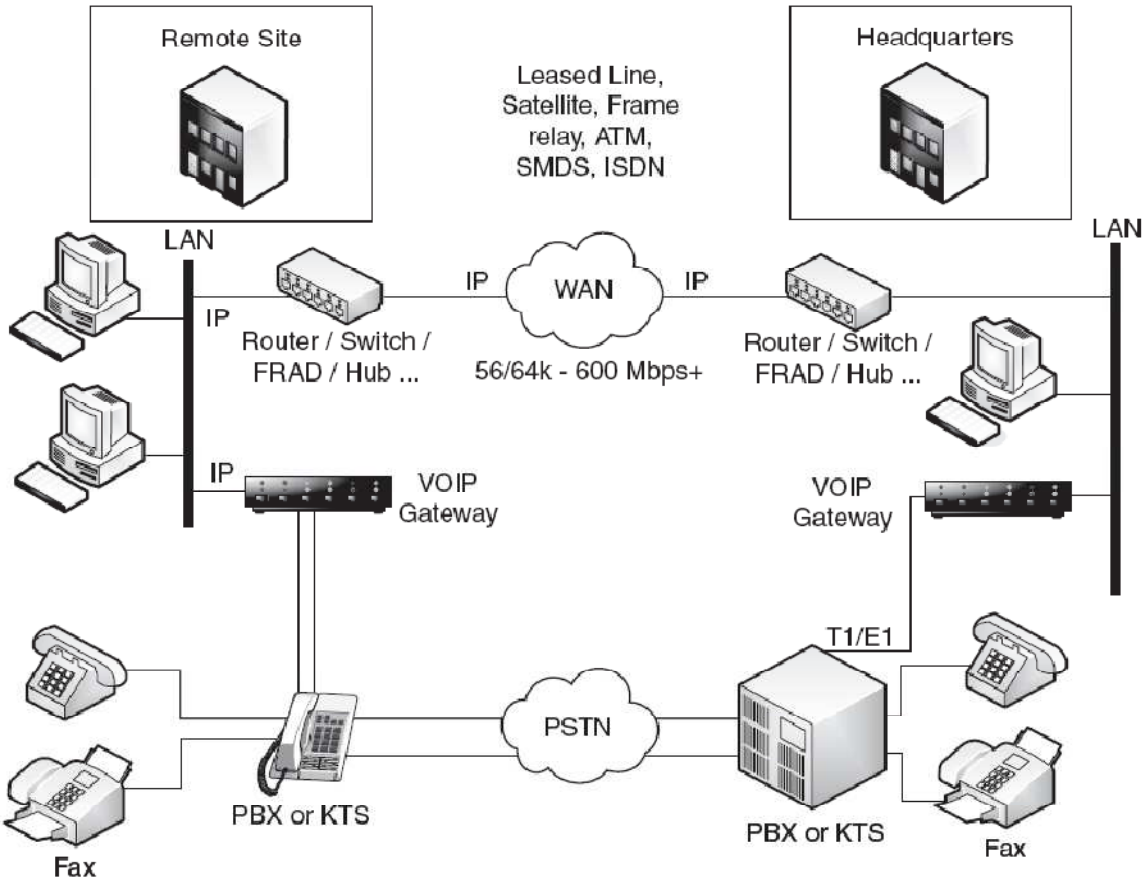


Figura 2.5 - Convergência entre redes telefónica e de dados [1]

Desta forma já é possível, por exemplo, para um utilizador de um computador dentro de uma LAN (*Local Area Network*) efectuar uma chamada para um utilizador residencial ligado à PSTN, denotando a convergência de que se referiu anteriormente. Na Figura 2.6 está uma configuração geral típica de um sistema VoIP com sistemas baseados em IP e dispositivos PSTN.

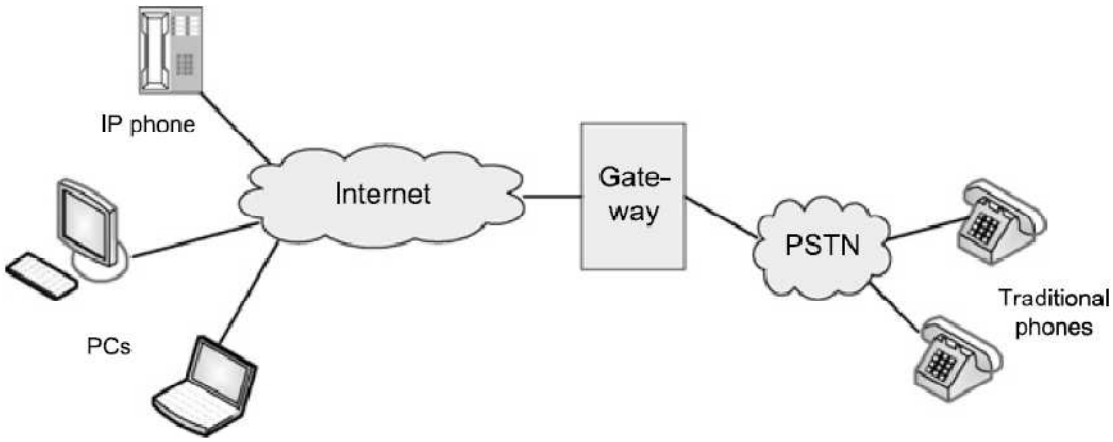


Figura 2.6 - Configuração típica de um sistema VoIP com sistemas baseados em IP e dispositivos tradicionais PSTN [3]

2.4.2 Limitações da tecnologia

Apesar dos pontos favoráveis apresentados anteriormente, existem uns desafios que têm que ser ultrapassados de forma a tornar a Voz sobre IP uma realidade. Os principais são o atraso, a perda de pacotes e a largura de banda disponível [1].

2.4.2.1 Atraso e jitter

O atraso é o tempo que um pacote demora para chegar a um determinado receptor, depois de ter sido transmitido. Esse atraso resulta de dois componentes: atraso fixo de rede e atraso variável de rede. O atraso variável é causado por filas (*queues*) que podem congestionar e por atrasos na ordenação, ao passo que o atraso fixo é provocado pelo processo a que o sinal é sujeito antes de ser transmitido, que será visto na secção 2.4.3. A ITU define o limite de 150 ms como limite aceitável para ter alta qualidade de voz [28].

O jitter é a variação do atraso, ou seja, a variação entre o tempo esperado de chegada do pacote e o tempo em que efectivamente ele chega. Os congestionamentos na rede podem acontecer em qualquer momento, o que fará com que os *buffers* encham de uma forma instantânea, provocando este fenómeno. Este tipo de ruído é um grande problema em VoIP [28].

2.4.2.2 Perda de pacotes

Apesar da perda de pacotes não ser algum muito desejável em comunicações em tempo real, esta é um pouco tolerável (<5% [1]) em VoIP. Isto deve-se ao facto da comunicação com voz apresentar muita informação redundante, portanto pode tolerar algumas perdas. Este valor tolerável não é completamente mensurável, uma vez que pode variar consoante o algoritmo utilizado para codificar a voz (*codec*). Os *codecs* não passam de modelos matemáticos capazes de amostrar *streams* de voz e codificá-los. Na

Tabela 2.1 está uma lista de *codecs* e respectivos *bitrates* (taxa de bit) usados em VoIP [4].

Tabela 2.1 - Lista de *codecs* de voz

Codec	Bitrate (kbps)	Nível de processamento computacional
G.711	64	Baixo
G.726	12, 24 ou 32	Baixo
G.723.1	5.3 ou 6.3	Alto
G.729[a]	8	Alto
GSM	13	Baixo
iLBC	13.3 ou 15.2	Alto
Speex	Variável de 2.15 a 22.4	Alto

2.4.2.3 Largura de banda

A largura de banda utilizada deverá ser a mais pequena possível, apesar de hoje em dia a evolução fazer com que a disponível seja cada vez maior. Ela vai depender directamente do tipo de *codec* utilizado e do tempo de empacotamento da informação. Como é obvio, largura de banda custa dinheiro, portanto quanto menor for, menos custos terá a comunicação.

2.4.3 Processo

Para obter um sinal de voz e enviá-lo para outro utilizador é necessário seguir um processo. Na Figura 2.7 estão os passos envolvidos nesse mesmo processo e de seguida haverá uma pequena descrição dos mesmos.

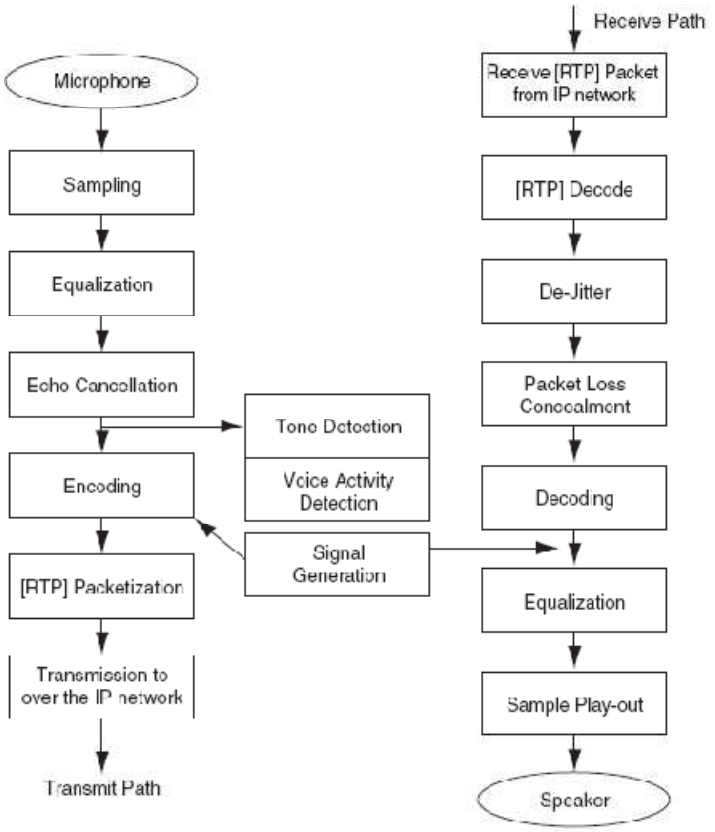


Figura 2.7 – Passos envolvidos em VoIP [1]

2.4.3.1 Amostragem

Trata-se da conversão do sinal de voz captado analogicamente pelo microfone em amostras digitais.

2.4.3.2 Equalização

Trata-se do ajuste dinâmico das amostras para melhorar certas limitações conhecidas do microfone.

2.4.3.3 Cancelamento de eco

Procura remover os resíduos de voz recebidos no emissor, que foram emitidos por acidente no receptor

2.4.3.4 Codificação

As amostras de som são agrupadas em frames de voz de duração dependente do *codec* a utilizar, e são codificadas.

2.4.3.5 Empacotamento RTP

Combinação de vários frames num pacote para transmitir. Quanto maior o período de empacotamento, maior a eficiência do sistema, contudo pode aumentar o atraso. O valor comum para este período é 20 ms [1]

Do lado do receptor o processo é semelhante, mas obviamente inverso. De destacar as operações para contrariar o *jitter* e as perdas de pacotes, que são necessárias devido aos problemas destacados em 2.4.2.

2.5 Protocolos usados em voz sobre IP

Este capítulo vai tratar, de uma forma geral, dos protocolos utilizados em VoIP, falando em pormenor do protocolo SIP (*Session Initiation Protocol*). Em primeiro lugar serão vistos os protocolos proprietários, de seguida os protocolos abertos.

2.5.1 A necessidade de protocolos

Como foi visto no capítulo anterior, há grandes desafios que têm que ser ultrapassados para que possa haver comunicação de voz em tempo real pela internet. O sinal tem que chegar ao destino essencialmente semelhante ao original, bem como com um atraso que torne possível a percepção e seja possível ter uma conversação.

Os protocolos de transporte não foram originalmente pensados para comunicações em tempo real [5]. Os terminais teriam que resolver problemas com pacotes perdidos, através do aumento de tempos de espera, ou do pedido de retransmissão. Como é óbvio, estas soluções não vão ser úteis quando falamos em conversações de voz, pois a perda de palavras e o atraso podem torná-las imperceptíveis. A forma como as pessoas falam é incompatível com a forma como o transporte de dados é feito em IP.

Visto não haver protocolos que encaixassem à medida das necessidades específicas deste tipo de dados, surgiu a necessidade de criar novos.

2.5.2 Protocolos VoIP

Tal como na telefonia "clássica", onde havia protocolos de sinalização de forma a criar, manter e destruir as chamadas telefónicas, também terá que haver em VoIP. Haverá necessidade de criar algum mecanismo para iniciar, manter e acabar com os fluxos de dados entre os intervenientes de uma chamada em telefonia IP. Os protocolos Skinny/SCCP (*Skinny Client Control Protocol*) e UNISTIM (*Unified Networks IP Stimulus*) são protocolos proprietários, ao passo que H.323, IAX (*Inter-Asterisk Exchange Protocol*), MGCP (*Media Gateway Control Protocol*), Megaco(H.248) e SIP (*Session Initiation Protocol*) são abertos.

2.5.2.1 Skinny/SCCP

Este protocolo é proprietário do equipamento VoIP da *Cisco* mais antigo, usado para a comunicação entre o *Cisco Call Manager* e os telefones VoIP da mesma empresa. Actualmente esta empresa já usa SIP nos seus produtos. As mensagens *Skinny* são transportadas usando TCP e usam a porta 2000 [6].

2.5.2.2 UNISTIM

Este protocolo é propriedade da *Nortel* e a sua finalidade é a comunicação entre telefones e PBX da mesma marca. É um protocolo de mais baixo nível que a maioria dos protocolos VoIP, considerando o telefone como um mero terminal, deixando a parte mais inteligente ao nível do *switch*. Isto significa que o telefone sinaliza, por exemplo, pressionamento de teclas, deixando para o *switch* a função de perceber o que essa tecla significa. Ele usa RTP para transportar Áudio, mas a sua sinalização usa a porta 5000 (tipicamente) e o protocolo UDP como transporte [7].

2.5.2.3 IAX

Este protocolo foi desenvolvido pela *Digium* para a comunicação entre servidores *Asterisk*. O *Asterisk* é um PBX (*Private Branch Exchange*) que será tratado com maior pormenor adiante. Apesar de ter sido criado para este *software*, é um protocolo aberto e usado por bastantes mais projectos em telecomunicações. Ele usa a porta UDP 4569 [5] para sinalização e transporte *Media*. Actualmente já houve uma actualização para IAX2, deixando o antigo de ser suportado, contudo a designação é a mesma.

2.5.2.4 MGCP

O protocolo MGCP (*Media Gateway Control Protocol*) foi introduzido pela IETF e está definido na RFC 3435 [5]. O seu objectivo é fazer os dispositivos terminais (como telefones) o mais simples possível, passando toda a lógica e processamento pelas mãos dos *gateways*. Trata-se de um modelo centralizado, sendo impossível a telefones MGCP ligar entre si directamente, tendo sempre que passar por um dispositivo de controlo. Este protocolo define 2 tipos de componentes: *Gateway* remoto e servidor de gestão de chamadas. As mensagens são texto ASCII e são enviadas por UDP.

2.5.2.5 Megaco (H.248)

É um protocolo muito parecido com o MGCP, na medida em que o seu modelo também é centralizado e resultou de um trabalho conjunto entre o IETF e o ITU. O seu conceito base é a existência de um gestor de chamadas que é o ponto central dos terminais Media. Ele é baseado em texto, tal como o MGCP, mas com diferente

sintaxe. Utiliza UDP e TCP e usa SDP (*Session Descripion Protocol*) para descrever as sessões *Media*.

Megaco tem dois componentes básicos, que são as terminações e os contextos. As terminações são fontes de *Media* (e.g. porto RTP), ao passo que os contextos são definidos como uma mistura de terminações.

2.5.2.6 H.323

Este protocolo foi introduzido pela ITU (*International Telecommunication Union*) pensado originalmente para o mecanismo de transporte para o suporte de videoconferência. Esta recomendação procura especificar comunicações multimédia que não proporcionam qualidade de serviço (QoS). Na realidade, é bastante popular neste tipo de comunicação, levando uma certa vantagem em relação aos outros protocolos, uma vez que já existem bastantes soluções implementadas baseadas em H.323.

H.323 é um protocolo distribuído com os seguintes elementos estruturais: Dispositivo terminal, *Gateway* (faz o interface entre redes H.323 e outras), *Gatekeeper* (é responsável por alguns serviços, tal como a tradução de números telefónicos para endereços IP, o controlo de largura de banda, entre outras) e Unidade de controlo multiponto (gere as conferências multiponto).

Este protocolo é chamado um padrão guarda-chuva (*umbrella standard*), uma vez que refere uma série de outros padrões para descrever o próprio padrão. Como se pode ver na Tabela 2.2, existe uma série de padrões para definir a codificação do áudio e vídeo, a transferência de dados e a forma como o transporte é feito.

Tabela 2.2 - Conjunto de protocolos H.323

Conjunto de protocolos H.323			
Vídeo	Áudio	Dados	Transporte
H.261	G.711	T.122	H.225
H.263	G.722	T.124	H.235
	G.723.1	T.125	H.245
	G.728	T.126	H.450.1
	G.729	T.127	H.450.2
			H.450.3
			RTP
			X.224.0

A sinalização obedece à recomendação H.225 e não é baseada em texto, mas sim em listas codificadas com registos de tipo/duração/valor. O protocolo de controlo é definido por H.245 [1].

2.5.3 SIP

O Protocolo de Iniciação de Sessão (*Session Initiation Protocol* – SIP) foi padronizado pelo IETF e é descrito na RFC3261. Trata-se de um protocolo de camada de aplicação [9], usado para estabelecer, modificar e terminar sessões multimédia. É um protocolo baseado em texto, bastante semelhante ao HTTP (*Hypertext Transfer Protocol*). Hoje em dia é um dos protocolos mais populares em VoIP, estando presente em quase todos os telefones IP do mercado [9]. Esta é a razão principal pela sua adopção neste trabalho, portanto será visto com maior pormenor.

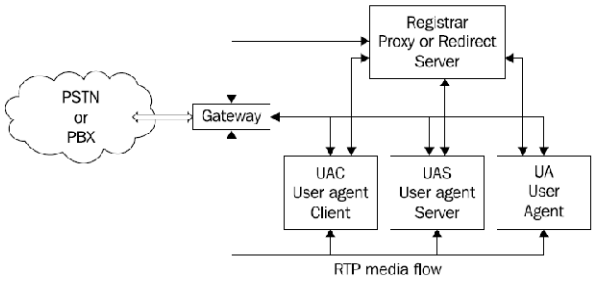


Figura 2.8 - Principais componentes SIP [9]

Tal como o Megaco e MGCP, a sessão Média e o seu controlo são descritos por SDP. A arquitectura pode ser bastante variada, uma vez que é possível fazer uma ligação directa entre dois *hosts*, bem como através de um dispositivo intermédio, conhecido como *proxy* SIP. Um terminal SIP é conhecido como *user agent*. Os elementos SIP estão listados na Tabela 2.3, bem como uma breve descrição da sua função. Na Figura 2.8 está ilustrada a sua arquitectura.

Tabela 2.3 - Elementos SIP e sua função

Componente SIP	Função
User Agent	Trata-se do terminal da comunicação multimédia. Um *user agent* tem dois componentes: *User agent client* (UAC) – responsável por fazer os pedidos das chamadas. *User agent server* (UAS) – responsável por responder às chamadas, enviando respostas. Uma aplicação de telefonia Internet contém, por exemplo, ambos UAC e UAS.
Servidor *Proxy*	Servidor responsável pelo redireccionamento dos pedidos e respostas SIP. Quando ele recebe as requisições SIP, altera os campos "FROM", como se fosse ele o originador da chamada, contudo quando recebe a resposta a essa requisição, encaminha-a para a sua origem original. No caso de co-existirem vários servidores, um *proxy server* pode duplicar uma requisição, enviando cópias para o próximo servidor. Deste modo, uma requisição de início de chamada tenta diversas localizações diferentes, até encontrar o cliente.
Servidor *Redirect*	Recebe requisições e determina um servidor, retornando o endereço do servidor *nexthop* para o cliente. A função primária dos servidores *proxy* e *redirect* é encaminhar chamadas.
Servidor de Registo	Servidor SIP que suporta requisições REGISTER. Elas são usadas para registar as informações dos utilizadores nos servidores.
Servidor de Localização	Armazena e consulta registo dos utilizadores.

SIP foi concebido para ser suficientemente escalável para suportar chamadas simultâneas para um número substancial de utilizadores e para ser extensível o suficiente para incluir mais funções e opções no futuro.

O transporte pode ser efectuado tanto por TCP como UDP. A pilha de protocolos para um telefone IP baseado em SIP está na Figura 2.9.

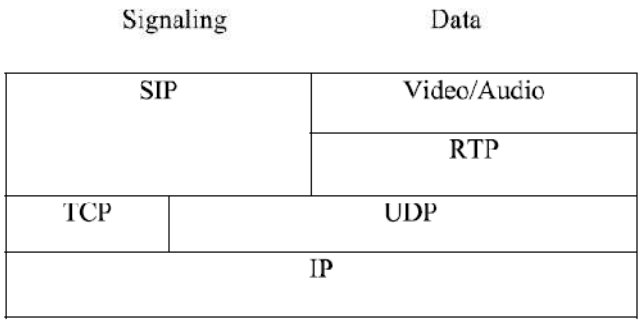


Figura 2.9 - Pilha de protocolos [3]

As principais mensagens de sinalização definidas para SIP estão listadas na Tabela 2.4.

Tabela 2.4 - Métodos de sinalização SIP [1]

Método SIP	Propósito
INVITE	Usado para iniciar uma chamada
UPDATE	Usado para actualizar a descrição da sessão Média durante uma chamada.
MESSAGE	Usado como mecanismo *ad-hoc* para troca de informações.
SUBSCRIBE	Usado para que uma entidade SIP subscreva eventos.
NOTIFY	Usado para reportar eventos.
PUBLISH	Semelhante ao NOTIFY.
CANCEL	Usado para cancelar um pedido pendente.
ACK	Usado para atestar que a mensagem 200 OK em resposta ao INVITE foi recebida, portanto a chamada está totalmente estabelecida.
INFO	Usado para levar informação de sinalização fora de banda entre entidades SIP.
REGISTER	Usado para registar um *User Agent* SIP a um Proxy.
REFER	Usado para pedir ao receptor SIP que envie um novo pedido SIP para outra entidade.
PRACK	Usado pelo chamador para atestar respostas intermediárias ao INVITE.
BYE	Usado para terminar a chamada.
TRANSACTION RESPONSES	Usado para enviar códigos ACK por mensagens recebidas.

A mensagem INVITE tem duas secções: os cabeçalhos e a descrição de sessão, codificados segundo o protocolo SDP. SIP utiliza vários tipos de cabeçalho nas suas mensagens. Na Tabela 2.5 está uma lista dos tipos de cabeçalho possíveis e uma breve descrição dos mesmos.

Tabela 2.5 - Descrição dos cabeçalhos SIP [1]

Cabeçalho SIP	Propósito
Via	Lista o identificador SIP de todos os dispositivos SIP por onde passou a mensagem.
To	Dá o endereço URI de destino para a mensagem.
From	Dá o endereço URI de origem da mensagem.
Contact	Dá o endereço da última fonte da mensagem.
Call-Id	Identificador de chamada.
CSeq	Número de sequência na transacção.
Content-Type	Define o tipo de informação contido no corpo da mensagem (e.g., SDP)
Content-Length	Dá o tamanho da informação no corpo da mensagem.
Max-Forwards	Dá o número máximo de saltos de encaminhamento SIP que a mensagem pode visitar.

As respostas SIP podem ser variadas, como se pode verificar na Tabela 2.6. O primeiro dígito define a classe da resposta. Os últimos dois não têm nenhuma categorização específica. Deste modo, qualquer resposta com código entre 100 e 199 é referida como “1xx” e assim sucessivamente. Podemos ter seis tipos de resposta que estão listados na Tabela 2.6, assim como o seu significado.

Tabela 2.6 - Respostas SIP e o seu significado

Código	Significado
1xx	Provisional (Provisória): O pedido foi recebido mas continua a ser processado.
2xx	Success (Sucesso): A acção foi recebida com sucesso, compreendida e aceite;
3xx	Redirection (Redireccionar): Para que o pedido seja completado, é necessário tomar outras acções
4xx	Client Error (Erro no cliente): Existe erro na sintaxe e o servidor não pode completar o pedido.
5xx	Server Error (Erro no servidor): O pedido não aparenta ter erros, mas o servidor falhou a completá-lo.
6xx	Global Failure (Falha geral): O pedido não pode ser satisfeito por nenhum servidor.

Na Figura 2.10 está um exemplo do estabelecimento de uma chamada usando SIP. Em primeiro lugar o utilizador comunica com o primeiro servidor, tentando contactar o utilizador *johnsmith@test.com*. Como este não o conhece, envia para um segundo servidor, que por sua vez consulta um servidor de DNS (*Domain Name Server*) para descobrir onde encontra o dito utilizador. Este responde com a sua localização, o segundo servidor actualiza o campo TO no convite e reenvia-o para o utilizador em questão. Este atende a chamada, respondendo com 200 OK, sendo que esta mensagem vai do utilizador para o segundo servidor, este envia para o primeiro e só aqui a mensagem vai chegar à origem. Após ele receber o 200 OK, envia o sinal ACK directamente para o outro utilizador e a chamada é estabelecida, através da criação de um canal RTP directamente entre ambos.

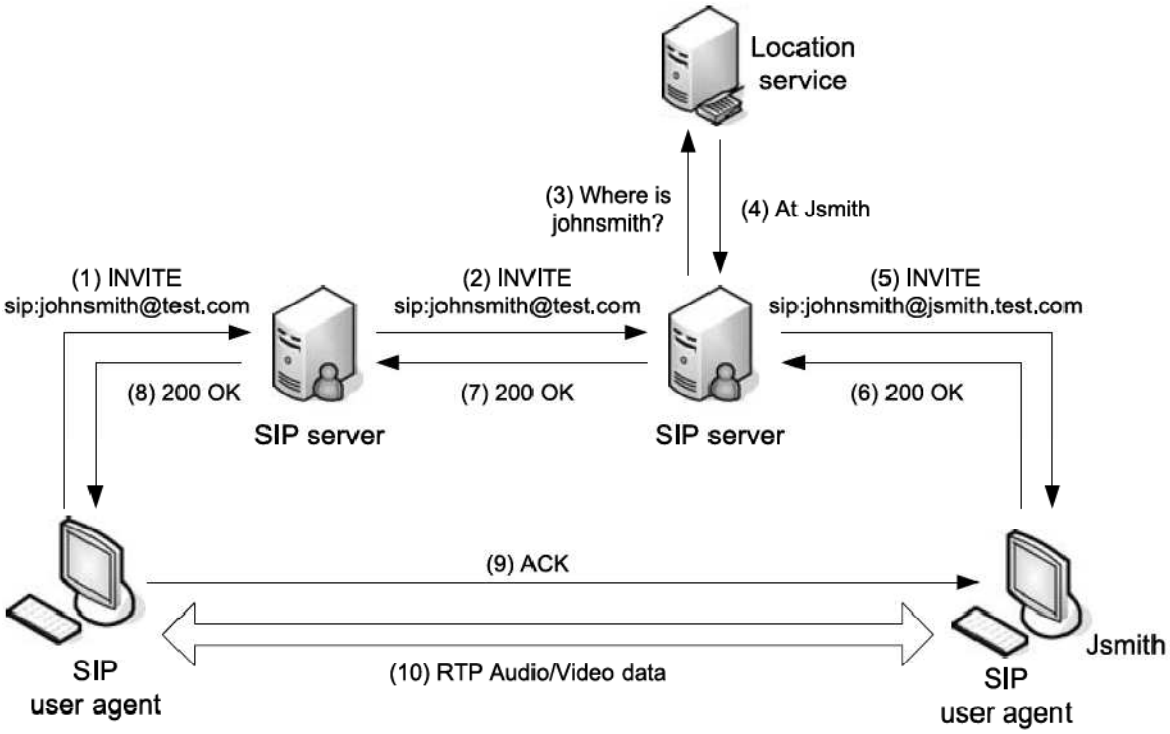


Figura 2.10 - Exemplo do estabelecimento de uma chamada usando SIP [3]

SIP é apenas um protocolo de sinalização, portanto apenas foi responsável pelo início e pelo estabelecimento da sessão. A comunicação entre os utilizadores é directa.

3 Implementação de um serviço

Este capítulo pretende fazer uma pequena resenha do que é a implementação de um serviço de voz sobre IP, quer a nível de soluções que já existem comercialmente, quer a nível de uma implementação de um servidor de código aberto e de um cliente a partir de bibliotecas abertas.

3.1 Soluções existentes

Sendo esta tecnologia comprovadamente mais económica que a telefonia clássica, existem bastantes empresas a comercializar este tipo de serviços, quer a nível de empresas, quer a nível residencial. A seguir serão falados alguns exemplos.

Para a maior parte das pessoas, ouvir falar em VoIP é sinónimo de falar no *Skype*. Trata-se da aplicação de voz sobre IP mais popular do mercado, apesar de não ser *software* aberto. Baseia-se num modelo *peer-to-peer*, ao contrário de uma arquitectura cliente/servidor como os servidores SIP. *Peer-to-peer* significa que cada utilizador na rede pode ser tanto servidor como cliente, descentralizando as funções da rede. O protocolo usado não é SIP, portanto não é livre para qualquer pessoa usar, o que torna esta solução pouco flexível para implementar um serviço proprietário. A utilização do *software* é livre, e as chamadas entre clientes não apresentam custos, mas para fazer telefonemas para a PSTN é necessário subscrever o serviço pago "*SkypeOut*". O mesmo acontece para receber chamadas da linha telefónica pública, que implica o pagamento do serviço "*SkypeIn*". De qualquer forma, este serviço é bastante popular, principalmente pela sua robustez e pela forma como ultrapassou algumas das fraquezas de clientes SIP, nomeadamente a sua dificuldade na utilização do serviço em redes NAT (*Network Address Translation*) ou atrás de uma firewall. Esta problemática será discutida adiante.

A rede *Skype* é, como já foi referido, uma rede *peer-to-peer*. Há dois tipos de nós nesta rede distribuída: os *hosts* normais e os super nós. Os primeiros são aplicações *Skype* usadas para fazer chamadas de voz e enviar mensagens de texto. Os segundos são os terminais dos utilizadores normais na rede *Skype*. Um Super Nó pode ser qualquer nó com um endereço IP público e com boa capacidade de processamento, memória e largura de banda [24]. Na Figura 3.1 está ilustrada a arquitectura desta rede como foi descrita. Cada nó tem que se ligar ao servidor de "*login*" para se autenticar. Este servidor é a única entidade central na rede *Skype*. Toda a informação é guardada e propagada de uma forma descentralizada.

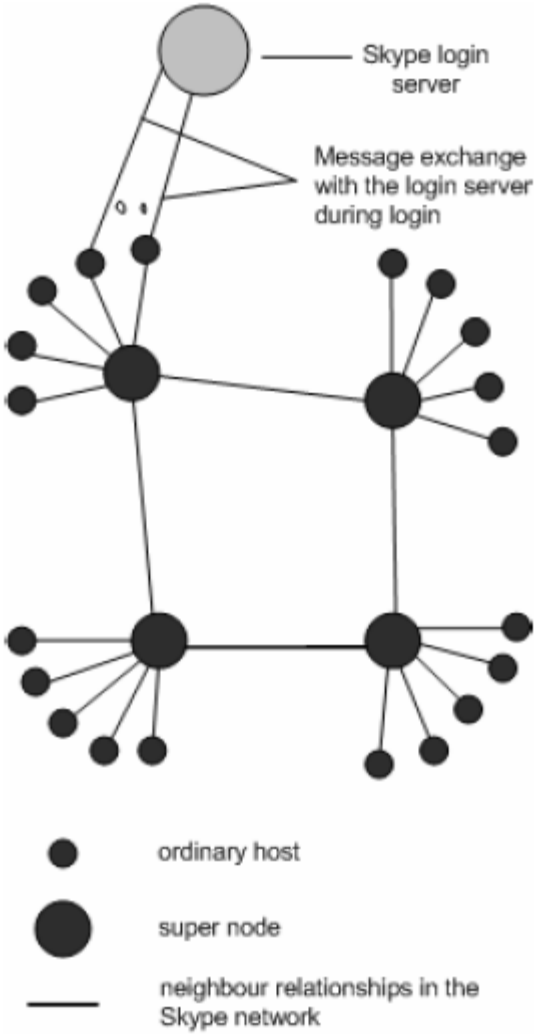


Figura 3.1 - Arquitectura da rede Skype [24]

Outro serviço bastante popular é o *VoipBuster*. Apresenta tarifas bastante competitivas e, inclusivamente, bastantes destinos são gratuitos, pelo menos durante algum período. A empresa responsável por ele chama-se *Betamax* e apresenta mais serviços idênticos mas sob outros nomes. Em [25] está uma comparação de taxas para os diferentes destinos a partir dos diferentes serviços. Na **Error! Reference source not found.** está uma lista destes mesmos serviços.

Tabela 3.1 - Serviços da empresa Betamax

12voip	budgetsip	calleasy	dialnow	Sipdiscount	internetcalls
intervoip	justvoip	poivy	Voiphit	netappel	webcalldirect
smsdiscount	smslisto	sparvoip	text4calls	nonoh	voipraider
voipstunt	voipcheap com	voipbusterpro	voipcheap uk	voipwise	voipdiscount
freecall	lowratevoip	voipbuster			

Relativamente ao *Skype*, estes provedores têm a vantagem de ser compatíveis com SIP. Não é obrigatório utilizar o cliente disponibilizado pelos diferentes servidores,

podendo utilizar um cliente SIP livre. Outra vantagem é o facto de ser possível ligar um PBX que permita essa função, como o *Asterisk*, e usar como gateway para a linha telefónica PSTN. Relativamente a tarifas de chamadas, todos são extremamente competitivos.

A lista de provedores deste tipo é bastante longa e os serviços são em tudo semelhantes. Em Portugal, por exemplo, existem algumas empresas dedicadas comercialmente a esta tecnologia. A lista é a seguinte (baseada em [26]):

Tabela 3.2 - Lista de Provedores VoIP em Portugal

Nome	Sede
125 Telecom	Portimão
3gntw - Tecnologias de Informação, Lda	Vila Nova de Famalicão
BitCanal	Vila Nova de Gaia
GSMBiT T.I.	Amadora
Lusosis - Soluções Informáticas, Lda.	Carnaxide
Netcall	Lisboa
Priory Computers	Silves
SipTel	Lisboa
StarTel	Torres Vedras
TelixSolutions	Viana do Castelo
Tntvoip Telecom IP	Vila Nova de Gaia
Tripleplay	Barreiro
VIPvoz, Serviços de telecomunicações Digitais, Lda.	Guimarães
VoipAnywhere	Lisboa
WebPhone	Madeira
Worldforce, Telecomunicações Lda	Faro

3.2 Implementação de um servidor

Existem várias soluções de servidores SIP disponíveis, quer comerciais, quer de código aberto (*Open Source*). Alguns exemplos de *software* de código aberto são:

Tabela 3.3 – Algumas soluções de código aberto [23]

Nome	Tipo de Licença	Observações
Asterisk	GPL	Não se trata de um servidor *proxy*, mas sim de um PBX. Ele pode actuar como B2BUA, ou seja, pode actuar como um UA em ambos os lados da chamada.
FreeSWITCH	MPL	Tal como o *Asterisk*, trata-se de um dipositivo que pode servir como PBX, telefone IP ou *softswitch* (é um dispositivo capaz de operar como os antigos *switches* das companhias telefónicas) [17]
Mysipswitch	BSD	É um projecto que visa encaminhar diferentes contas SIP de diferentes provedores, através de apenas um login SIP [18].
SIP Express Router (SER)	GNU	Servidor SIP de alta performance. Originou o *OpenSER*, que por sua vez deu origem ao *Kamailio* e ao *OpenSIPS*. São verdadeiros servidores SIP, capazes de manipular milhões de utilizadores e milhares de chamadas simultaneamente.
OpenSER	GPL	
Kamailio	GPL	
OpenSIPS	GPL	
sipX	LGPL	Também um PBX da empresa *SipFoundry*.

Existem inúmeras soluções comerciais. Apenas nomeando alguns exemplos de software de licença proprietária:

Tabela 3.4 - Algumas soluções comerciais [23]

Nome	Algumas características
Avaya Communication Manager	Servidor SIP [21].
Brekeke Software	Servidor SIP (*Proxy/Registrar*) e SIP PBX.
Cisco SIP Proxy Server	Servidor SIP. Descontinuado [19].
NEC SV7000 PBX	Solução de *software* e *hardware* bastante escalável [20].
Nortel SCS500	PBX Baseado em *software* [22].
TANDBERG Video Communication Server	Servidor SIP, e *gateway* H.323.
3Com VCX IP Telephony Module	SIP B2BUA e PBX
Microsoft Office Communications Server 2007	Sistema da *Microsoft* que gere todo o tipo de comunicações em tempo real.
RADVision SIP Server Platform	É um *framework* para o desenvolvimento de um servidor SIP (*proxy/registrar*), B2BUA e outras aplicações de servidor.
Siemens Hipath 8000	SIP *softswitch*, *mediaserver* (também H.323)

Destas extensas listas foram analisados dois servidores: *OpenSER* (gratuito) e *Brekeke* (comercial, mas gratuito para estudantes). Este último é compatível com o sistema operativo *Microsoft Windows XP*, portanto foi o primeiro a ser testado. A sua instalação é extremamente simples, contudo apresentou algumas razões que levaram a o descartar. Em primeiro lugar, como seria de prever num sistema proprietário, é impossível fazer qualquer alteração que seja no seu código fonte, o que poderia vir a ser útil para satisfazer alguma especificação. O mesmo se pode falar relativamente ao código para fazer o encaminhamento das mensagens SIP. Contudo, o *interface* é bastante intuitivo, já que é feito graficamente através de um cliente HTTP. De modo a suportar autenticação e autorização num servidor RADIUS (*Remote Authentication Dial In User Service* - RFC 2865), esta solução necessita da adição de 2 *plugins* em

java. No caso do Sistema Operativo *Windows*, este tipo de autenticação e autorização é raro, principalmente ao nível de soluções gratuitas. Foi testado o *Freeradius.net*, que se trata de uma adaptação do popular *FreeRADIUS*, mas compilado para ser compatível com o *Windows*. Esta solução mostrou não ser muito fiável, já que apresentava falhas frequentes no seu funcionamento. De destacar que apenas funcionava no modo *Debug*, isto é, numa consola *MS-DOS* indicando passo a passo as operações realizadas. No seu modo normal, não funcionava de todo.

Uma vez que a solução em *Windows* se apresentava com custos e pouca fiabilidade, a opção recaiu sobre uma solução em *Linux*, sistema operativo em excelência para soluções de código aberto. De seguida serão apresentados com pormenor os detalhes da configuração de um servidor SIP com autenticação, autorização, base de dados e contabilidade.

Como foi visto anteriormente, existem diversos servidores SIP em código aberto. O escolhido foi o *OpenSER*, já que é de código aberto, tem boa documentação, é totalmente configurável consoante as necessidades e extremamente robusto.

Ele resultou de um projecto inicial, o SER (SIP Express Router), um servidor SIP com licença GPL, desenvolvido na Alemanha [9]. Este é caracterizado pela rapidez com que encaminha os pedidos e consegue manipular milhares de utilizadores num único servidor. Com a paragem deste projecto, surgiu o *OpenSER* e uma versão comercial, sob a responsabilidade da *IPTel*. Na realidade não há muitas diferenças entre o SER e o *OpenSER* [9]. Uma das características que tornam este *software* tão atractivo é a possibilidade de adicionar módulos *Plug-in* facilmente. Desta maneira o programa vai ganhando novas funcionalidades, pois há diversos módulos que não existiam inicialmente, tal como *RADIUS*, *DIAMETER*, *ENUM*, entre outros, e frequentemente surgem novos.

Este projecto já não existe actualmente. Foi bifurcado em dois projectos independentes e com objectivos comerciais também diferentes: *OpenSIPs* e *Kamailio*. Contudo, apenas será feita referência ao *OpenSER*, já que a versão utilizada era mais antiga (versão 1.2.2), uma vez que era comprovado o seu sucesso e estabilidade na configuração e topologia pretendidas neste trabalho.

O desempenho e robustez do *OpenSER* permite-lhe ser usado para servir milhões de utilizadores, tal como se pode verificar no teste realizado em Março de 2007 [12]. O *OpenSER* 1.2.x foi capaz de manipular pedidos de um equivalente a 4 milhões de utilizadores e cerca de 28 milhões de chamadas por hora sem registar erros.

A topologia a implementar está na Figura 3.2.

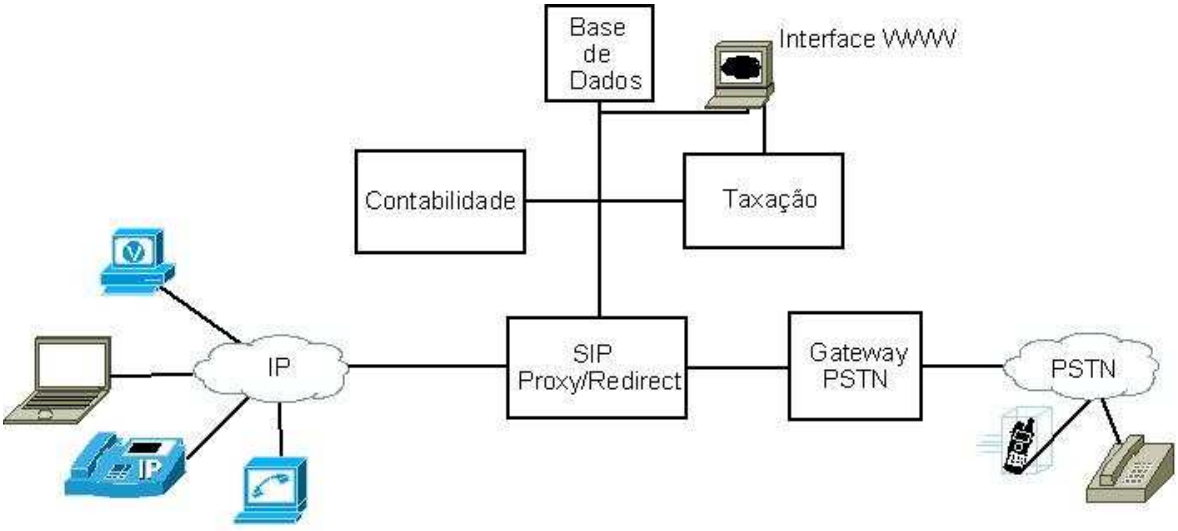


Figura 3.2 - Topologia servidor SIP

Como se pode verificar, os diferentes tipos de dispositivos que suportam SIP estão ligados numa rede IP, da qual fará parte o servidor SIP. Este estará ligado aos diferentes componentes de contabilidade, taxação, base de dados e *gateway* PSTN, que podem estar na mesma máquina ou noutras máquinas na rede IP. Por último, estão os dispositivos PSTN ligados na respectiva rede, sendo o *gateway* PSTN a ligação entre esta e a rede SIP.

3.2.1 Instalação do OpenSER:

Em primeiro lugar é necessário instalar um sistema operativo. A escolha recaiu no *Ubuntu Server 8.04*. Os passos para a instalação do sistema não serão abordados uma vez que não vão de encontro ao âmbito da dissertação.

O primeiro passo é fazer o *download* do *OpenSER*, instalar as respectivas dependências, extrair o ficheiro e instalar. Para o fazer, os comandos são os seguintes:

```
wget http://www.kamailio.net/pub/openser/1.2.2/src/openser-1.2.2-tls_src.tar.gz
sudo apt-get install gcc bison flex make openssl libmysqlclient-dev libradiusclient-ng2 libradiusclient-ng-dev mysql-server
tar –xzvf openser-1.2.2-tls_src.tar.gz
```

O *OpenSER* apresenta uma arquitectura por módulos, que o torna bastante versátil. Na sua instalação, através da edição do ficheiro *Makefile*, é possível escolher quais os módulos que serão instalados e quais os descartados. No caso, foi necessário apagar todos os módulos relativos à autenticação e contabilidade usando RADIUS, bem como para a utilização de bases de dados *MySQL*, da secção "*exclude_modules*=", de forma a estes serem incluídos. De seguida basta executar os seguintes comandos para que o *software* fique instalado.

```
sudo make all
sudo make install
```

A partir deste momento o programa já está operacional e pode correr usando a configuração pré-definida. Para a alterar, é necessário editar o ficheiro "*openser.cfg*" que está no directório "*/usr/local/etc/openser/*". Este é o ficheiro mais importante para o funcionamento do programa, uma vez que é carregado quando a aplicação é iniciada, indicando quais os módulos são necessários. Isto é uma vantagem, pois desta forma o programa evita a utilização de recursos excessivos com módulos que não são necessários para a sua configuração actual. É ele também que, cada vez que uma mensagem SIP é recebida, define qual o seu destino, garantindo que a norma RFC 3261 é cumprida. Se não se compreender bem esta norma (e), vão surgir problemas no ficheiro de configuração e dificilmente o servidor vai funcionar correctamente.

De seguida, coloca-se o OpenSER no arranque da máquina através do ficheiro "*/etc/init.d/openser*" que se pode encontrar na pasta de instalação do programa, bastando copiá-lo e alterar as permissões para que possa correr cada vez que o sistema arranca.

3.2.2 Instalação do suporte *MySQL*

Ao instalar as dependências do *OpenSER* já foi incluído o *mysql-server*. Existe um *script shell* para criar as tabelas *MySQL*, o que vai facilitar bastante o trabalho, mas vai configurar com parâmetros pré-definidos (nome da base de dados, nomes de utilizador, palavra-chave, etc). O comando para criar uma tabela é o seguinte:

```
./openser_mysql.sh create
```

Existe outro *script* para gerir o *OpenSER* através da linha de comandos. Trata-se do *Openserctl Shell script*. Ele pode ser usado para diversas operações, das quais se destacam: iniciar, parar e reiniciar o programa; adicionar e remover utilizadores da base de dados, bem como domínios; monitorizar o funcionamento do *OpenSER*. Existem mais funcionalidades mas estas são as mais importantes para este contexto. Este script vai buscar os recursos mais importantes a um ficheiro: o *"/usr/local/etc/openser/openserctlrc*". É necessário editá-lo para configurar correctamente com os dados da base de dados. Agora pode ser usado o comando para adicionar utilizadores à base de dados criada anteriormente. Mais adiante será possível fazê-lo através de uma interface HTTP, mas de qualquer forma o comando para o fazer está de seguida, onde os campos a editar estão a maiúsculas:

```
openserctl add UTILIZADOR PALAVRA-CHAVE ENDEREÇO_EMAIL
```

Para que haja autenticação através da base de dados, é necessário alterar o openser.cfg, como se verá adiante.

As tabelas criadas podem ser consultadas em anexo.

3.2.3 Instalação de uma *interface* HTTP para gerir a base de dados

Apesar de existirem os scripts vistos anteriormente para gerir a base de dados, esta solução não é muito elegante e exige alguns conhecimentos por parte do administrador. Visto isto, foram criadas algumas soluções para facilitar a gestão, podendo ser feita através de um browser de internet comum.

Existem 2 soluções mais populares: *SerMyAdmin* e *OpenserAdmin*. O segundo exige instalação da *Framework* "*Ruby on Rails*" e "*Ruby Gems*", não suporta utilizadores em domínios diferentes e o projecto aparenta estar parado, portanto a escolha recaiu sobre o *SerMyAdmin*. O primeiro projecto deste tipo era chamado *SERWeb*, contudo deixou de ser compatível com o *OpenSER* a partir de versões mais recentes. O *SerMyAdmin* está sob licença GPLv2, portanto pode ser usado sem qualquer custo.

A instalação propriamente dita desta ferramenta apresenta algumas dificuldades. As instruções têm que ser seguidas minuciosamente, a partir de [13], caso contrário a aplicação não funcionará. O primeiro passo consiste na instalação do *Framework Java* da empresa *Sun*. As instruções para tal encontram-se em anexo.

Agora é necessário definir onde o *SerMyAdmin* vai buscar a informação para aceder à base de dados. Estes dados podem ser encontrados no ficheiro "*/usr/local/tomcat6/conf/context.xml*". É Nesse ficheiro que vão ser guardadas informações como o endereço da base de dados, o nome de utilizador para aceder e respectiva palavra-chave. O conteúdo do ficheiro está no anexo. O passo seguinte consiste em aceder à linha de comandos *MySQL* e criar o utilizador especificado no ficheiro anterior. Os comandos são os seguintes:

```
mysql –u root -p
mysql> use openser;
mysql> grant all privileges on openser.* to administrador@'localhost' identified by 'user';
mysql> grant all privileges on openser.* to administrador@'%' identified by 'user';
```

Em jeito de observação, os dois últimos comandos são bastante semelhantes. Em princípio deveria ser necessário fazer apenas o segundo, mas a verdade é que um utilizador que se tentasse ligar a partir da mesma máquina não era identificado. Forçando essa identificação através do penúltimo comando, já não há esse problema.

O passo seguinte é fazer o *download* do *SerMyAdmin* de [13] neste caso a versão 0.8, extrair o ficheiro com a extensão "*war*" e colocá-lo do directório "*/usr/local/etc/tomcat6/webapps*/", abreviando o nome para "*SerMyAdmin.war*". Agora,

reiniciando o *tomcat*, a aplicação vai ser automaticamente instalada e, ao aceder ao endereço "*http://localhost:8080/SerMyAdmin*", deve aparecer a página inicial do *SerMyAdmin* mas não se deve efectuar o *login* ainda, uma vez que este ainda não foi criado. De qualquer forma, a partir do momento que a aplicação foi iniciada, ela vai proceder às alterações necessárias na base de dados que já existia. Agora é necessário adicionar um utilizador para aceder ao programa, bem como proceder a umas alterações na base de dados. Para facilitar esta tarefa, os criadores do *SerMyAdmin* criaram um ficheiro de texto, bastando só inserir na linha de comandos *MySQL*. Este ficheiro pode ser encontrado em anexo e os comandos seguintes mostram como fazer:

```
wget http://www.sermyadmin.org/openser/openser.sql
mysql -u root -p openser <openser.sql
```

A instalação está praticamente terminada. Existe uma aplicação útil, para enviar um e-mail para os novos utilizadores. Chama-se MTA (*Message Transfer Agent*). O modo de instalação desta aplicação encontra-se em anexo.

Para finalizar, basta usar o browser e entrar no endereço "*http://localhost:8080/SerMyAdmin*" e entrar com o nome de utilizador "*admin@setup*" e palavra-chave "secret". Estes dados foram definidos no ficheiro "openser.sql". Após a autenticação com sucesso convém alterar. Na Figura 3.3 está patente o aspecto da página inicial. Como se pode ver, é possível efectuar duas operações: Entrar (*Login*) e registar (*Register*).

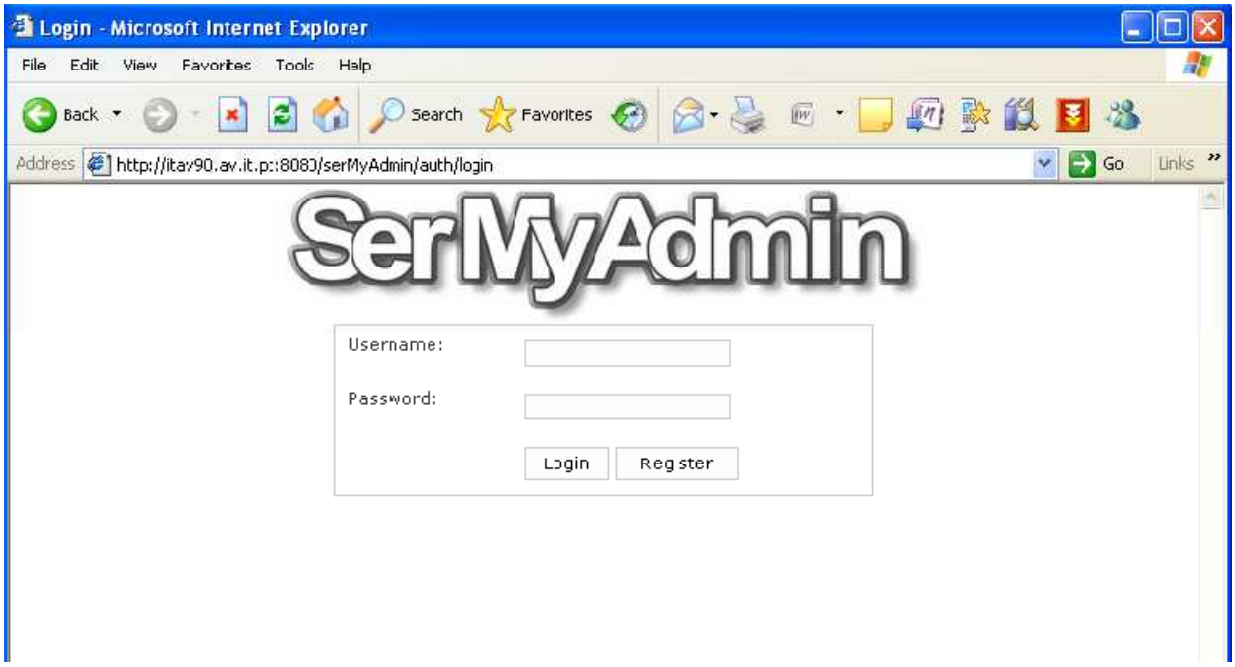


Figura 3.3 - Página inicial SerMyAdmin

O primeiro remete-nos para a página ilustrada na Figura 3.4 enquanto o segundo abre uma página onde se preenchem os dados para novos utilizadores, que posteriormente ficarão pendentes de autorização por parte do administrador.

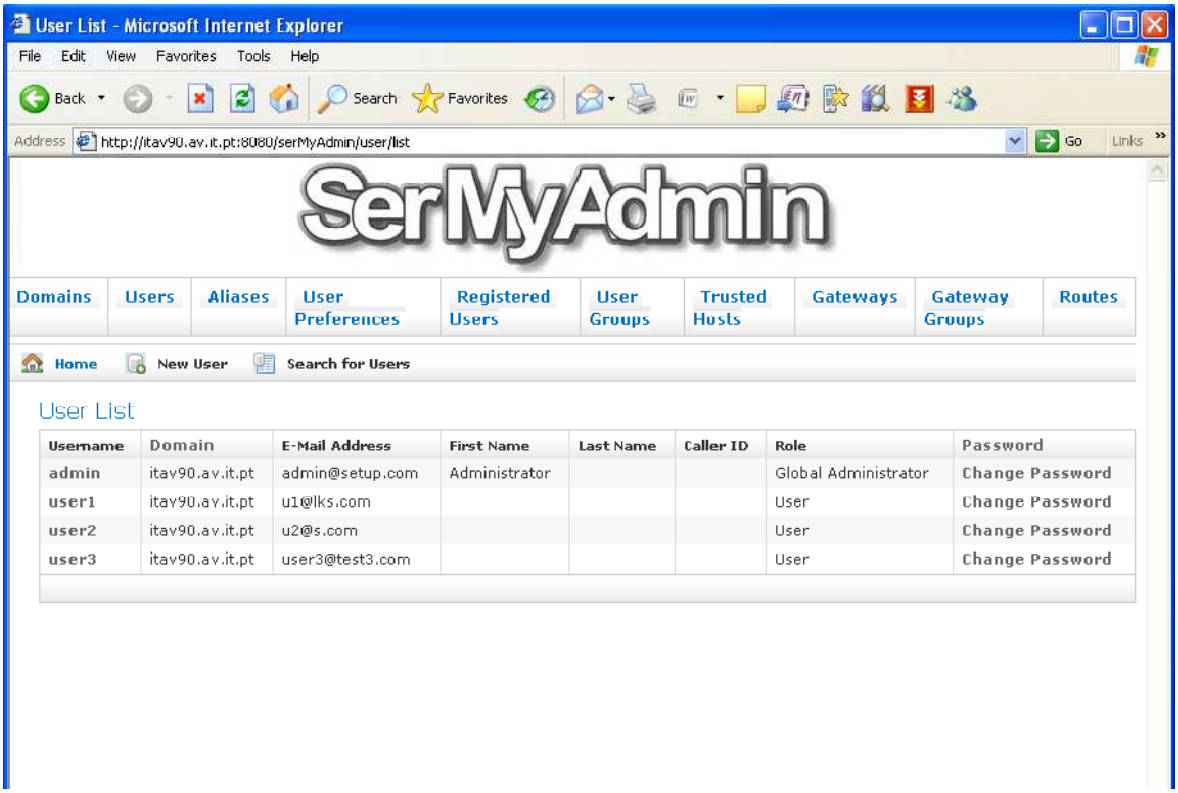


Figura 3.4 - Página principal depois de *login*

Este é o aspecto da página após a autenticação do administrador. É possível adicionar ou remover domínios e utilizadores (quais os autorizados), gerir os utilizadores registados (ver o seu estado e verificar os pedidos pendentes) e verificar os grupos e definir quais os utilizadores de cada grupo. As últimas 3 secções não serão muito exploradas, mas a sua função é de gerir o caminho para o *gateway* de saída. Na eventualidade de existir mais que um, pode ser útil na medida em que para um determinado tipo de chamadas pode ser melhor levar um determinado caminho, quer economicamente quer a nível de atraso.

3.2.4 Ligação à linha telefónica pública

Para enviar e receber chamadas para a linha telefónica pública (PSTN), é necessário um dispositivo chamado SIP PSTN *Gateway*. Existem diversos fabricantes com equipamentos do género no mercado, tal como a *Cisco*, *AudioCodes*, *Nortel*, *Quintum*, entre outros [9]. Existe uma alternativa economicamente mais vantajosa que é usar uma solução de *software* e correr num computador. O *software* em questão é um PBX virtual. O mais popular é o *Asterisk*, da empresa *Digium* e está disponível

com uma licença GPL, logo é livre de custos. Ele usa o conceito de canais e suporta mais protocolos além de SIP.

A instalação do *Asterisk* foi baseada em [16]. O seguinte comando serve para instalar os pacotes necessários para a instalação do *Asterisk*

```
sudo apt-get install bison openssl libssl-dev libasound2-dev libc6-dev libnewt-dev libncurses5-dev zlib1g-dev
```

Para fazer o download do *Asterisk* e dos, os comandos são os seguintes:

```
wget http://ftp.DIGIUM.com/pub/zaptel/zaptel-1.4.x.tar.gz
wget http://ftp.DIGIUM.com/pub/libpri/libpri-1.4.x.tar.gz
wget http://ftp.DIGIUM.com/pub/ASTERISK/ASTERISK-addons-1.4.x.tar.gz
wget http://ftp.DIGIUM.com/pub/ASTERISK/ASTERISK-1.4.x.tar.gz
```

A compilação do driver *Zaptel* e da biblioteca *libpri* está em anexo.

A instalação do *Asterisk* é da seguinte forma:

```
sudo tar xzvf ASTERISK-1.4.x.tar.gz
sudo tar xzvf ASTERISK-addons-1.4.x.tar.gz
sudo make clean
sudo ./configure
sudo make menuselect
sudo make
sudo make install
sudo make samples
sudo make config
```

Para a configuração SIP, que é o que interessa neste caso para que o *Asterisk* seja o *gateway* PSTN, edita-se o ficheiro "*SIP.conf*", no directório "/etc/asterisk/". Ele encontra-se em anexo com a configuração actual e será discutida mais à frente.

Uma alternativa mais dispendiosa desta configuração seria, como já foi referido, usando um equipamento específico para o fim. Um exemplo deste tipo de equipamentos é o *Router Cisco 2601* [9]. A configuração para um equipamento deste tipo está em anexo.

3.2.5 Contabilidade e tarifação

Um dos aspectos mais importantes num serviço de voz sobre IP, tal como na maioria dos serviços, é a sua capacidade para gerar receitas ou, pelo menos, não ter prejuízo. A partir do momento que os utilizadores se podem ligar à linha telefónica tradicional e efectuar chamadas, o uso dessas linhas vai ter custos, como tal, os utilizadores devem ser taxados mediante a sua utilização.

Em primeiro lugar é necessário um dispositivo que faça a contabilidade de utilização do sistema e a registe numa base de dados. O segundo passo é usar esses dados e aplicar uma tarifa. É neste contexto que entra o conceito de RADIUS (*Remote Authentication Dial In User Service*) que é um standard para AAA (*Autenthication,*

Authorization and Accounting – Autenticação, Autorização e Contabilidade). Este protocolo foi definido inicialmente em duas RFCs: RFC2865 para autenticação e RFC2866 para a contabilidade. Também se poderia usar uma tabela (com o nome ACC) na base de dados do *OpenSER*, contudo é mais simples trabalhar com um servidor RADIUS porque este usa um pacote para iniciar a contabilidade a cada transacção INVITE e outro para parar a cada transacção BYE, escrevendo um único registo durante a chamada. Usando *MySQL,* é necessário fazer manualmente a relação entre estas transacções [9].

A solução mais popular que não implica custos é o servidor *FreeRADIUS*.

3.2.5.1 Instalação FreeRadius

A instalação e configuração deste *software* são um pouco complicadas. Consistem em diversos passos, entre os quais a instalação dos pacotes e dependências, configuração da base de dados, configuração geral do programa, configuração do cliente Radiusclient-ng e posterior configuração do *OpenSER*. Os dois primeiros consistem nos seguintes comandos:

```
sudo apt-get install freeradius freeradius-mysql
mysql –u root –p create radius
cd /usr/share/doc/freeradius/examples
gunzip mysql.sql.gz
mysql –u root –p radius <mysql.sql
```

Estes comandos são responsáveis pela instalação do software, criação da base de dados e configuração das tabelas para que o *FreeRADIUS* as possa usar. Existe um problema conhecido com a contabilidade que pode ser resolvido aplicando um patch. Esta informação está em anexo, tal como as tabelas da base de dados criada.

3.2.5.2 Instalação CDRTool

Em primeiro lugar é necessário fazer o download. De seguida instalar o pacote. Os comandos são os seguintes:

```
wget http://download.dns-hosting.info/CDRTool/cdrtool_6.6.9_all.deb
sudo dpkg –i cdrtool_6.6.9_all.deb
sudo apt-get –f install
```

Depois é necessário aplicar um patch para fazer contabilidade que vem com o *CDRTool*.

```
cd /var/www/CDRTool/setup/radius/OpenSIPS
./radacct-patch.sh
```

A seguir edita-se o ficheiro *"/var/ww/CDRTool/setup/radius/sql.conf*" com os parâmetros da base de dados, para que o programa aceda à mesma. Por fim, copia-se este ficheiro para o directório "*/etc/freeradius*".

Para o programa correr é necessário configurar o ficheiro "*global.inc*" e colocá-lo no directório "/etc/cdrtool". O conteúdo deste ficheiro está no anexo, tal como as tabelas da base de dados criada.

3.2.5.3 Configuração FreeRADIUS e CDRTool

Agora que está tudo instalado é necessário configurar devidamente o *OpenSER*, que é o cliente, ao servidor, que neste caso é o *FreeRADIUS*. É usada a biblioteca *libradiusclient-ng* para ligar ao servidor *RADIUS*. Ela está alojada no directório "*/etc/radiusclient-ng*"

Os ficheiros de configuração deste programa estão em *"/etc/freeradius/*". Em primeiro lugar é necessário adicionar o *OpenSER* como cliente. O ficheiro em questão é o "clients.conf" e as linhas que o fazem são as seguintes:

```
Client 127.0.0.1 {
      secret=user
      shortname=localhost
      nastype=other
}
```

Para explicar o que é feito nas linhas anteriores, seria necessária uma vasta compreensão da semântica usada no *FreeRADIUS*. Portanto, não entrando em pormenores, é definido o cliente, neste caso a aceder na mesma máquina (endereço 127.0.0.1) e com segredo de acesso "*user*". Esta palavra é necessária adiante, caso contrário o cliente será recusado.

De seguida é editado, dentro da mesma pasta, o ficheiro "*radiusd.conf"* visando activar a contabilidade. O ficheiro é bastante vasto e lida com as configurações gerais do programa. O que é necessário fazer é retirar o carácter de comentário de algumas linhas. São elas:

```
Accounting {
      acct_unique
      detail
      sql
      Unix
      Radutmp
}
```

Este programa utiliza um ficheiro de dicionário que define atributos e métodos. O programa *CDRTool* traz um ficheiro deste tipo e, uma vez que os programas vão estar em sintonia, convém que usem o mesmo. Portanto é copiado o ficheiro de um directório para o outro e incluído no dicionário do *FreeRADIUS*.

```
cp /var/www/CDRTool/setup/radius/OpenSIPS/dictionary.ser /etc/freeradius/
sudo vim /etc/freeradius/dictionary
```

é adicionada a linha:

```
$INCLUDE    /etc/freeradius/dictionary.ser
```

Agora falta configurar o cliente RADIUS (*radiusclient-ng*). Copia-se o ficheiro de dicionário incluído no *OpenSER* e inclui-se o dicionário do *FreeRADIUS*.

```
cp /usr/local/etc/openser/dictionary.radius /etc/radiusclient-ng/
```

Edita-se este ficheiro é adicionada a seguinte linha no fim do ficheiro:

```
$INCLUDE    /etc/freeradius/dictionary.ser
```

Agora deve-se configurar os servidores do *radiusclient-ng*. Na mesma pasta existe um ficheiro chamado "*servers*". Neste ficheiro deve ser colocado o nome do servidor e a palavra-chave definida anteriormente. A linha deve ser como a seguinte:

```
127.0.0.1                user
```

Finalmente, o servidor deve ser colocado no ficheiro "*radiusclient.conf*". A linha é a seguinte:

```
Acctserver    127.0.0.1
```

3.2.6 Ficheiro de configuração do *OpenSER*: "*openser.cfg*"

Como já foi dito, este é o ficheiro mais importante para o funcionamento do servidor. Agora que todos os programas estão instalados, é necessário que o *OpenSER* saiba como e quando os aceder. Por exemplo, vai aceder ao *Asterisk* quando for uma chamada para a linha pública, ou vai aceder ao *FreeRADIUS* cada vez que uma chamada é estabelecida e necessita ser contabilizada. A seguir será feita uma pequena descrição de cada secção (resumida) do ficheiro na sua configuração actual.

3.2.6.1 Definições globais do programa

```
debug=7          # debug level (cmd line: -ddddddddddd)
fork=yes
log_stderror=yes    # (cmd line: -E)
children=4
port=5060
```

Estas linhas acima definem alguns dos parâmetros globais do programa. Neste caso apenas está especificado o funcionamento ou não em modo *debug*, a porta no qual o *OpenSER* vai estar a operar (5060) e a definição de processos utilizados (*fork e children*).

3.2.6.2 Módulos e parâmetros

```
mpath="/usr/local/lib/openser/modules/"
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
```

```
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "auth.so"
loadmodule "auth_db.so"
loadmodule "acc.so"
loadmodule "avpops.so"
## ----------------Parametros dos modulos
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("group","group_column","name")
modparam("acc","radius_config", "/etc/radiusclient-ng/radiusclient.conf")
modparam("acc", "radius_flag", 2)
modparam("acc", "radius_missed_flag", 3)
modparam("acc", "radius_extra", "User-Name=$Au; Calling-Station-Id=$from; Called-Station-Id=$to; Sip-Translated-Request-URI=$ruri; SIP-Rpid=$avp(s:rpid); Source-IP=$si; Source-Port=$sp; Canonical-URI=$avp(s:can_uri); Billing-Party=$avp(s:billing_party); Divert-Reason=$avp(s:divert_reason); X-RTP-Stat=$hdr(X-RTP-Stat); Contact=$hdr(contact); Event=$hdr(event); SIP-Proxy-IP=$avp(s:sip_proxy_ip); ENUM-TLD=$avp(s:enum_tld)")
```

Nestas linhas são definidos os módulos externos que vão ser carregados pelo *OpenSER* e definidos alguns parâmetros dos mesmos. O primeiro comando define o directório onde os módulos estão instalados, o comando "*loadmodule*" vai carregar esse mesmo módulo e o comando "*modparam*" configura o respectivo módulo. Uma explicação mais detalhada de cada módulo implica um conhecimento mais compreensivo e detalhado de cada módulo, que não é relevante. Destaque para os parâmetros de contabilidade ("*acc*"), que definem que irá ser no modo RADIUS, definem as *flags* que vão ser usadas no código de encaminhamento e a última linha de comando vai definir uma série de informações que vão ser usadas para fazer a taxação.

3.2.6.3 Módulo principal

Este bloco de texto parece-se bastante com qualquer linguagem de programação. Os blocos são delimitados por chavetas, as funções por parêntesis e é usado o ponto e vírgula para indicar o fim de linha. A primeira parte, que está no bloco em baixo, faz as verificações iniciais de rotina, com o objectivo de evitar *loops* ou mensagens demasiado longas.

```
Route{
```

```
if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
};
if (msg:len >=  2048 ) {
        sl_send_reply("513", "Message too big");
        exit;
};
```

Agora, as mensagens que passaram nestes testes, significa que são válidas e vão ser submetidas ao algoritmo que vai determinar o seu destino. Se o método for diferente de REGISTER, o *OpenSER* vai guardar o caminho. Isto vai fazer com que o servidor continue no caminho entre dois UACs. Assim as mensagens subsequentes vão tomar o caminho guardado.

```
if (!method=="REGISTER")
        record_route();
```

O bloco seguinte vai assegurar que não existe re-INVITE. Isto é importante porque nestes casos não vão ser pedidas as credenciais novamente. A função "*has_totag*()" assegura que o pedido é sequencial e nestes casos terá que ter um cabeçalho ROUTE. A existência deste é assegurada pela função "*loose_route*()". Caso não o tenha, o pedido será descartado e será respondido com o código "404". Se o método for "BYE", então necessita ser contabilizado, portanto vai tomar a *flag* "2", que está definida em cima, nos parâmetros. Se o método for INVITE, então é pedido ao UAC as credenciais para se autenticar no servidor. Caso as condições se verifiquem, o código chama a função "*route(1)*", que será vista adiante.

```
If (has_totag()) {
        if (loose_route()) {
                if(method=="BYE") {
                        #account
                        setflag(2);
                };
                if(method=="INVITE") {
                        if(!proxy_authorize("","subscriber")) {
                                proxy_challenge("","1");
                                exit;
                        } else if (!check_from()) {
                                sl_send_reply("403"," Forbidden, use From=ID");
                                exit;
                        };
                };
                route(1);
        } else {
                sl_send_reply("404","Not here");
        }
        exit;
}
```

De acordo com a norma RFC 3261, os pedidos CANCEL têm que ser processados na mesma maneira que os pedidos INVITE. A secção de código abaixo vai verificar se a mensagem CANCEL vai corresponder a uma transacção existente e vai ser devidamente processada. Por vezes existem retransmissões associadas a uma transacção existente, e nesse caso a função "t_check_trans()" vai tratar do assunto e vai sair do *script*.

```
    #Cancel processing
    if (is_method("CANCEL")) {
            if (t_check_trans()) t_relay();
            exit;
    }
    t_check_trans();
```

Se o método for REGISTER, significa que é um pedido para registar no *Proxy*, portanto será manipulado pela função "*route(2)*" que será vista adiante. Caso não o seja será a função "*route(3)*" que vai actuar.

```
    if (method=="REGISTER") {
            route(2);
    } else {
            route(3);
    };
}
```

3.2.6.4 Função para encaminhar chamadas: route(1)

Este bloco é utilizado para encaminhar as chamadas e retornar mensagens de erro caso existam.

```
route[1] {
    # send it out now; use stateful forwarding as it works reliably
    # even for UDP2TCP
    t_on_reply("1");
    t_on_failure("1");
    if (!t_relay()) {
            sl_reply_error();
    };
    exit;
}
```

3.2.6.5 Função que lida com pedidos de registo: route(2)

Este bloco vai lidar com os pedidos de registo. Primeiro verifica se o domínio é local, domínio este que tem que ser adicionado na base de dados; depois vai pedir autenticação; caso esta seja aceite, vai guardar a localização do UAC (função "*save("location")"*).

```
route[2] {
    if (is_uri_host_local()) {
            if (!www_authorize("","subscriber")) {
```

```
                www_challenge("", "1");
                exit;
        };
        if (!check_to()) {
                sl_send_reply("403","Forbidden");
                exit;
        };
        save("location");
        exit;
    } else if {
        sl_send_reply("403", "Forbidden");
    };
}
```

3.2.6.6 Função que lida com os outros tipos de pedido: route(3)

Este bloco vai lidar com os pedidos que não são de registo. Como estes podem ser de vários tipos, será separado por tipo. Se o método é INVITE, terá que ser contabilizado.

```
route[3] {
    if(method=="INVITE") {
        #acc
        setflag(2);
        setflag(3);
        $avp(s:sip_proxy_ip)="127.0.0.1";
    };
```

Agora será verificada a fonte. Caso seja domínio interno terá que ser pedida autenticação e verificado o campo FROM.

```
    if (is_from_local()) {
    #dominio interno
        if (!proxy_authorize("","subscriber")) {
                proxy_challenge("","1");
                exit;
        } else if (!check_from()) {
                sl_send_reply("403", "Forbidden, use From=ID");
                exit;
        };
```

Verificadas todas as condições, a chamada é encaminhada.

```
        if(avp_db_load("$ru/username","$avp(s:callfwd)")) {
                avp_pushto("$ru", "$avp(s:callfwd)");
                route(1);
                exit;
        }
        consume_credentials();
```

O destino é verificado e vai ser encaminhada mediante essa informação:

```
        lookup("aliases");
        if (is_uri_host_local()) {
```

```
                        #Dentro do domínio para dentro do domínio
                        route(10);
                } else {
                        #Dentro para fora
                        route(11);
                };
        } else {
                lookup("aliases");
                if (is_uri_host_local()) {
                        #Fora para dentro
                        route(12);
                } else {
                        #fora para fora
                        route(13);
                };
        };
}
```

3.2.6.6.1 Função que encaminha chamadas do domínio interno, para o mesmo domínio: route(10)

Os domínios internos são descobertos na própria tabela. Caso o endereço seja numérico, significa que se trata de um número telefónico, portanto terá que ser encaminhada a chamada para a PSTN. Antes de fazer este encaminhamento é verificado se o utilizador pertence ao grupo “pstn”, que foi um grupo criado para separar os utilizadores que podem fazer chamadas para a rede telefónica dos que só podem fazer chamadas entre telefones IP.

```
route[10] {
        append_hf("P-hint: inbound->inbound \r\n");
        if (uri=~"^sip:[0-9]*@") {
                if(is_user_in("credentials","pstn")) {
                        route(4);
                        exit;
                } else {
                        sl_send_reply("403","No permissions for pstn calls");
                        exit;
                };
        };
        if (!lookup("location")) {
                sl_send_reply("404", "Not Found");
                exit;
        };
        route(1);
}
```

3.2.6.6.2 Função que encaminha chamadas do domínio interno, para outro domínio: route(11)

Apenas verifica o endereço por DNS e encaminha a chamada.

```
route[11] {
        append_hf("P-hint: inbound->outbound \r\n");
        route(1); }
```

3.2.6.6.3 Função que encaminha chamadas de um domínio externo e para o interno: route(12)

Verifica a existência de *aliases*, isto é, nomes definidos na tabela para substituir o número. Caso seja encontrado, substitui o endereço URI.

```
route[12] {
    lookup("aliases");
    if (!lookup("location")) {
        sl_send_reply("404", "Not Found");
        exit;
    };
    route(1);
}
```

3.2.6.6.4 Função que encaminha chamadas de um domínio externo para outro externo: route(13)

Por uma questão de segurança, chamadas de fora do domínio para fora do domínio não vão ser permitidas.

```
route[13] {
    sl_send_reply("403","Forbidden");
    exit;
}
```

3.2.6.7 Função para encaminhar chamadas para a PSTN: route(4)

Este bloco está responsável por chamadas para a linha telefónica clássica. A função "*rewritehostport()*" vai fazer com que a chamada seja encaminhada para, neste caso, o *Asterisk*, usando também a sinalização SIP. Como será visto adiante, o *Asterisk* está configurado para usar a porta 5061. Isto deve-se ao facto de correr na mesma máquina do *OpenSER* que já utiliza a porta 5060 que é a porta mais comum no protocolo SIP.

```
route[4] {
    rewritehostport("localhost:5061");
    route(1);
}
```

3.2.7 Configuração do *gateway* para a PSTN

Como já foi referido, o *gateway* escolhido para este cenário foi o *Asterisk*. Para o ligar à rede telefónica, existem duas maneiras. A primeira é fazer uma ligação física a uma linha telefónica, usando para o efeito placas FXS (*Foreign eXchange Subscriber*) e FXO (*Foreign eXchange Office*). A segunda é aproveitando as suas capacidades de ser um UAC e ligar a um provedor.

FXS é a *interface* que fornece a linha analógica ao cliente, isto é, trata-se do dispositivo que fornece o tom de ligação, corrente de energia e som. Por sua vez, o

FXO é a interface que recebe a linha analógica, ou seja, é o dispositivo no telefone ou no sistema de telefonia analógica. Ele sinaliza se o telefone está ou não no gancho. Um *modem* pode ser um dispositivo deste tipo, desde que tenha um dos *chipsets* suportados pelo *Asterisk*, listados na Tabela 3.5. Os *modems* deste tipo são chamados X100P *clone*, uma vez que os primeiros deste tipo eram denominados X100P. A configuração de um dispositivo deste tipo encontra-se em anexo.

Tabela 3.5 - Lista de chipsets suportados pelo *Asterisk*

Chipset
Intel 537PG and 537PU
Ambient MD3200
Motorola 62802

A configuração escolhida, pelo baixo preço das chamadas e pela ausência de um equipamento compatível, foi a de usar um provedor para efectuar as chamadas para a linha telefónica pública. A configuração está em anexo.

3.2.8 Configuração da tarifação

A instalação da ferramenta utilizada na taxação já foi efectuada anteriormente, contudo ainda não está devidamente configurada. Mais uma vez, esta configuração exige que os passos sejam seguidos minuciosamente.

O programa traz um *script shell* para configurar a base de dados, contudo necessita que seja editado um ficheiro para que ele o faça devidamente. Trata-se do ficheiro "*/var/www/CDRTool/setup/mysql/create_users.mysql*" que necessita de ser editado e devem ser colocados os valores correctos da base de dados local. De seguida é usado o comando:

```
./setup_mysql.sh "" localhost
```

Este passo vai criar a base de dados do *CDRTool*, criando também a conta de utilizador inicial (utilizador "*admin*" e palavra-chave "*password*"). Agora vem a parte mais complexa e onde surgiram mais problemas, que é o ficheiro de configuração do programa. Trata-se do "*global.inc"* que está em anexo, mas não vai ser analisado em detalhe. É necessário, também, a configuração do servidor PHP e dependências, que se pode encontrar em anexo.

Na Figura 3.5 está patente a arquitectura do *CDRTool*. Ela usa duas bases de dados, mais especificamente, a tabela "*radacct*" e a própria tabela chamada "*CDRTool*". O processo principal é o motor de taxação (*rating engine*), que é responsável por tirar a duração da chamada e atribuir o respectivo preço, mediante algumas condições específicas.

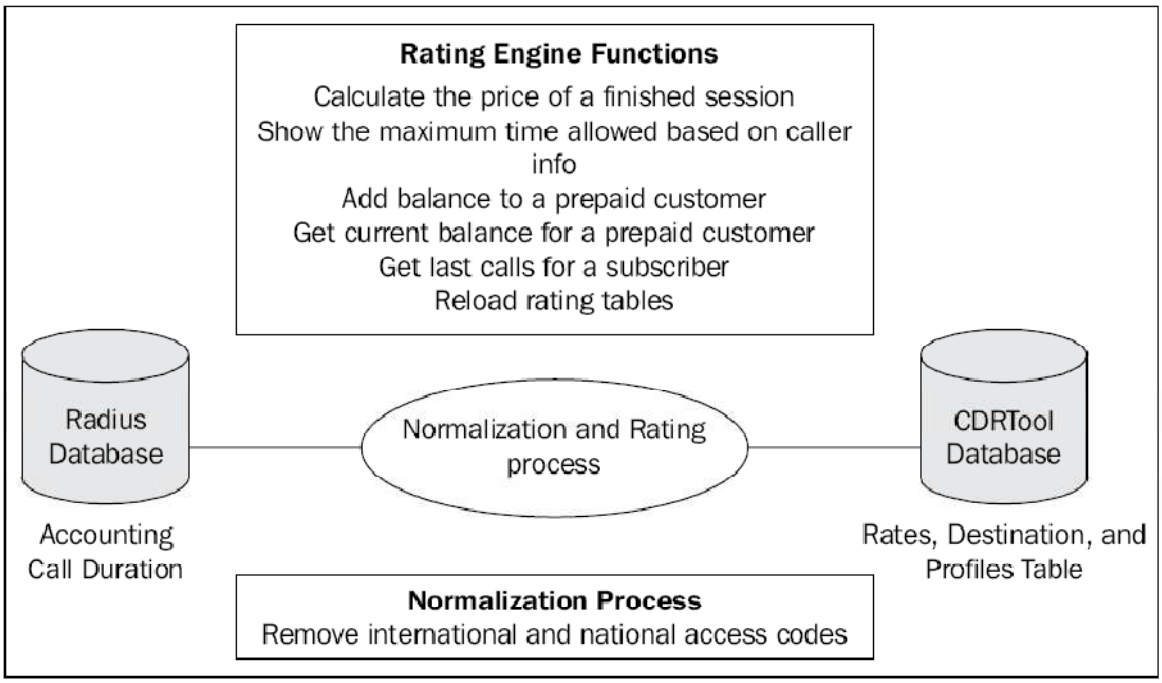


Figura 3.5 - Arquitectura do *CDRTool* [9]

O programa faz a taxação da chamada imediatamente, baseado num plano. Ele pode ir buscar a múltiplas fontes de dados, tal como o *Asterisk*, *Cisco* ou *Openser*. Ele usa a base de dados RADIUS que contém a duração, intervenientes e fontes de informação da chamada. O preço da sessão é calculado em tempo real e gravado na tabela. Esse cálculo pode depender de vários factores configuráveis, tais como a altura do dia, dia da semana ou feriados. Podem ser criados diversos perfis para utilizadores tipo diferentes.

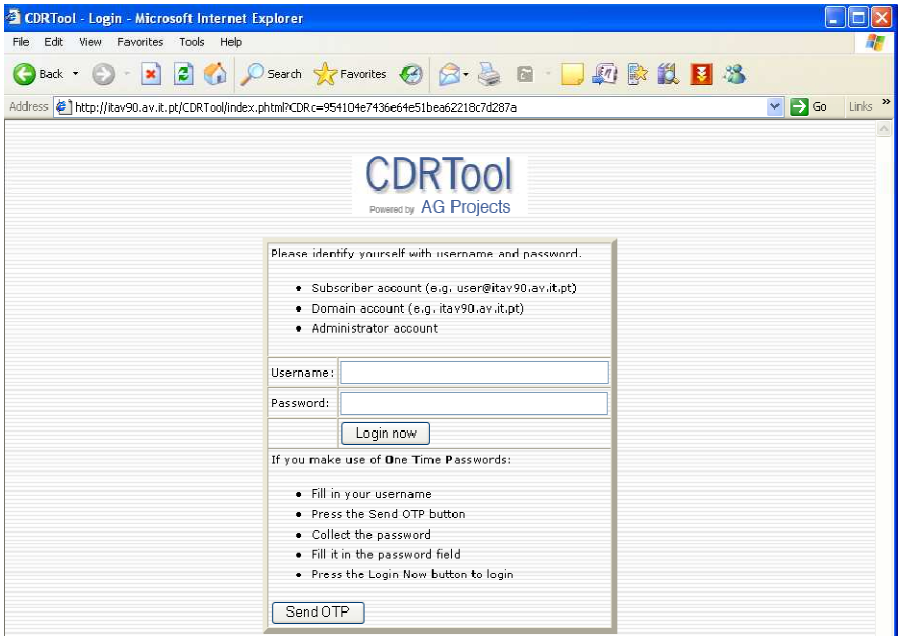


Figura 3.6 - *CDRTool*: Página inicial

Na Figura 3.6 está patente o aspecto da página inicial da ferramenta *CDRTool*. Na primeira utilização os dados para autenticação são os definidos no ficheiro de configuração como foi visto na secção 3.2.5.2. logo após a primeira visita deve-se alterar os dados para personalizar o sistema. Após isto, o sistema deve ser configurado. Devem ser criados os planos de taxação (*Rating Plans*), como está patente na Figura 3.7.

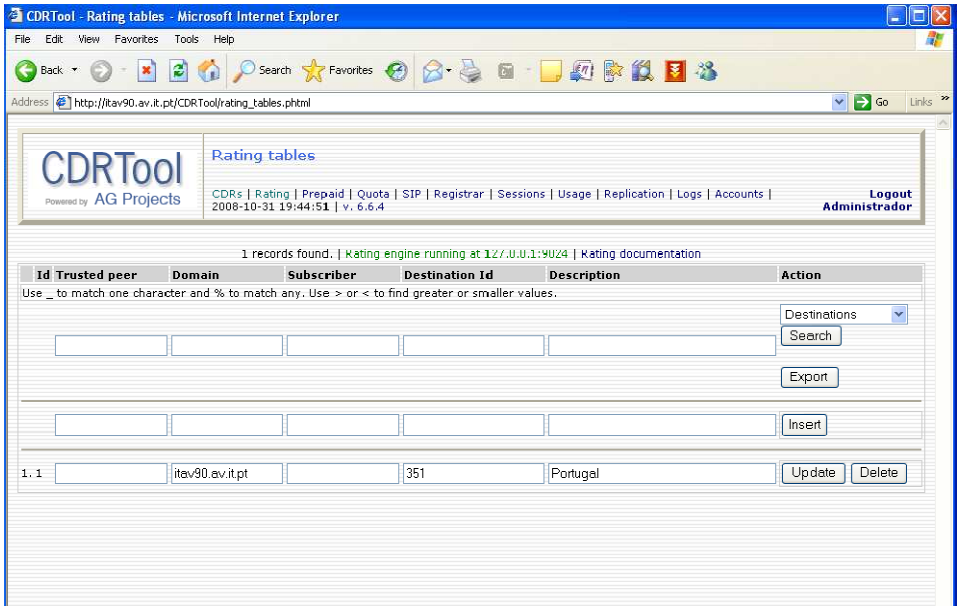


Figura 3.7 - *CDRTool*: Tabelas de taxas

Há diversos perfis que podem ser configurados, tais como diferentes taxas consoante o dia da semana, a hora do dia, diferentes fusos horários, entre outros. O programa determina o plano a aplicar baseando-se na entidade de contabilidade. No caso é o RADIUS. O destino das chamadas é identificado, por ordem preferencial, pelo endereço canónico, que se trata do endereço de destino após todas as verificações do endereço pelo *proxy* SIP, pelo URI de pedido tal como o UA o apresenta ou pelo conteúdo do campo “TO” no cabeçalho.

A duração das chamadas é determinada pela recepção da mensagem “BYE”. Caso esta seja perdida convém haver uma medida alternativa para sinalizar o fim da chamada, caso contrário poderá haver erro no cálculo. A utilização de um *gateway* para os dados *Media* pode resolver o problema ou através da verificação do estado da chamada no código de encaminhamento do *OpenSER*.

O procedimento para configuração desta aplicação encontra-se em anexo.

3.2.9 Arquitectura do servidor

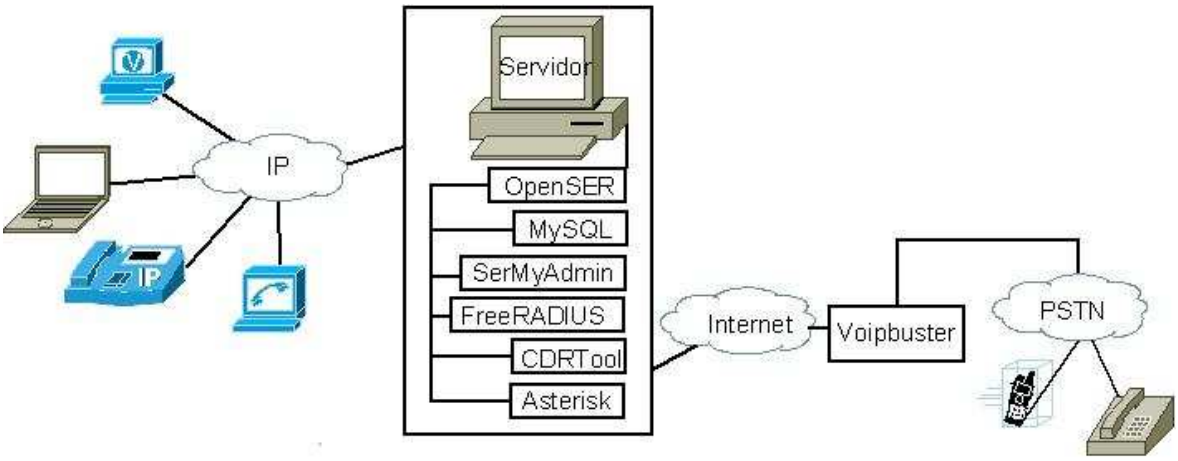


Figura 3.8 - Arquitectura do servidor

Após a instalação de todos os programas vistos anteriormente, o sistema tem o funcionamento como está ilustrado na Figura 3.8. Esta figura é semelhante à Figura 3.2, contudo demonstra que todos os diferentes componentes do servidor estão em funcionamento na mesma máquina. Demonstra, também, que a integração com a linha PSTN foi efectuada usando o serviço *Voipbuster*.

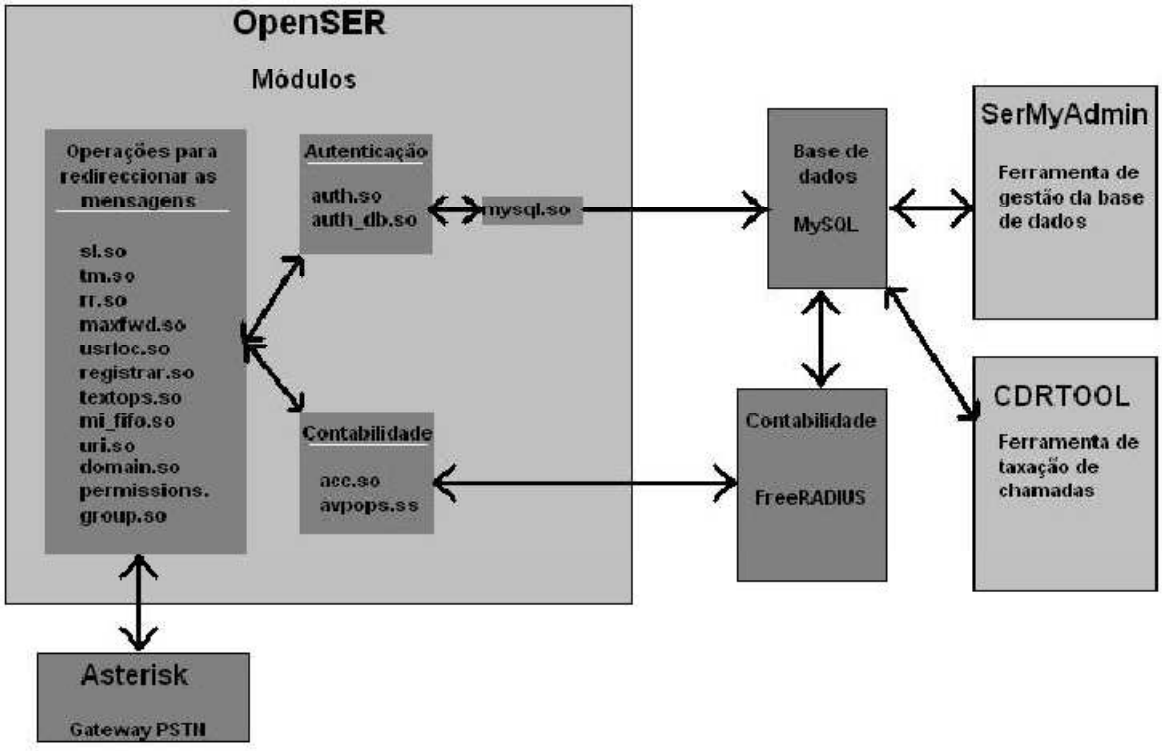


Figura 3.9 - Arquitectura do servidor com os módulos do *OpenSER*

O *OpenSER* por si só já se trata de uma aplicação constituída por diversos módulos, como está patente na Figura 3.9. São necessários diversos módulos para tratar do algoritmo de encaminhamento das mensagens. A autenticação necessita de dois

módulos adicionais, que vão usar funções do módulo "*mysql*" para manipular a base de dados. A contabilidade também necessita de dois módulos adicionais, que vão permitir o *FreeRADIUS* contabilizar as chamadas e, por sua vez, guardar os dados na base de dados. A ferramenta *SerMyAdmin* vai ser importante para manipular mais fácil e intuitivamente a base de dados. A ferramenta *CDRTool* trata da taxação das chamadas mediante a informação na base de dados guardada pelo *FreeRadius*. Por último, o *Asterisk* trata de servir como *gateway* para a linha PSTN.

3.3 Implementação de um cliente

O primeiro passo quando se pretende implementar *software* será uma pesquisa do que já existe, e dentro deste grupo, qual se pode utilizar. Baseado numa lista elaborada em [10] e [11], a Tabela 3.6 lista as bibliotecas e pilhas de protocolos existentes.

Tabela 3.6 - Lista de bibliotecas SIP

Nome	Linguagem	Algumas Características
Aloha	Java	Para a Spring Framework. Não existe muita informação acerca desta Sip *Stack*.
YASS	C++	Usada em Yate (*Yet Another Telephone Engine*); Trabalha em *Windows* e *Linux;* Pequena, mas com muitas funções; Licença GPL (*General Public License*), portanto é de uso público livre.
MjSip	Java	Licença GPL; Biblioteca SIP completa e poderosa para plataformas Java.
oSIP	C	Biblioteca SIP da empresa AntiSip.
eXosip	C	Extensão da biblioteca oSIP; Licença comercial ou Livre (GPL);
Vovida SIP	C++	Não está a ter actualmente desenvolvimento.
sipXtackLib	C++	Faz parte do software de código aberto da empresa Pingtel.
Reciprocate	C++	Desenvolvida pela comunidade SIPfoundry.
NIST SIP	Java	Várias aplicações e ferramentas em Java.
PJSIP:	C/Python	É leve, tem alto desempenho e é muito portável. Apresenta também uma colecção de codecs e toda a implementação de funções de transporte.
Twisted	Python	Pilha de protocolos e aplicações escritas em Python.
libmsip	C++	Foi criada para o cliente minisip.
phAPI	C	Baseado em eXoSIP e faz parte do projecto openwengo.
Sofia-SIP	C	Desenvolvido no *Nokia Research Center.* Licença LGPL (Lesser GPL).
OpenSipStack	C++	Licença MPL (Mozilla Public License). Várias ferramentas implementadas.
Verona -	C/Python	Licença GPL, baseada em oSIP. Trabalha em Linux, Windows e Mac-OS.
PhClickDial -		É um plugin para o Internet Explorer, baseado no Verona, permitindo a funcionalidade ClickToDial.
Microsoft RTC (Real-Time Communications)	C++	Pilha de protocolos SIP e RTP de alto nível, incluída no Windows XP e nas várias versões do Windows Messenger. As novas funcionalidades incluídas na versão 1.2 têm comportamentos estranhos quando usados com outros clientes SIP.

3.3.1 PJSIP e *SIPekSDK*

A implementação do protocolo SIP escolhida é denominada PJSIP, que se trata de uma plataforma *open source* completamente compatível com a norma SIP (RFC 3261).

As vantagens principais prendem-se com o facto de ser *open source*, isto é, não apresenta custos, quer de uso, quer de alteração do código fonte; além disto, é bastante escalável, desde pequenos dispositivos SIP até servidores multi-processador; outra vantagem é a sua portabilidade para diversas arquitecturas (*32bit, 64bit, big/little endian*, qualquer sistema operativo); Por último, ela apresenta uma

implementação por camadas, que não só incorpora a definição SIP, como também uns módulos *Media* e transporte, apresentando um conjunto de *codecs* áudio para compressão e transmissão da voz.

Na Figura 3.10 está patente essa mesma arquitectura por camadas da pilha de protocolos PJSIP.

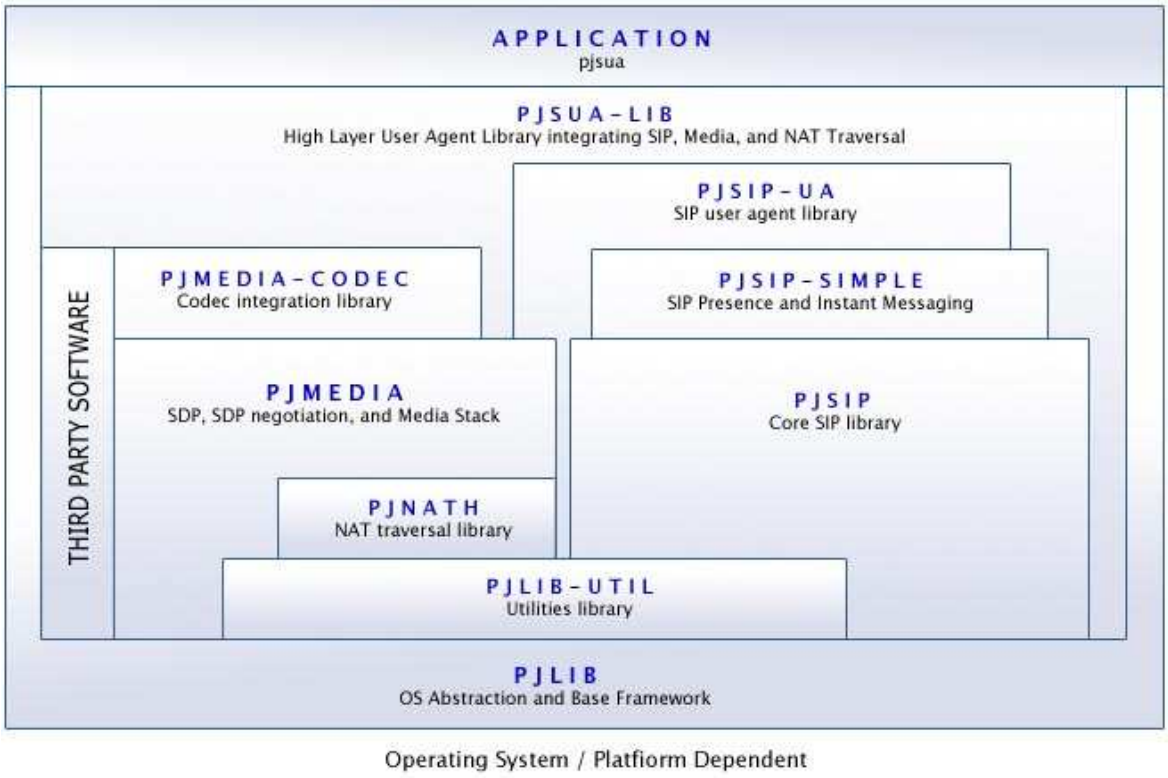


Figura 3.10 - Implementação por camadas PJSIP [11]

A maior desvantagem é o facto de se tornar um pouco complicado o uso desta *stack*, principalmente numa aplicação gráfica para *Windows*, já que está totalmente escrita em C. Contudo foi criado um SDK (*Software Development Kit*) baseado em PJSIP que o torna compatível com C# e, mais propriamente, a plataforma ".NET". O seu nome é o *SIPekSDK*. Esta compatibilidade resulta de um projecto "*wrapper*" que se trata de uma adaptação da interface das classes, permitindo que objectos com interfaces incompatíveis possam interagir. Deste modo, é possível fazer programas em C# usando a biblioteca PJSIP e o *software Microsoft Visual Studio 2008*.

Escrever um programa em C# tem como vantagens o facto de ser uma linguagem orientada a objectos, ao contrário de, por exemplo, C ou *Visual Basic*. Além disto é orientada a eventos, isto é, o fluxo do programa é guiado por uma orientação externa. Outra vantagem é a possibilidade de desenvolvimento em ambientes *Windows*, que é bastante popular. Por último, torna fácil a validação de dados e tratamento de erros.

O SDK (*SIPekSDK*) apresenta 3 módulos: *pjsipDLL*, *pjsipWrapper* e *CallControl*. O primeiro (*pjsipDLL*) trata-se de um projecto C++ integrado na solução PJSIP. A sua saída é uma biblioteca DLL (*Dynamic-link Library*) que pode ser usada em aplicações ".NET". O segundo (*pjsipWrapper*) é um conjunto de classes adaptadas C# que estão dinamicamente ligadas ao *pjsipDLL*: Oferece um API (*Application Programming Interface*) de baixo nível para aplicações SIP. Por último, o *CallControl*: é um API de alto nível escrito em C#, cujo design é baseado no *pjsipWrapper*. Tenta ser o mais simples possível de ser usado, fazendo com que a implementação VoIP seja intuitiva.

Tal como PJSIP, o Design deste SDK é baseado em camadas: a camada de apresentação, a de lógica de controlo de chamadas e a camada VoIP. Na Figura 3.11 está patente esse esquema.

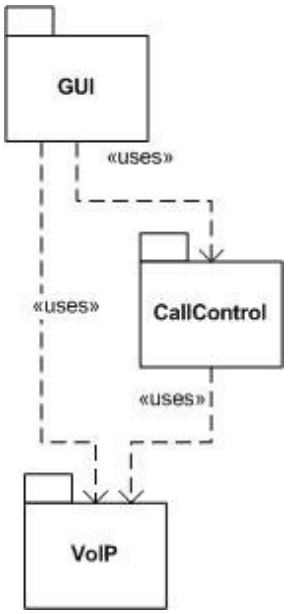


Figura 3.11 - Design em camadas do *SipekSDK* [12]

A camada de apresentação é a parte da aplicação que interage com o utilizador. A camada lógica de controlo de chamadas (*CallControl*) trata do manuseamento das chamadas e das sessões propriamente ditas. Trata-se de uma abstracção para o controle de sessão, suportando múltiplas sessões. Os elementos que a constituem são a classe *CCallManager* e o evento *CallStateRefresh*. Por fim, a camada VoIP que é a camada que liga de uma forma abstracta ao PJSIP. Os seus elementos são: *AbstractWrapper*, que é a interligação entre classes do código em C# e do código em C do PJSIP; *IVoipProxy* que não é orientado à chamada e providencia ao API manipulações básicas e eventos invocados pelo PJSIP; por fim, o *ICallProxyInterface* que é orientado à chamada e providencia ao API manipulação de sessão. Apesar desta implementação em camadas, é possível usar directamente na aplicação,

funções da camada mais baixa sem usar necessariamente uma função de camadas superiores.

As desvantagens do uso de *SIPekSDK* prendem-se com a reestruturação que está a sofrer, havendo algumas funções por implementar. Outra desvantagem inerente a esta é o facto da documentação existente não ser muito completa.

Para finalizar, existe uma função em PJSIP que é bastante útil no desenvolvimento do software, uma vez que é criado um ficheiro de *log* de cada sessão. Lá são registados todos os eventos realizados no programa, desde o registo nos diferentes módulos PJSIP, à inicialização RTP, UDP, bem como todas as mensagens SIP recebidas e enviadas.

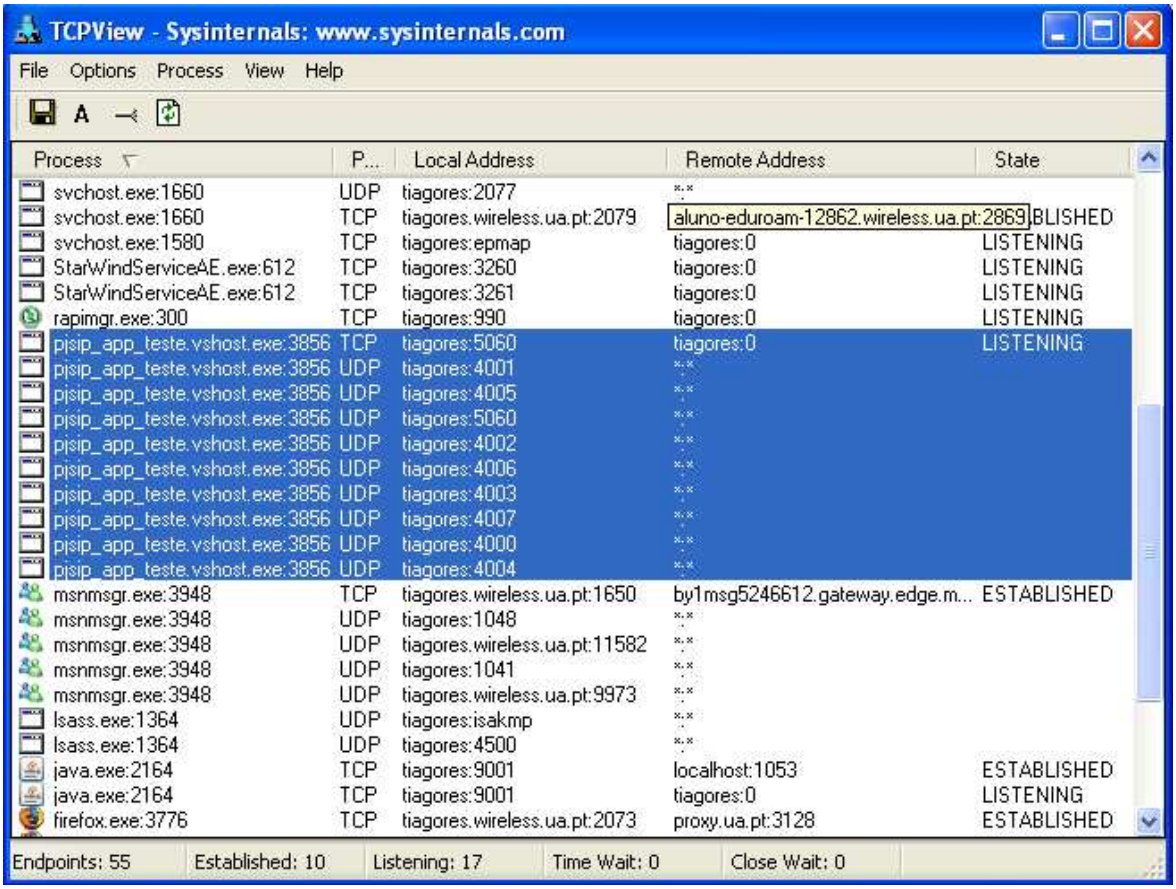
3.3.2 Aspecto da aplicação

Figura 3.12 - Aspecto do programa (primeiro separador)

Na Figura 3.12 está patente o aspecto geral do programa ao abrir. A partir deste momento está apto a receber chamadas, bem como efectuar chamadas directas, isto é, sem o recurso de um servidor. O campo “Status” pode tomar os seguintes valores: NULL, IDLE, CONNECTING, ALERTING, ACTIVE, RELEASED, INCOMING, HOLDING. Cada um tem uma mensagem em português respectiva

Figura 3.13 - Aspecto do programa (segundo separador)

Para usar um servidor SIP, usa-se o separador seguinte, como se pode ver na Figura 3.13. Preenche-se os campos correctamente e carrega-se em "ligar".



Process	P...	Local Address	Remote Address	State
svchost.exe:1660	UDP	tiagores:2077	*:*	
svchost.exe:1660	TCP	tiagores.wireless.ua.pt:2079	aluno-eduroam-12862.wireless.ua.pt:2869	BLISHED
svchost.exe:1580	TCP	tiagores:epmap	tiagores:0	LISTENING
StarWindServiceAE.exe:612	TCP	tiagores:3260	tiagores:0	LISTENING
StarWindServiceAE.exe:612	TCP	tiagores:3261	tiagores:0	LISTENING
rapimgr.exe:300	TCP	tiagores:990	tiagores:0	LISTENING
pjsip_app_teste.vshost.exe:3856	TCP	tiagores:5060	tiagores:0	LISTENING
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4001	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4005	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:5060	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4002	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4006	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4003	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4007	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4000	*:*	
pjsip_app_teste.vshost.exe:3856	UDP	tiagores:4004	*:*	
msnmsgr.exe:3948	TCP	tiagores.wireless.ua.pt:1650	by1msg5246612.gateway.edge.m...	ESTABLISHED
msnmsgr.exe:3948	UDP	tiagores:1048	*:*	
msnmsgr.exe:3948	UDP	tiagores.wireless.ua.pt:11582	*:*	
msnmsgr.exe:3948	UDP	tiagores:1041	*:*	
msnmsgr.exe:3948	UDP	tiagores.wireless.ua.pt:9973	*:*	
lsass.exe:1364	UDP	tiagores:isakmp	*:*	
lsass.exe:1364	UDP	tiagores:4500	*:*	
java.exe:2164	TCP	tiagores:9001	localhost:1053	ESTABLISHED
java.exe:2164	TCP	tiagores:9001	tiagores:0	LISTENING
firefox.exe:3776	TCP	tiagores.wireless.ua.pt:2073	proxy.ua.pt:3128	ESTABLISHED

Figura 3.14 - "*Screenshot*" do programa *TCPView*

Na Figura 3.14 é possível verificar qual o estado das portas (*Sockets*) abertas. Trata-se do programa *TCPView*, que é uma aplicação bastante leve e que permite ver o estado das portas, bem como fechá-las, caso seja necessário.

Figura 3.15 - Estado do programa após ligar a um servidor

Na Figura 3.15 podemos verificar que o programa se ligou com sucesso ao servidor, após receber a mensagem “200 OK”. A partir deste momento, ao efectuar chamadas, elas passarão pelo servidor. Isto está salvaguardado no código.

4 Teste

Este capítulo pretende descrever o funcionamento do serviço implementado, nomeadamente através de capturas de pacotes e de diagramas. Desta forma, será possível compreender melhor o protocolo SIP, bem como o sistema em geral.

4.1 Registo no servidor

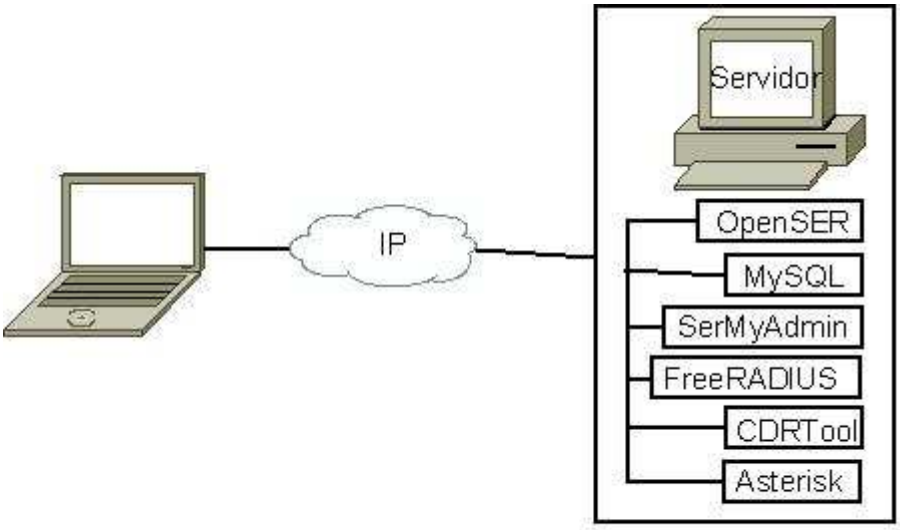


Figura 4.1 - Cenário de teste do registo no servidor

O cenário deste teste é bastante simples: trata-se de um cliente numa máquina e o servidor noutra, ambos numa rede IP. As capturas serão efectuadas ao nível do servidor, usando a ferramenta *ngrep*, que é bastante simples de utilizar e leve. O comando usado foi "*sudo ngrep –q –p 5060*". As seguintes capturas foram resumidas.

```
interface: eth0 (193.136.92.0/255.255.254.0)
filter: (ip or ip6) and ( port 5060 )

U 193.136.93.161:5060 -> 193.136.93.90:5060
REGISTER sip:itav90.av.it.pt SIP/2.0.
Via: SIP/2.0/UDP 193.136.93.161:5060;rport;branch=z9hG4bKPj116091e000b947faa9cd96ed3fc94dcf.
Max-Forwards: 70.
From: <sip:user1@itav90.av.it.pt>;tag=e7f85dd3b09d4cd9a9df16ed5bc22bc3.
To: <sip:user1@itav90.av.it.pt>.
Call-ID: a659363fe4b64e94b44b6389049da930.
CSeq: 40837 REGISTER.
User-Agent: SIPek on PJSUA v0.8.0/win32.
Contact: <sip:user1@193.136.93.161:5060;transport=UDP>.
Expires: 3600.
Content-Length:  0
```

Nesta captura acima, o cliente (endereço IP 193.136.93.161) envia uma mensagem REGISTER para o servidor (endereço IP 193.136.93.90).

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
SIP/2.0 401 Unauthorized.
Via: SIP/2.0/UDP
193.136.93.161:5060;rport=5060;branch=z9hG4bKPj116091e000b947faa9cd96ed3fc94dcf.
From: <sip:user1@itav90.av.it.pt>;tag=e7f85dd3b09d4cd9a9df16ed5bc22bc3.
To: <sip:user1@itav90.av.it.pt>;tag=329cfeaa6ded039da25ff8cbb8668bd2.81be.
(…)
```

O pedido será recusado uma vez que o servidor está configurado para utilizar o modo "*digest*" como modo de autenticação.

```
U 193.136.93.161:5060 -> 193.136.93.90:5060
REGISTER sip:itav90.av.it.pt SIP/2.0.
(…)
Authorization: Digest username="user1", realm="itav90.av.it.pt",
nonce="490b3944eda6bb0bc40be59802187f818af5a431", uri="sip:itav90.av.it.pt",
response="0eece8794efc4fd06494f92122c51b97", cnonce="8d7f05d2685b44e8a9252209fdcda4a6",
qop=auth, nc=00000001.
Content-Length: 0.
```

Visto isto, o cliente volta a enviar os dados, desta feita satisfazendo os requisitos do servidor, e utilizando o modo "*digest*" na autenticação.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
SIP/2.0 200 OK.
(…)
```

A resposta do servidor, tendo as condições satisfeitas e os dados coincidirem com a sua base de dados, é "200 OK", o que significa que o utilizador está autenticado.

4.2 Chamada entre dois telefones IP

O cenário utilizado para o teste da implementação de um serviço é o seguinte:

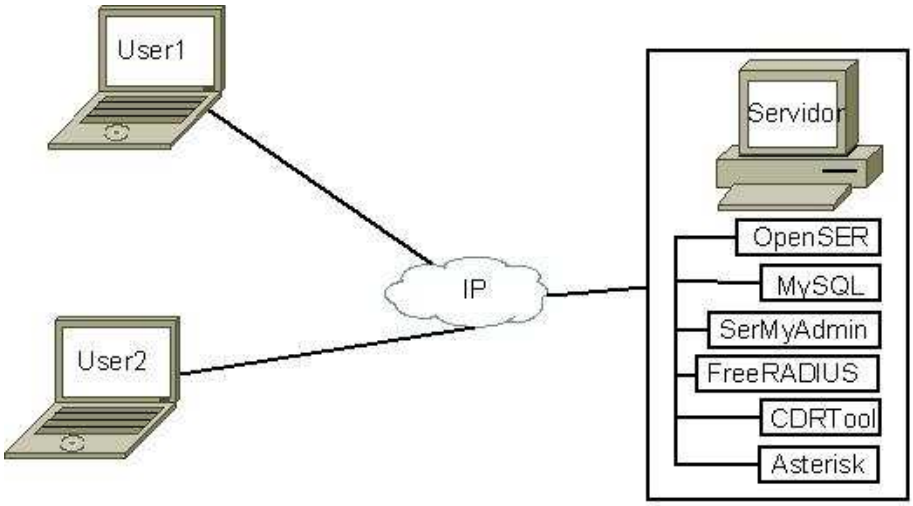


Figura 4.2 - Cenário de teste de uma chamada entre 2 utilizadores

Foram utilizados três computadores ligados numa LAN (*Local Area Network*). O servidor tem instalado o *software* descrito na secção 3.2 e os clientes têm o *software* descrito na secção 3.3. Nos clientes SIP, no campo "servidor" será colocado o endereço da primeira máquina (itav90.av.it.pt) e os campos de utilizador serão os previamente adicionados à base de dados (user1:test1 e user2:test2).

Para fazer as capturas serão utilizadas fundamentalmente duas ferramentas. No *Windows* será utilizado o *Wireshark*, que é um programa bastante popular para analisar protocolos. No servidor, que utiliza o sistema operativo *Linux*, será utilizada a ferramenta *ngrep*, tal como no teste anterior.

O programa *Wireshark* tem algumas opções bastante úteis para a análise de estatísticas em chamadas IP: estatísticas SIP e estatísticas de chamadas VoIP. A primeira está patente na Figura 4.3. Esta figura mostra a estatística de uma chamada entre 2 clientes SIP.

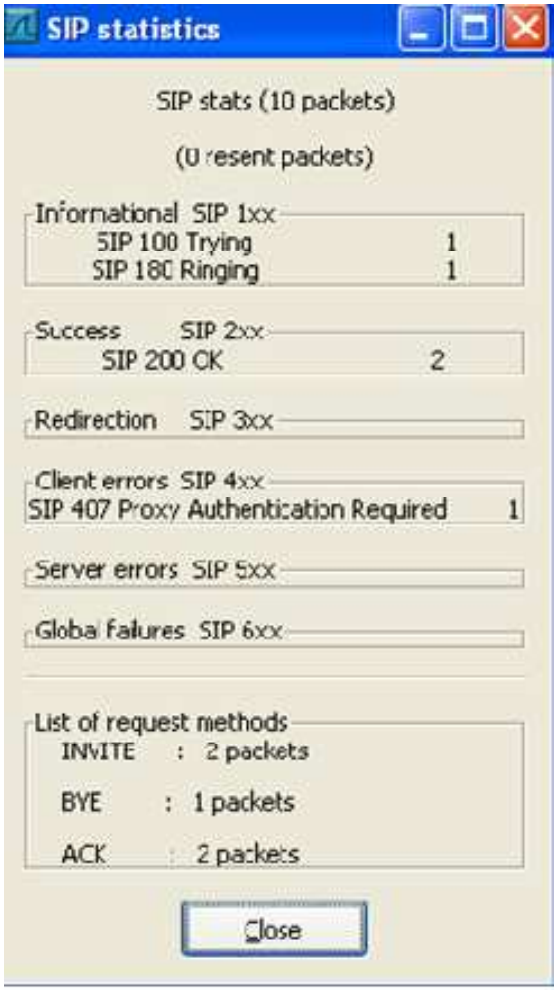


Figura 4.3 Estatísticas SIP efectuando uma chamada.

Como se pode verificar, foram capturados 10 pacotes SIP. Eles são separados conforme a sua categoria, conforme se pode ver. Apesar de aparecer uma mensagem

de erro de cliente, esta está prevista no código de funcionamento do *OpenSER*, que pede autenticação ao utilizador antes deste efectuar uma chamada.

A segunda característica que foi destacada deste programa é a análise de chamadas VoIP e está ilustrada na Figura 4.4.

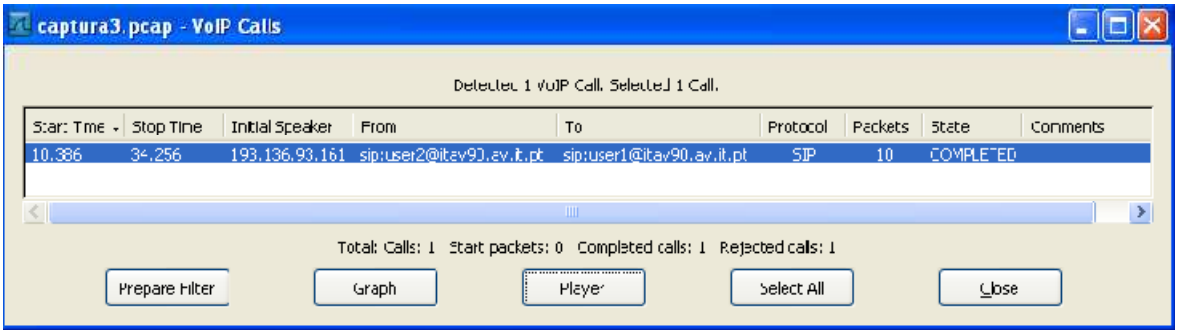


Figura 4.4 - Análise de chamadas VoIP

Nesta captura foi detectada apenas uma chamada entre os dois clientes. Seleccionando essa conversa e carregando do botão em baixo "*Graph*", é criada um diagrama com as transacções SIP. Este diagrama é bastante útil para analisar o funcionamento de uma chamada IP e avaliar o bom funcionamento do servidor. A Figura 4.5 ilustra isto mesmo.

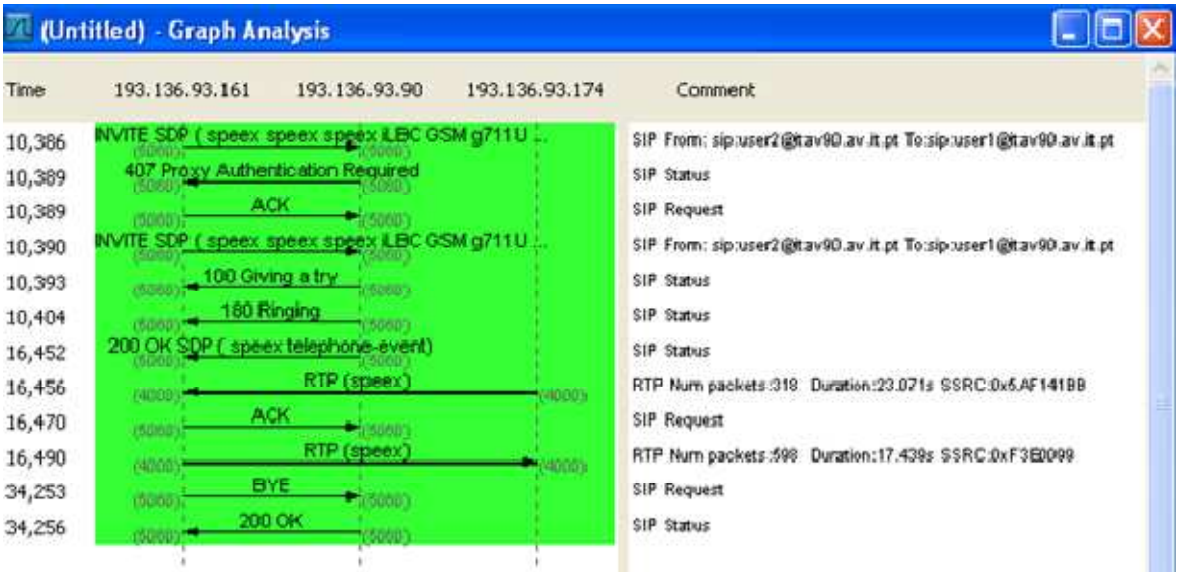


Figura 4.5 - Análise gráfica das transacções SIP efectuando uma chamada

O cliente vai enviar o INVITE que vai passar pelo servidor. Este, como já foi referido, vai-lhe pedir a autenticação para poder completar o pedido. O cliente vai responder com a mensagem "ACK" e vai repetir o INVITE, desta feita com os dados para se poder autenticar. O servidor responde com as mensagens "100 *Giving a Try*" e "180 *Ringing*", o que significa que o outro cliente recebeu o INVITE. Quando a chamada é atendida no outro partido, é enviada a mensagem "200 OK" que passa pelo servidor

antes de chegar ao primeiro partido. A partir deste momento é recebido o *stream* RTP, e enviada a mensagem "ACK" que indica que o emissor recebeu a mensagem "200 OK". Agora este começa a enviar também. O fim da chamada é sinalizado com a mensagem BYE, que neste caso foi enviada pelo utilizador que iniciou a chamada, ao qual receberá a mensagem "200 OK".

Este diagrama está incompleto, uma vez que o *Wireshark* está instalado no computador do cliente que efectua a chamada, portanto vai capturar os pacotes no seu ponto de vista. Para ter um diagrama completo, terá que se analisar os pacotes sob o ponto de vista de todos os intervenientes na chamada: os clientes e o servidor. De seguida será feita a mesma análise recebendo uma chamada.

Na Figura 4.6 está patente a estatística de pacotes SIP capturados ao receber uma chamada.

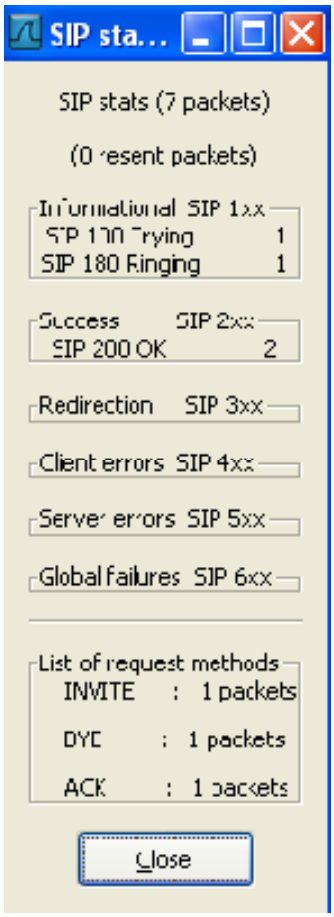


Figura 4.6 - Estatísticas SIP recebendo uma chamada

De destacar que o número de pacotes é inferior, uma vez que a autenticação referida anteriormente não é pedida ao cliente que recebe a chamada. Na Figura 4.7 está patente isso mesmo.

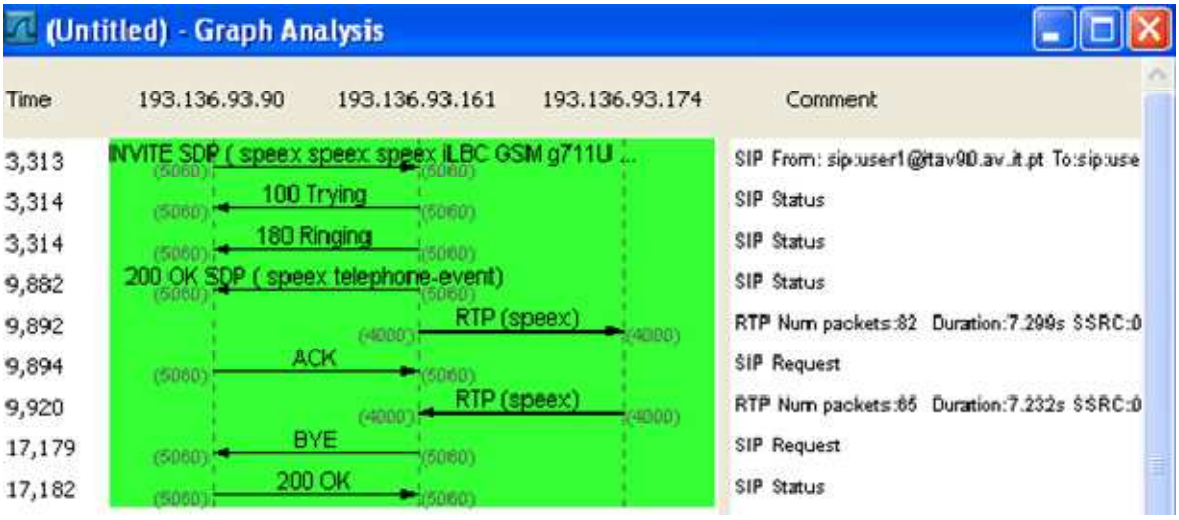


Figura 4.7 - Gráfico de transacções SIP recebendo uma chamada

O esquema está um pouco diferente do primeiro, uma vez que a primeira mensagem que é recebida provém do servidor. Portanto o programa vai interpretar como sendo este o cliente que está a fazer a chamada para si. Como se pode verificar, o cliente recebe o INVITE e responde com as mensagens "100 *Trying*" e "180 *Ringing*". Quando o utilizador atende a chamada, a mensagem "200 OK" é enviada. A partir daí o *stream* RTP é estabelecido entre o cliente que recebe a chamada e o que a inicia. Quando a mensagem "200 OK" chega ao outro partido, este responde com a mensagem "ACK" e estabelece o *stream* RTP.

Outra funcionalidade interessante do *Wireshark* é a possibilidade de analisar os *streams* RTP. Isto é útil na medida em que os pacotes RTP vão ser determinantes na qualidade de voz. Apesar de não haver nenhuma recomendação que determine os parâmetros de uma boa ou má qualidade na transmissão, é considerado que a latência terá que ser menor que 150 ms, o *jitter* inferior a 20 ms e a perda de pacotes abaixo dos 3% [9]. O factor latência poderá ser superior, mas a interactividade de uma conversação vai-se perder. Relativamente ao *jitter*, este vai deteriorar a qualidade da voz. A Figura 4.8 ilustra a função do programa para analisar os *streams* RTP.

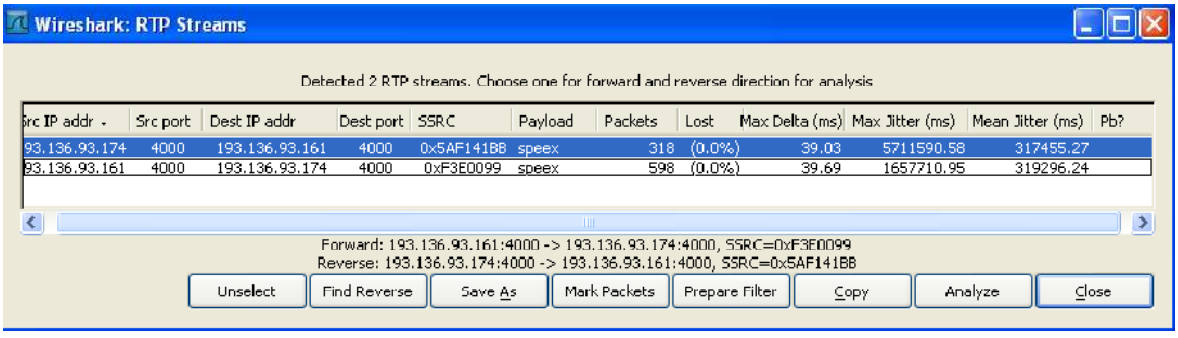


Figura 4.8 - *Streams* RTP

Na imagem acima, deve-se seleccionar o stream, seleccionar o inverso e carregar em "*analyse*". Agora é possível analisar, como se pode verificar na Figura 4.9, pacote a pacote o *jitter*, latência (delta), largura de banda IP e perda de pacotes.

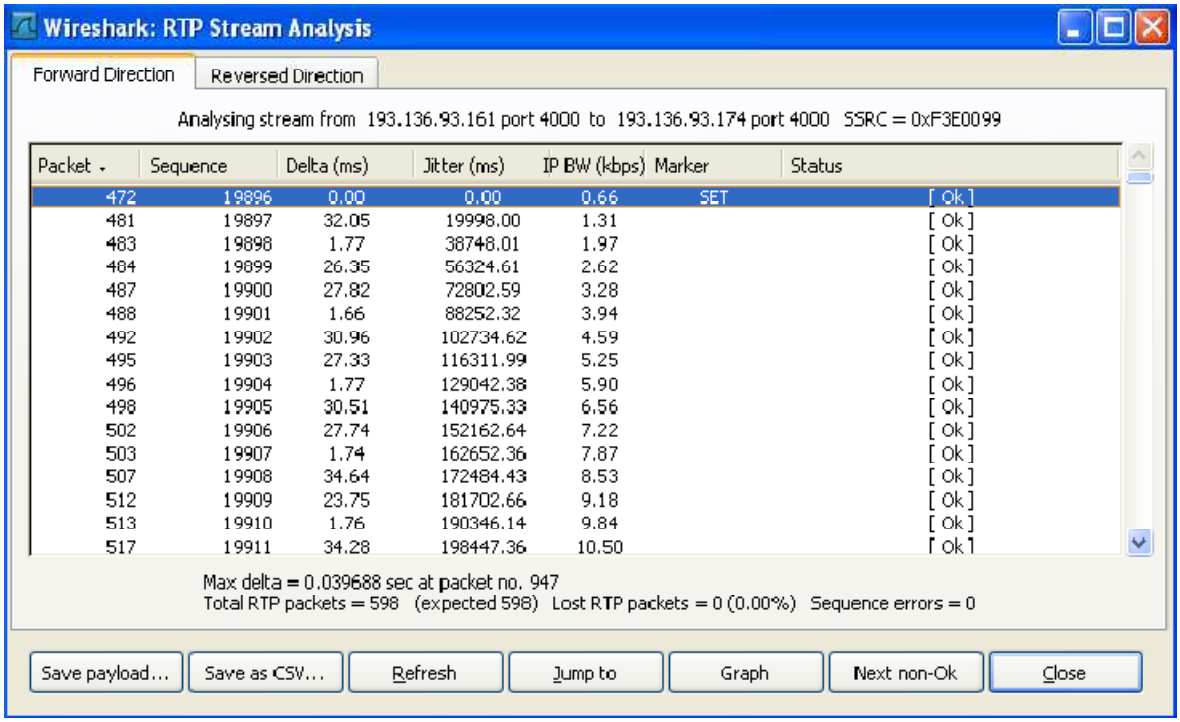


Figura 4.9 - Análise do *stream* RTP

Para analisar ao nível do servidor é usado o comando "*sudo ngrep –q –p 5060*". As mensagens são as seguintes (resumidas).

```
interface: eth0 (193.136.92.0/255.255.254.0)
match: 5060

U 193.136.93.161:5060 -> 193.136.93.90:5060
  INVITE sip:user1@itav90.av.it.pt SIP/2.0..Via: SIP/2.0/UDP 193.136.93.161:5
  060;rport;branch=z9hG4bKPj33435502de4c4be6a1aa562649a3f5c2..Max-Forwards: 7
  0..From: sip:user2@itav90.av.it.pt;tag=6f98f5236f574beabb7ca64b0ebd01bc..To
  : sip:user1@itav90.av.it.pt..Contact: <sip:user2@193.136.93.161:5060;transp
  ort=UDP>..Call-ID: 32144e09b5c8454286d9e49c7375a8a7..CSeq: 11538 INVITE..Al
  low: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, PUBLISH, R
  EFER, MESSAGE, OPTIONS..Supported: replaces, 100rel, norefersub..User-Agent
  : SIPek on PJSUA v0.8.0/win32..Content-Type: application/sdp..Content-Lengt
  h:   441....v=0..o=- 3434713340 3434713340 IN IP4 193.136.93.161..s=pjmedia
  ..c=IN IP4 193.136.93.161..t=0 0..a=X-nat:0..m=audio 4000 RTP/AVP 103 102 1
  04 117 3 0 8 101..a=rtcp:4001 IN IP4 193.136.93.161..a=rtpmap:103 speex/160
  00..a=rtpmap:102 speex/8000..a=rtpmap:104 speex/32000..a=rtpmap:117 iLBC/80
  00..a=fmtp:117 mode=20..a=rtpmap:3 GSM/8000..a=rtpmap:0 PCMU/8000..a=rtpmap
  :8 PCMA/8000..a=sendrecv..a=rtpmap:101 telephone-event/8000..a=fmtp:101 0-1
  5..
```

Esta é a mensagem de INVITE do "*user2*" para o "*user1*". Nesta mensagem já estão contidos os parâmetros SDP. Ela não vai directamente para o utilizador, já que tem

que passar primeiro no servidor. Portanto o endereço IP de destino é o do *proxy*: 193.136.93.90.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 407 Proxy Authentication Required..Via: SIP/2.0/UDP 193.136.93.161:
  5060;rport=5060;branch=z9hG4bKPj33435502de4c4be6a1aa562649a3f5c2..From: sip
  :user2@itav90.av.it.pt;tag=6f98f5236f574beabb7ca64b0ebd01bc..To: sip:user1@
  itav90.av.it.pt;tag=329cfeaa6ded039da25ff8cbb8668bd2.91d3..Call-ID: 32144e0
  9b5c8454286d9e49c7375a8a7..CSeq: 11538 INVITE..Proxy-Authenticate: Digest r
  ealm="itav90.av.it.pt", nonce="490f13cfc5594dc94903f9bfbef00a39e7432e4e", q
  op="auth"..Server: OpenSER (1.2.2-notls (i386/linux))..Content-Length: 0...
```

Nesta mensagem o servidor pede ao utilizador os dados de autenticação para prosseguir com o pedido. A mensagem é originada pelo servidor e vai para o cliente que iniciou o pedido de chamada.

```
U 193.136.93.161:5060 -> 193.136.93.90:5060
  ACK sip:user1@itav90.av.it.pt SIP/2.0..Via: SIP/2.0/UDP 193.136.93.161:5060
  ;rport;branch=z9hG4bKPj33435502de4c4be6a1aa562649a3f5c2..Max-Forwards: 70..
  From: sip:user2@itav90.av.it.pt;tag=6f98f5236f574beabb7ca64b0ebd01bc..To: s
  ip:user1@itav90.av.it.pt;tag=329cfeaa6ded039da25ff8cbb8668bd2.91d3..Call-ID
  : 32144e09b5c8454286d9e49c7375a8a7..CSeq: 11538 ACK..Content-Length:  0....
```

Esta é uma mensagem “ACK”, que o cliente envia para o servidor indicando que recebeu a última mensagem.

```
U 193.136.93.161:5060 -> 193.136.93.90:5060
  INVITE sip:user1@itav90.av.it.pt SIP/2.0..Via: SIP/2.0/UDP 193.136.93.161:5
  060;rport;branch=z9hG4bKPj6b376154f10441d79024544b5b1012d1..Max-Forwards: 7
  0..From: sip:user2@itav90.av.it.pt;tag=6f98f5236f574beabb7ca64b0ebd01bc..To
  : sip:user1@itav90.av.it.pt..Contact: <sip:user2@193.136.93.161:5060;transp
  ort=UDP>..Call-ID: 32144e09b5c8454286d9e49c7375a8a7..CSeq: 11539 INVITE..Al
  low: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, PUBLISH, R
  EFER, MESSAGE, OPTIONS..Supported: replaces, 100rel, norefersub..User-Agent
  : SIPek on PJSUA v0.8.0/win32..Proxy-Authorization: Digest username="user2"
  , realm="itav90.av.it.pt", nonce="490f13cfc5594dc94903f9bfbef00a39e7432e4e"
  , uri="sip:user1@itav90.av.it.pt", response="0ff359fea9a452a354533476a1f439
  95", cnonce="63f150df49334d68ae3129c58f94e459", qop=auth, nc=00000001..Cont
  ent-Type: application/sdp..Content-Length:   441....v=0..o=- 3434713340 343
  4713340 IN IP4 193.136.93.161..s=pjmedia..c=IN IP4 193.136.93.161..t=0 0..a
  =X-nat:0..m=audio 4000 RTP/AVP 103 102 104 117 3 0 8 101..a=rtcp:4001 IN IP
  4 193.136.93.161..a=rtpmap:103 speex/16000..a=rtpmap:102 speex/8000..a=rtpm
  ap:104 speex/32000..a=rtpmap:117 iLBC/8000..a=fmtp:117 mode=20..a=rtpmap:3
  GSM/8000..a=rtpmap:0 PCMU/8000..a=rtpmap:8 PCMA/8000..a=sendrecv..a=rtpmap:
  101 telephone-event/8000..a=fmtp:101 0-15..
```

Aqui está patente a nova mensagem de INVITE semelhante à primeira, mas desta feita com os dados de autenticação contidos no corpo da mensagem.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 100 Giving a try..Via: SIP/2.0/UDP 193.136.93.161:5060;rport=5060;b
  ranch=z9hG4bKPj6b376154f10441d79024544b5b1012d1..From: sip:user2@itav90.av.
  it.pt;tag=6f98f5236f574beabb7ca64b0ebd01bc..To: sip:user1@itav90.av.it.pt..
  Call-ID: 32144e09b5c8454286d9e49c7375a8a7..CSeq: 11539 INVITE..Server: Open
  SER (1.2.2-notls (i386/linux))..Content-Length: 0....
```

Ao ser autenticado o pedido, o servidor responde ao cliente que irá proceder ao pedido. A sua resposta é "100 *Giving a try*".

```
U 193.136.93.90:5060 -> 193.136.93.174:5060
  INVITE sip:user1@193.136.93.174:5060;transport=UDP SIP/2.0
 (…)
```

O INVITE é encaminhado para o segundo cliente. Como se pode verificar, o endereço da fonte é o do servidor e o do destino é do novo cliente.

```
U 193.136.93.174:5060 -> 193.136.93.90:5060
  SIP/2.0 100 Trying..Via: SIP/2.0/UDP 193.136.93.90;received=193.136.93.90;
(…)
```

O segundo cliente vai responder para o servidor indicando que a mensagem chegou e está a processá-la.

```
U 193.136.93.174:5060 -> 193.136.93.90:5060
  SIP/2.0 180 Ringing..Via: SIP/2.0/UDP 193.136.93.90;received=193.136.93.90;
(…)
```

Desta feita o mesmo cliente indica ao servidor que está a chamar.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 180 Ringing..Via: SIP/2.0/UDP 193.136.93.161:5060;rport=5060;
 (…)
```

Consequentemente, o servidor vai redireccionar esta mensagem para o primeiro cliente, o que efectuou a chamada.

```
U 193.136.93.174:5060 -> 193.136.93.90:5060
  SIP/2.0 200 OK..Via: SIP/2.0/UDP 193.136.93.90;received=193.136.93.90;
(…)
```

Agora o segundo cliente atendeu a chamada, portanto responde com a mensagem "200 OK". A partir desta altura ele vai enviar o seu *stream* RTP para este cliente, contudo esses dados não foram capturados, uma vez que são trocados entre os dois clientes sem a participação do servidor.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 200 OK..Via: SIP/2.0/UDP 193.136.93.161:5060;rport=5060;
(…)
```

Por sua vez, o servidor vai redireccionar o pedido para o cliente remetente.

```
U 193.136.93.161:5060 -> 193.136.93.90:5060
  ACK sip:user1@193.136.93.174:5060;transport=UDP SIP/2.0..Via: SIP/2.0/UDP
(…)
```

O primeiro cliente vai responder "ACK", indicando que recebeu a última mensagem com sucesso e inicia também o seu fluxo de informação.

```
U 193.136.93.90:5060 -> 193.136.93.174:5060
  ACK sip:user1@193.136.93.174:5060;transport=UDP SIP/2.0
(…)
```

Como seria de esperar, o servidor redirecciona o pedido para o segundo cliente. Neste momento não haverá mais sinalização, uma vez que os dois clientes estão ligados directamente entre si através de *streams* RTP. Agora será visto o que acontece quando se termina uma chamada.

```
U 193.136.93.174:5060 -> 193.136.93.90:5060
  BYE sip:user2@193.136.93.161:5060;transport=UDP SIP/2.0..Via: SIP/2.0/UDP
 (…)
```

Neste caso, o segundo utilizador terminou a chamada, enviando a mensagem “BYE” e parando o fluxo RTP. Esta mensagem vai do cliente para o servidor.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  BYE sip:user2@193.136.93.161:5060;transport=UDP SIP/2.0
(…)
```

O servidor redirecciona para o outro cliente.

```
U 193.136.93.161:5060 -> 193.136.93.90:5060
  SIP/2.0 200 OK..Via: SIP/2.0/UDP 193.136.93.90;received=193.136.93.90;
(…)
```

Este cliente responde com a mensagem “200 OK” e termina o fluxo.

```
U 193.136.93.90:5060 -> 193.136.93.174:5060
  SIP/2.0 200 OK..Via: SIP/2.0/UDP 193.136.93.174:5060;rport=5060;
(…)
```

Agora o servidor reenvia a mensagem para o cliente que terminou a chamada.

Analisadas as mensagens dos pontos de vista do emissor, receptor e do servidor, é possível fazer um diagrama rigoroso da chamada.

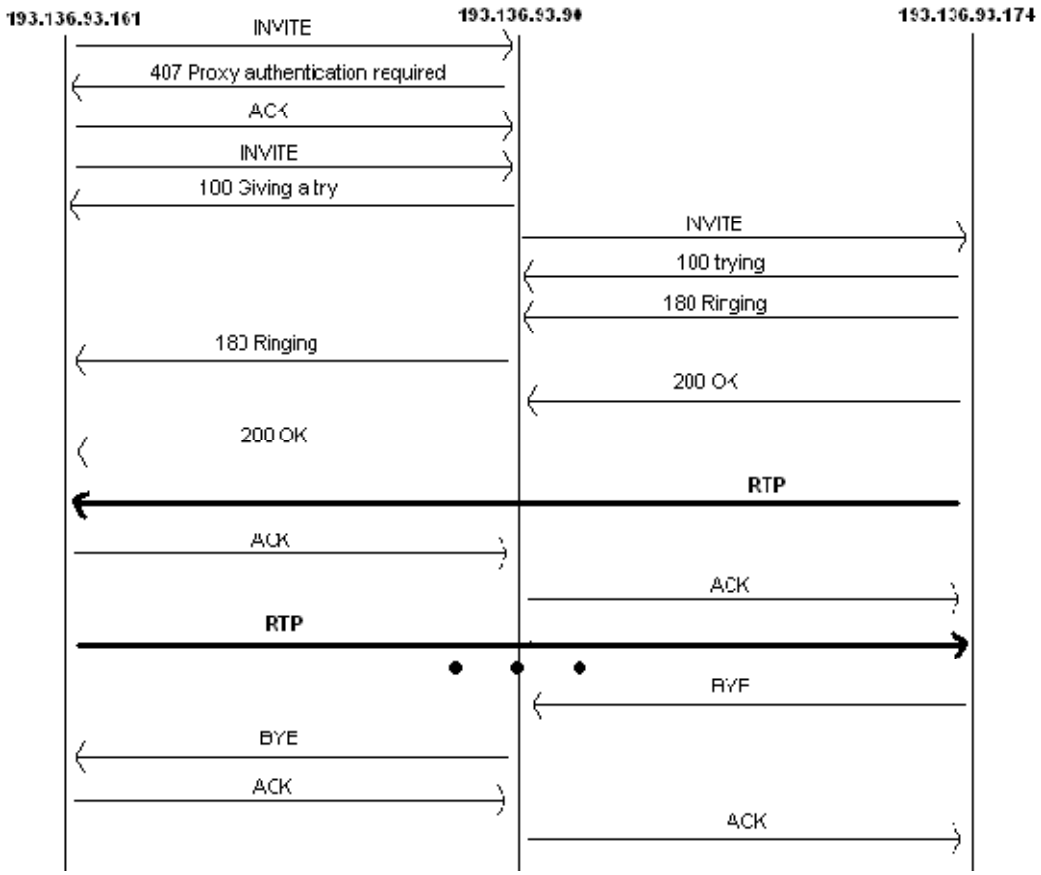


Figura 4.10 - Diagrama de um diálogo SIP

Em jeito de observação, apesar dos fluxos RTP estarem representados separadamente, tanto no esquema da Figura 4.10, como nas figuras Figura 4.5 e Figura 4.7, são considerados um fluxo bidireccional, uma vez que são enviados e recebidos nas mesmas portas em ambos os lados. A representação procurou ser a mais rigorosa possível tendo em conta a sequência de mensagens.

4.3 Chamada entre um *softphone* e um telefone PSTN

Neste caso o cenário será um pouco diferente dos anteriores. Será necessário um cliente, um servidor, um *gateway* para a PSTN e um telefone ligado à rede. Como foi visto na implementação, o *gateway* é baseado em *software* e está no mesmo computador que o servidor, portanto são efectuadas capturas à porta 5060 e também 5061, que é a porta que o *Asterisk* está a usar. Também é necessária uma conta com crédito no "*Voipbuster*" para que se possa fazer a chamada.

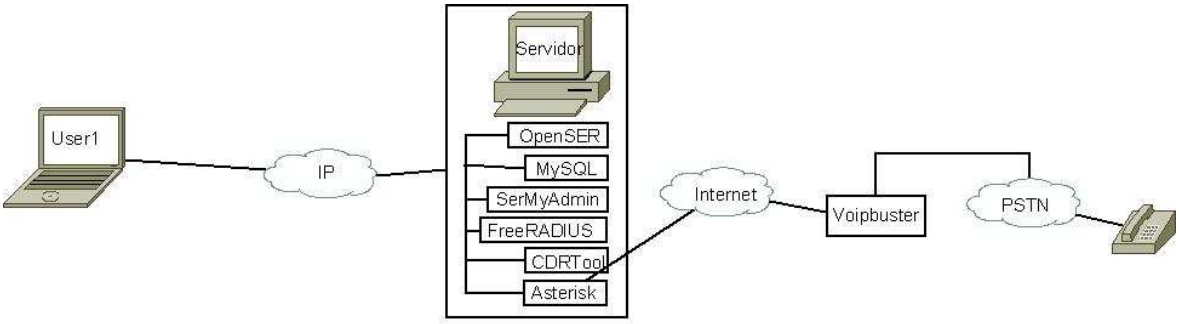


Figura 4.11 - Cenário de teste de uma chamada entre um *Softphone* e um telefone normal

Em primeiro lugar, o *Asterisk* terá que se registar no *Voipbuster*. O ficheiro *sip.conf* que se encontra em anexo mostra como isto é feito. A seguinte captura mostra o registo.

```
U 193.136.93.90:5061 -> 194.120.0.198:5060
REGISTER sip:sip.voipbuster.com SIP/2.0.
Via: SIP/2.0/UDP 193.136.93.90:5061;branch=z9hG4bK23840e58;rport.
From: <sip:itav90@sip.voipbuster.com>;tag=as249e9388.
To: <sip:itav90@sip.voipbuster.com>.
Call-ID: 150756cd6653089d7a2acafb485199ce@127.0.1.1.
CSeq: 215 REGISTER.
User-Agent: Asterisk PBX.
Max-Forwards: 70.
Authorization: Digest username="itav90", realm="sip.voipbuster.com", algorithm=MD5,
uri="sip:sip.voipbuster.com", nonce="704850406", response="2daa2332f9e3cd23d2fa95f2e540c324".
Expires: 120.
Contact: <sip:s@193.136.93.90:5061>.
Event: registration.
Content-Length: 0.
```

Aqui deu-se o pedido para registo no servidor sip.voipbuster.com. Na captura seguinte estará a resposta.

```
U 194.120.0.198:5060 -> 193.136.93.90:5061
SIP/2.0 200 Ok.
Via: SIP/2.0/UDP 193.136.93.90:5061;branch=z9hG4bK23840e58;rport.
From: <sip:itav90@sip.voipbuster.com>;tag=as249e9388.
To: <sip:itav90@sip.voipbuster.com>.
Contact: <sip:s@193.136.93.90:5061>;expires=120.
Call-ID: 150756cd6653089d7a2acafb485199ce@127.0.1.1.
CSeq: 215 REGISTER.
Server: (Very nice Sip Registrar/Proxy Server).
Allow: ACKBYE,CANCEL,INVITE,REGISTER,OPTIONS,INFO,MESSAGE.
Content-Length: 0.
```

A resposta foi “200 OK”, portanto o *Asterisk* registou-se com sucesso no servidor, tal como se fosse um UAC normal.

Agora, para efectuar a chamada para o número público, o processo vai ser semelhante ao visto na secção 4.2. A seguir será visto o funcionamento do *Asterisk*, fazendo uma ligação ao programa de forma a aceder à sua consola de comandos e

usando o modo *Debug* de forma a visualizar as operações efectuadas. O comando para o fazer está patente na primeira linha da captura seguinte.

```
sudo asterisk –rvvvvvvvvv
Asterisk 1.4.21.2, Copyright (C) 1999 - 2008 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
  == Parsing '/etc/asterisk/asterisk.conf': Found
  == Parsing '/etc/asterisk/extconfig.conf': Found
Connected to Asterisk 1.4.21.2 currently running on voip-server (pid = 4290)
voip-server*CLI>
Verbosity was 6 and is now 10
[Kvoip-server*CLI>
   -- Remote UNIX connection
```

No momento em que é feita uma chamada para um utilizador da rede telefónica, o servidor *OpenSER* vai encaminhar a chamada para o *Asterisk*. O processo de convite não será mostrado, uma vez que é semelhante ao da secção 4.2, com a excepção do endereço ser sip:00351234105105@itav90.av.it.pt. Na captura seguinte está patente o encaminhamento da mensagem desde o *Asterisk* (na porta 5061) para o servidor externo (*Voipbuster*)

```
U 193.136.93.90:5061 -> 194.120.0.198:5060
INVITE sip:234105105@sip.voipbuster.com SIP/2.0.
(…)
User-Agent: Asterisk PBX.
(…)
```

Na captura seguinte pode-se verificar o *Asterisk* a lidar com a chamada.

```
[Kvoip-server*CLI>
   -- Executing [00351234105105@sipincoming:1] Dial("SIP/itav90.av.it.pt-081e2908",
"SIP/00351234105105@sipproxy") in new stack
[Kvoip-server*CLI>
   -- Called 00351234105105@sipproxy
[Kvoip-server*CLI>
   -- SIP/sipproxy-081e6880 is making progress passing it to SIP/itav90.av.it.pt-081e2908
[Kvoip-server*CLI>
```

Após o convite, o estabelecimento da chamada vai ser semelhante ao verificado na secção anterior, contudo é entre o *Asterisk* e o *Voipbuster*.

```
U 194.120.0.198:5060 -> 193.136.93.90:5061
SIP/2.0 100 Trying.
(…)
U 194.120.0.198:5060 -> 193.136.93.90:5061
SIP/2.0 183 Session progress.
(…)
U 194.120.0.198:5060 -> 193.136.93.90:5061
SIP/2.0 200 OK
```

```
(…)
```

Mas como é óbvio, as mensagens vão ser encaminhadas para o *OpenSER* e este vai entregá-las ao utilizador, como está patente na captura seguinte.

```
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 100 Giving a try..Via: SIP/2.0/UDP 193.136.93.161:5060
 (…)
U 193.136.93.90:5060 -> 193.136.93.161:5060
  SIP/2.0 183 Session Progress..Via: SIP/2.0/UDP 193.136.93.161:5060;rport=50
(…)
U 194.120.0.198:5060 -> 193.136.93.90:5061
  SIP/2.0 200 OK..Via: SIP/2.0/UDP 193.136.93.90:5061;
(…)
```

De notar que na última mensagem o campo "VIA" tem o endereço do *Asterisk*, pois a chamada terá que ser efectuada através dele. A seguir pode-se verificar o PBX a lidar com a chamada quando esta foi atendida e, posteriormente, quando foi terminada.

```
   -- SIP/sipproxy-081e6880 answered SIP/itav90.av.it.pt-081e2908
[Kvoip-server*CLI>
  == Spawn extension (sipincoming, 00351234105105, 2) exited non-zero on 'SIP/itav90.av.it.pt-081e2908'
```

As capturas da terminação da chamada não serão demonstradas, uma vez que são bastante semelhantes às verificadas na secção anterior, contudo, tal como se verificou no estabelecimento da chamada, são trocadas mensagens entre o cliente e o *OpenSER,* entre este e o *Asterisk* e, finalmente, entre este último e o servidor exterior.

4.4 Sistema de tarifação

O sistema de tarifação CDRTool é acedido via HTTP, sendo possível configurar todos os parâmetros e consultar os dados. Para verificar o seu funcionamento, acede-se a "http://itav90.av.it.pt/CDRTool" e faz-se a autenticação como administrador. De seguida, a página inicial é a consulta dos CDR (*Call Detail Record*). Deverá seleccionar-se uma data apropriada e fazer a pesquisa. Na Figura 4.12 pode-se verificar o registo de uma chamada e o respectivo custo. Ao carregar no "*Id"* abre informação mais detalhada, como é visível na mesma imagem. Como se pode ver, a chamada durou 6 segundos, sendo taxada a 36 cêntimos por minuto, ficando o preço final 0.036€. Estes valores foram definidos na configuração que pode ser consultada em anexo. Em jeito de observação, a chamada apresentada na figura não diz respeito à chamada estudada na secção anterior.

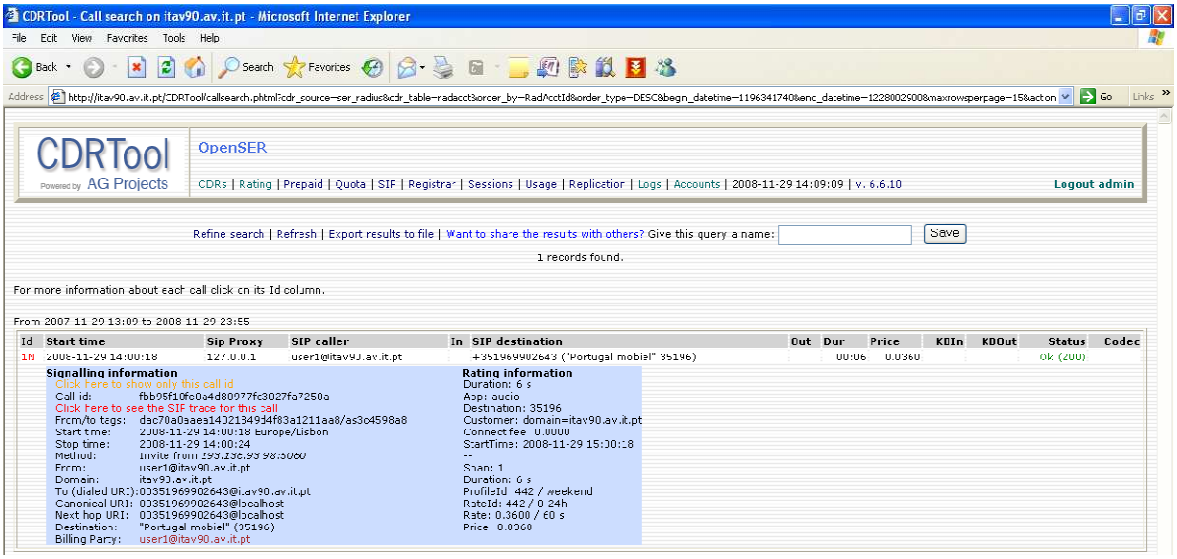


Figura 4.12 – Consulta dos CDR no CDRTool.

5 Conclusões e trabalho futuro

Este capítulo pretende fazer uma interpretação dos resultados obtidos nos testes, uma resenha das dificuldades encontradas e perspectivar o que poderá ser feito de forma a optimizar o sistema.

5.1 Funcionamento do serviço

Neste trabalho foi feito um estudo da tecnologia de voz sobre IP, focando bastante o protocolo SIP. Este é o protocolo utilizado em VoIP com maiores perspectivas de crescimento, maior flexibilidade e maior escalabilidade. Depois do estudo, a vertente prática mostrou o funcionamento de um serviço, através da instalação de um servidor com os serviços básicos de redireccionamento, gestão de utilizadores, integração PSTN e taxação. Qualquer cliente SIP se pode ligar a este serviço, contudo foi desenvolvido um capaz de efectuar as operações básicas. Todas estas ferramentas não apresentaram custos ao nível de licenças de utilização.

A secção 4 demonstrou o funcionamento do servidor na arquitectura pretendida. Ele é capaz de realizar as tarefas básicas de um servidor *Proxy/Registrar*, sendo possível a um cliente registar-se no servidor, este vai guardar as informações na base de dados *MySQL* e, quando chega um pedido é com base nessas informações que este é encaminhado. Caso o pedido seja o convite para efectuar uma chamada, outras considerações são levantadas. É pedido ao cliente a sua autenticação, é verificado o grupo a que este pertence, com o intuito de permitir, ou não a chamada e só depois é que o pedido é satisfeito. A partir do momento que todas as mensagens relativas à transacção são encaminhadas, o servidor deixa de fazer parte da transacção, uma vez que a ligação de dados RTP é feita directamente entre os clientes envolvidos.

Neste campo, as dificuldades encontradas foram mais focadas no funcionamento do *OpenSER* com as outras ferramentas. A sua integração com o *FreeRADIUS* não é trivial, apresentando bastantes detalhes na configuração. A ferramenta *SerMyAdmin* também foi bastante complicada de configurar, principalmente a forma como interage com a base de dados. Finalmente, O *CDRTool* é uma ferramenta bastante complexa e evoluída para a sua configuração por parte de uma pessoa sem experiência. A sua instalação é minuciosa e a configuração é um pouco complexa. Ao nível do código para encaminhar as chamadas, consultando a documentação e uns exemplos práticos, torna-se relativamente simples construir os blocos, mas exige um

conhecimento avançado do protocolo SIP para que os métodos fiquem bem implementados.

Relativamente ao cliente criado, a sua principal característica é a simplicidade. O seu grafismo rege-se mediante linhas simples, bem como o seu funcionamento. Neste contexto, as suas tarefas são o registo no *proxy*, o estabelecimento de chamadas e a recepção das mesmas. De destacar a utilização de uma biblioteca livre que opera numa pilha de protocolos também ela livre. Esta parte do trabalho não seria necessária, na medida em que existem no mercado inúmeras aplicações de voz sobre IP que utilizam SIP e, em grande parte dos casos, estas aplicações não apresentam custos de utilização. Contudo, a opção de criar um cliente próprio foi importante para a aprendizagem em maior detalhe do protocolo em si e da tecnologia de voz sobre IP em geral. Permitiu uma maior compreensão do funcionamento de uma aplicação por camadas e, também, uma perspectiva prática de programação por objectos.

Neste campo as dificuldades foram maiores, na medida em que se exige o conhecimento de linguagens de programação orientadas a objectos que tiveram que ser estudados. Por outro lado, existem bastantes bibliotecas e, por vezes, torna-se complicada a escolha. O facto de se trabalhar em *Windows* também é um factor que introduz dificuldade, já que a maioria das soluções de código aberto são vocacionadas para o sistema operativo *Linux*.

Falando do sistema de taxação, *CDRTool*, trata-se de um sistema bastante evoluído, com muitas características e potencialidades que nem chegaram a ser exploradas. Contudo, a sua instalação foi complicada e a sua integração com as bases de dados que já existiam, nomeadamente do sistema de *RADIUS*, foi longe de ser trivial. A aplicação ficou a funcionar, é possível consultar a base de dados criada pelo *FreeRADIUS* e o sistema efectua um cálculo de uma taxa a aplicar a cada chamada baseada nas mensagens "*INVITE*" e "BYE" e nos valores colocados na configuração das taxas.

5.2 Trabalho futuro

Este projecto apresentou bons resultados no campo das chamadas de voz sobre IP, contudo isto é apenas uma vertente das potencialidades que podem ser retiradas do protocolo SIP. Uma das características deste protocolo é a sua versatilidade, uma vez que a sua maior função é a sinalização da sessão multimédia. Como tal, pode ser usado para sinalização de outro tipo de fluxos de dados. Um bom exemplo é a videoconferência, que hoje em dia é bastante popular. Apesar disto, os equipamentos

não são muito comuns e a utilização do computador para este tipo de comunicação é o cenário mais habitual. As mensagens de texto também são uma boa ferramenta de comunicação e a sua integração no cliente é relativamente simples, uma vez que a biblioteca utilizada, *SIPekSDK* traz essa funcionalidade implementada. Outro aspecto que merece ser melhorado é a possibilidade de fazer chamadas em conferência, isto é, ter mais do que 2 clientes a comunicar entre si.

Do ponto de vista do servidor, o *proxy* em si tem um funcionamento bastante aceitável. O que se pode melhorar está ao nível dos serviços que podem vir a existir. É bastante comum haver serviços de *Voice Mail*, por exemplo, e o *Asterisk* tem provas dadas na integração destes serviços com sucesso [9]. A taxação pode ser melhorada uma vez que neste momento funciona com serviço pós-pago e, hoje em dia, é mais comum utilizar serviços pré-pagos. O *CDRTool* permite este serviço. Também tem uma limitação na medida em que o preço da chamada é calculada sob a condição de receber correctamente as mensagens de "*INVITE*" e "*BYE*", contudo caso estas sejam perdidas, o que ocorre é que a chamada já foi terminada mas não deixa de ser contabilizada. Para contornar este problema é possível melhorar o código de encaminhamento do *OpenSER* para que faça uma verificação do estado da chamada e sinalize o *FreeRADIUS* no caso de esta já ter terminado, ou então através da monitorização dos fluxos RTP para saber com exactidão quando estes são iniciados ou terminados.

Bibliografia

[1] Chandra P. and Lide D., Wi-Fi Telephony Challenges and Solutions For Voice Over WLANs, Newnes, 2007.

[2] http://paginas.fe.up.pt/~mrs01003/TCP_IP.htm

[3] Van Der Schaar M. and Chou A. P., Multimedia Over IP Wireless Networks Compression, Networking and Systems, Academic Press, 2007.

[4] Simionovich Nir, AsteriskNow, Packt Publishing, 2008.

[5] Van Meggelen J., Madsen L. and Smith J., Asterisk: The Future of Telephony second edition, O'Reilly, 2007.

[6] http://www.protocols.com/pbook/VoIPFamily.htm

[7] http://wiki.wireshark.org/UNISTIM

[8]http://www.fccn.pt/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=405&947cda2253a1dc58fe23dc95ac31cbed=3bd28d08a629e296aa06127db96698d0

[9] Gonçalves Flávio E., Building Telephony Systems Using Openser, Packt Publishing, 2008.

[10] http://www.voip-info.org/wiki-Open+Source+VOIP+Software

[11] www.pjsip.org

[12] http://voipengine.googlepages.com

[13] http://www.sermyadmin.org/index.php?view=article&catid=4%3Adoc&id=3%3Asermyadmin-installation-updated&option=com_content&Itemid=2

[14] http://tomcat.apache.org/download-60.cgi

[15] http://dev.mysql.com/downloads/connector/j/5.1.html

[16] Gonçalves Flávio E., Asteriskguide 1,2e 3 free chapters, www.asteriskguide.com.

[17] http://wiki.freeswitch.org/wiki/Users_Guide_Introduction

[18] http://www.mysipswitch.com/

[19] http://www.cisco.com/en/US/products/sw/voicesw/ps2157/index.html

[20] http://www.nec.com.au/purevoip/univergesv7000.htm

[21] http://www.avaya.com/gcm/master-usa/en-us/products/offers/communication_manager.htm

[22] http://products.nortel.com/go/product_content.jsp?parId=0&segId=0&prod_id=63420

[23] http://www.voip-info.org/wiki/view/VOIP+PBX+and+Servers

[24] Baset Salman A. and Schulzrinne H., An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Department of Computer Science, Columbia University, New York, 2004.

[25] http://progx.ch/home-voip-prixbetamax-3-1-1.html preços BETAMAX

[26] http://www.voipproviderslist.com/country/voip-portugal/voip-providers-portugal/

[27] http://www.vivaolinux.com.br/artigo/Integrando-Asterisk-ao-PABX-atraves-de-placa-de-fax modem?pagina=2

[28] http://www.gta.ufrj.br/~rezende/cursos/eel879/trabalhos/voip1/Real-TimeVsBest-Effort.html

Anexo A – Instalação de programas

1- Instalação *java*

A instalação da *Framework java* é necessária para a execução da interface gráfica *SerMyAdmin.* Para o fazer é necessário seguir as directivas descritas de seguida.

Como tal, é necessário editar o ficheiro de fontes onde o Ubuntu actualiza o repositório. Este ficheiro é o "/etc/apt/sources.list" e é necessário adicionar as palavras "contrib" e "non-free" à frente de cada entrada. Desta forma a aplicação de actualização de software vai procurar mais fontes. De seguida executa-se os seguintes comandos:

sudo apt-get update

sudo apt-get install sun-java5-jdk

sudo update-java-alternatives –s java-1.5.0-sun

2- Instalação do *Tomcat6*

Esta aplicação é necessária para que o servidor funcione como servidor HTTP, de forma à aplicação *SerMyAdmin* possa ser acedida graficamente.

Pode-se fazer o download desta aplicação em [14] e os comandos para a instalar são os seguintes:

tar –xzvf apache-tomcat-6.0.16.tar.gz

ln –s apache-tomcat-6.0.16 tomcat6

De seguida é necessário copiar o ficheiro que se encontra a seguir, para "/etc/init.d/tomcat6" de forma ao Tomcat correr quando o sistema operativo inicia. Os comandos para isto acontecer são os seguintes:

sudo chmod 755 /etc/init.d/tomcat6

sudo update-rc.d tomcat6 defaults 99

Depois deve ser instalado um driver MySQL para o Tomcat, para que o SerMyAdmin possa aceder à base de dados. Este pode ser encontrado em [15] e deve ser copiado para a directoria "/usr/local/tomcat6/lib/".

3- Ficheiros necessários para a configuração do *SerMyAdmin*

Na caixa seguinte está o conteúdo do ficheiro "context.xml".

```
<?xml version="1.0" encoding="UTF-8"?><Context path="/serMyAdmin"> <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver" maxActive="20" maxIdle="10" maxWait="-1" name="jdbc/openser_MySQL" type="javax.sql.DataSource" url="jdbc:mysql://localhost:3306/openser" username="administrador" password="user"/>
```

Na caixa seguinte está o conteúdo do ficheiro "openser.sql".

```
--
-- changes for table `domain`
--
ALTER TABLE `domain` ALTER `last_modified` SET DEFAULT  '2008-01-01 00:00:00';
UPDATE domain set `last_modified`='2008-01-01 00:00:00';
INSERT INTO `domain` (`id`, `domain`, `last_modified`, `version`) VALUES
(1, 'setup', '2008-01-01 00:00:00', 0);
---
--- changes for the table `jsec_user_role_rel`
---
INSERT INTO `jsec_user_role_rel` (`id`, `version`, `role_id`, `user_id`) VALUES
(1, 0, 3, 1);
---
--- changes in the subscriber table
---
```

```
UPDATE subscriber set `class`='user';
UPDATE subscriber set `role_id`='1';
UPDATE subscriber set `domain_id`='2';
--
--changes for table `subscriber`
--
INSERT INTO `subscriber` (`id`, `username`, `domain`, `password`, `first_name`, `last_name`, `email_address`, `datetime_created`, `ha1`, `ha1b`, `rpid`, `version`, `password_hash`, `auth_username`, `class`, `domain_id`, `role_id`) VALUES
(1, 'admin', 'setup', 'secret', 'Administrator', '', 'admin@setup.com', '2008-01-01 00:00:00', 'e2df32663fbff3bb2bd22ed7a4a35636', '7d9f741c07c122817e51c4fc817b15dd', NULL, 0, 'e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4', 'admin@setup', 'User', 1, 3);
--alias
ALTER TABLE alias ALTER version SET DEFAULT 0;
ALTER TABLE alias ALTER user_id SET DEFAULT 1;
--domain
ALTER TABLE domain ALTER version SET DEFAULT 0;
ALTER TABLE domain ALTER last_modified SET DEFAULT '2008-01-01 00:00:00';
--grp
ALTER TABLE grp ALTER version SET DEFAULT 0;
ALTER TABLE grp ALTER group_id SET DEFAULT 1;
ALTER TABLE grp ALTER last_modified SET DEFAULT '2008-01-01 00:00:00';
ALTER TABLE grp ALTER user_id SET DEFAULT 1;
--gw
ALTER TABLE gw ALTER version SET DEFAULT 0;
ALTER TABLE gw ALTER grp_id SET DEFAULT 1;
ALTER TABLE gw ALTER port SET DEFAULT 5060;
ALTER TABLE gw ALTER prefix SET DEFAULT NULL;
ALTER TABLE gw ALTER strip SET DEFAULT NULL;
ALTER TABLE gw ALTER transport SET DEFAULT 1;
ALTER TABLE gw ALTER uri_scheme SET DEFAULT 1;
--gw_grp
ALTER TABLE gw_grp ALTER version SET DEFAULT 0;
--jsec_permission
ALTER TABLE jsec_permission ALTER version SET DEFAULT 0;
--jsec_role
ALTER TABLE jsec_role ALTER version SET DEFAULT 0;
--jsec_role_permission_rel
ALTER TABLE jsec_role_permission_rel ALTER version SET DEFAULT 0;
ALTER TABLE jsec_role_permission_rel ALTER role_id SET DEFAULT 1;
--jsec_user_permission_rel
ALTER TABLE jsec_user_permission_rel ALTER version SET DEFAULT 0;
ALTER TABLE jsec_user_permission_rel ALTER user_id SET DEFAULT 1;
--jsec_user_role_rel
ALTER TABLE jsec_user_role_rel ALTER version SET DEFAULT 0;
ALTER TABLE jsec_user_role_rel ALTER role_id SET DEFAULT 1;
ALTER TABLE jsec_user_role_rel ALTER user_id SET DEFAULT 1;
--lcr
ALTER TABLE lcr ALTER version SET DEFAULT 0;
ALTER TABLE lcr ALTER grp_id SET DEFAULT 1;
--puorg
ALTER TABLE puorg ALTER version SET DEFAULT 0;
--register_user
ALTER TABLE register_user ALTER version SET DEFAULT 0;
ALTER TABLE register_user ALTER datetime_created SET DEFAULT '2008-01-01 00:00:00';
ALTER TABLE register_user ALTER domain_id SET DEFAULT 1;
ALTER TABLE register_user ALTER role_id SET DEFAULT 1;
--subscriber
```

```
ALTER TABLE subscriber ALTER version SET DEFAULT 0;
ALTER TABLE subscriber ALTER class SET DEFAULT 'User';
ALTER TABLE subscriber ALTER datetime_created SET DEFAULT '2008-01-01 00:00:00';
ALTER TABLE subscriber ALTER domain_id SET DEFAULT 1;
ALTER TABLE subscriber ALTER role_id SET DEFAULT 1;
--trusted
ALTER TABLE trusted ALTER version SET DEFAULT 0;
ALTER TABLE trusted ALTER proto SET DEFAULT 'any';
--usr_preferences
ALTER TABLE usr_preferences ALTER version SET DEFAULT 0;
ALTER TABLE usr_preferences ALTER last_modified SET DEFAULT '2008-01-01 00:00:00';
ALTER TABLE usr_preferences ALTER type SET DEFAULT 0;
ALTER TABLE usr_preferences ALTER user_id SET DEFAULT 1;
```

4- Instalação da aplicação MTA – Message Transfer Agent

A aplicação MTA é usada para que o programa *SerMyAdmin* possa enviar mensagens de correio electrónico para os utilizadores.

Deve-se usar os seguintes comandos para instalar a aplicação:

```
sudo apt-get install exim4
sudo dpkg-reconfigure exim4-config
```

Algumas instruções aparecem no ecrã, devendo ser devidamente seguidas. De seguida deve-se editar o ficheiro "/usr/local/etc/tomcat6/webapps/SerMyAdmin/WEB-INF/spring/resource.xml".

5- Instalação das dependências do *Asterisk*

Para a aplicação *Asterisk* funcione necessita da instalação de bibliotecas adicionais. De seguida serão apresentados as directivas para instalar o *Zaptel* e o *Libpri.*

```
sudo tar xzvf zaptel-1.4.x.tar.gz
cd /usr/src/zaptel-1.4.x/
make clean
./configure
make menuselect
make install
make install-udev
make config
update-rc.d zaptel defaults 99


sudo tar xzvf libpri-1.4.x.tar.gz
cd /usr/src/libpri-1.4.x/
make clean
make
make install
```

6- Aplicar um *patch* ao *FreeRadius* para corrigir erros conhecidos

Primeiro fazer o *download*:

```
wget http://download.dns-hosting.info/CDRTool/freeradius/freeradius.patch
```

De seguida aplicar:

```
sudo apt-get build-dep freeradius
sudo apt-get source freeradius
cd freeradius-1.1.6
sudo patch –p1 –s < ../freeradius.patch
sudo debuild
cd ..
```

```
sudo dpkg –I freeradius*.deb
```

7- Instalação *php* e dependências.

Ao usar o comando "*sudo apt-get –f install*" na instalação do *CDRTool*, ele já instalou o php e as dependências, contudo falta a sua configuração. Nas directorias dentro de "*/etc/php5/*" existe um ficheiro "php.ini". É necessário colocar em todos o comando "*extension=mysql.so*" para activar o módulo "*php-mysql*".

De seguida é necessário activar o php no servidor "*Apache*". Para tal edita-se o ficheiro "*/etc/apache/httpd.conf*" e coloca-se o texto:

```
DirectoryIndex index.phtml index.php index.html index.htm index.shtml index.cgi
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
```

Para ter a certeza que o módulo "php" está activado na configuração apache, a seguinte linha deverá estar no ficheiro *"/etc/apache/modules.conf*". Também se deve alterar o valor de "*AllowOverride*"

```
LoadModule php4_module /usr/lib/apache/1.3/libphp4.so
(…)
AllowOverride All
```

Depois deve-ser criar um *host* virtual para o *CDRTool*, tal como no seguinte exemplo:

```
sudo cp setup/apache2/sites-available/cdrtool.example.com /etc/apache2/sites-enabled
sudo cp setup/apache2/conf.d/cdrtool /etc/apache2/conf.d/
```

O conteúdo do ficheiro deverá ser como o seguinte:

```
<VirtualHost cdrtool.example.com:80>
  ServerName        cdrtool.example.com
  DocumentRoot      /var/www/
  CustomLog         /var/log/apache2/cdrtool-access.log combined
  ErrorLog          /var/log/apache2/cdrtool-errors.log
  SetEnvIf User-Agent ".*MSIE.*"     nokeepalive ssl-unclean-shutdown
  # To enable SSL:
  # a2enmode ssl
  # add Listen 443 to ports.conf file
  # generate site certificates
  # SSLEngine         On
  # SSLCertificateFile    /etc/apache2/ssl/snakeoil-rsa.crt
  # SSLCertificateKeyFile /etc/apache2/ssl/snakeoil-rsa.key
</VirtualHost>
```

Agora é necessário activar o site virtual usando o comando:

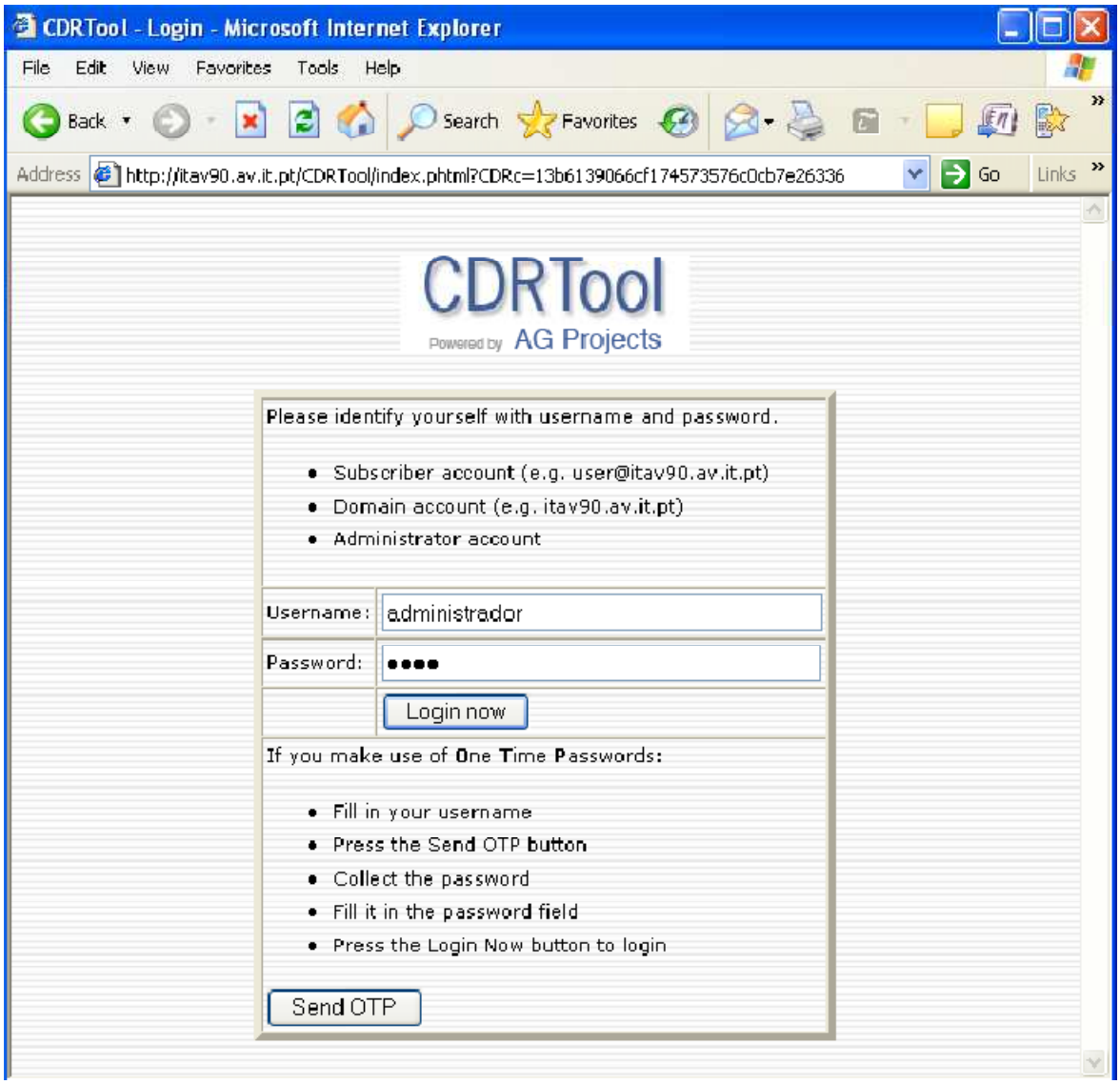
```
a2ensite cdrtool.example.com
```

Anexo B – Configuração de programas

1- Configuração do CDRTool

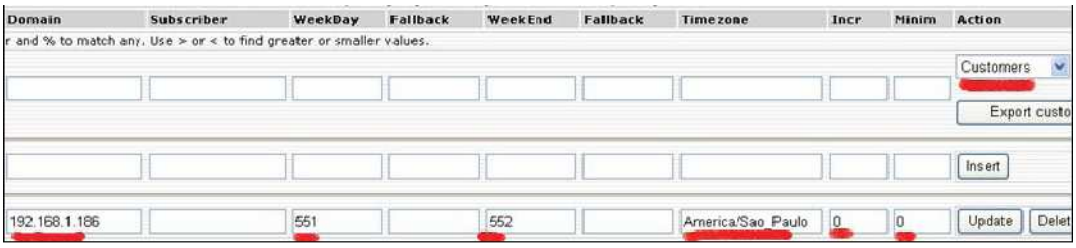
Como foi referido na secção 3.2.5.3, terá que ser configurado o plano de taxas (*Rating Plan*) para que as chamadas sejam correctamente taxadas.

O primeiro passo será aceder ao site "http://itav90.av.it.pt/CDRTool" e fazer a autenticação no sistema.



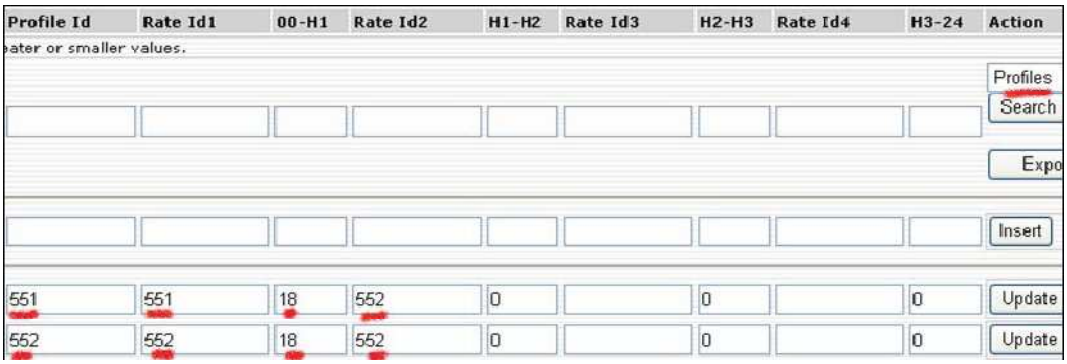
B. 1 – Página de entrada do CDRTool.

Após a autenticação na página da Figura B. 1, irá aparecer um ecrã onde é possível consultar a base de dados. O passo seguinte será carregar na ligação "*Rating*". Existe um menu do lado direito com o texto "*Destinations"*. Terá que se aceder ao campo "*Costumer"*, e criar um cliente para o sistema, como se pode verificar na Figura B. 2.



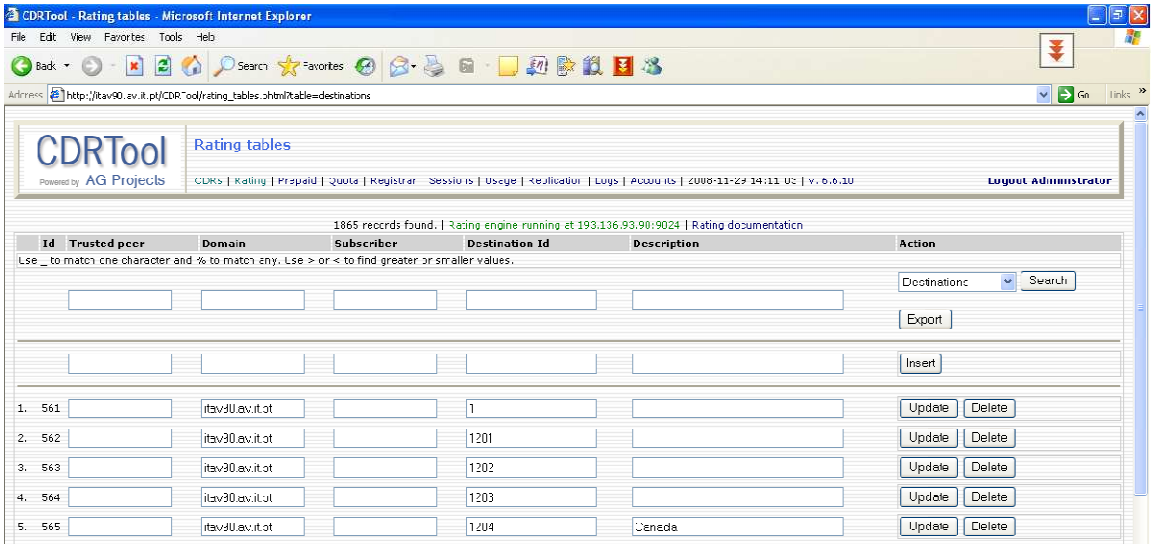
B. 2 - Menu "Costumers" no CDRTool [9]

Nos campos "*Weekdays*" e "*Weekend*" são colocados valores numéricos que vão corresponder a perfis que são configurados acedendo novamente ao menu e carregando em "*profiles"*. Deve-se colocar o mesmo valor e nesta página é possível atribuir um perfil a um diferente período horário, de forma que a taxa a aplicar seja diferente consoante a altura do dia. A Figura B. 3 mostra isso mesmo.



B. 3 - Menu "Profiles" no CDRTool [9].

De seguida deve-se aceder ao menu novamente e aceder a "*Rates"*. É aqui que se define o preço da chamada consoante o destino e aplica-se um dos perfis criados anteriormente. Por fim acede-se novamente à página "*Destinations"* e pode ser definido um nome a cada destino para que seja mais fácil de identificar, como por exemplo definir "351" como "Portugal".



B. 4 - Página "Destination" no CDRTool

Na Figura B. 4 está patente a página onde são definidos os destinos. A secção acedida é a "*Rating*".

Existe uma forma mais simples de configurar estas secções. Com a aplicação existem uns ficheiros de exemplo que podem ser carregados para a base de dados de uma forma automática, ficando o sistema totalmente configurado. Para o fazer deve-se aceder ao directório "/var/www/CDRTool/setup/csv" e copiar os ficheiros para "/var/spool/cdrtool". De seguida executa-se o comando "./var/www/CDRTool/setup/script/importRatingTables.php" e os ficheiros são carregados. Os valores que se podem ver na Figura B. 1 foram introduzidos desta forma, bem como os dos campos atrás referidos.

2- Configuração do *Asterisk* para utilizar o *VoipBuster* como provedor

Sip.Conf

```
[general]
context=sipincoming
bindport=5061
register=>itav90:user@sip.voipbuster.com
[openser]
type=peer
host=127.0.0.1
[sipproxy]
type=peer
host=sip.voipbuster.com
secret=user
username=itav90
fromuser=itav90
port=5060
canreinvite=no
insecure=very
qualify=no
```

Extensions.conf

```
[general]
[globals]
[sipincoming]
exten=>_[0-9].,1,Dial(SIP/${EXTEN}@sipproxy)
exten=>_[0-9].,2,Hangup()
[sipoutgoing]
exten=_[0-9].,1,Answer()
exten=_[0-9].,2,Dial(SIP/user1@openser); por testar
exten=_[0-9].,3,Hangup()
```

3- Ficheiro de configuração do CDRTool: global.inc

```
# System and web paths
$CDRTool['tld']    = "/CDRTool";
$CDRTool['Path']   = "/var/www/CDRTool";
$_PHPLIB['libdir'] = $CDRTool['Path']. "/phplib/";
include($_PHPLIB["libdir"] . "prepend.php3");
# PHP Error reporting
$errorReporting = (E_ALL & ~E_NOTICE);
$errorReporting = 1;    // comment this out to enable PHP warnings
error_reporting($errorReporting);
# Service provider information
$CDRTool['provider']['name']                = "itav90.av.it.pt";
$CDRTool['provider']['service']             = "SIP service";
$CDRTool['provider']['timezone']            = "Europe/Lisbon";
$CDRTool['provider']['fromEmail']           = "tres@av.it.pt";
$CDRTool['provider']['toEmail']             = "tres@av.it.pt";
$CDRTool['provider']['sampleLoginSubscriber'] = "user@itav90.av.it.pt";
$CDRTool['provider']['sampleLoginDomain']   = "itav90.av.it.pt";
# Rating engine settings
$RatingEngine=array("socketIP"              => "127.0.0.1",
                    "socketPort"            => "9024",
                    "CDRS_class"            => "ser_radius",
                    "prepaid_lock"          => true,
                    "log_delay"             => 0.05,
                    "split_rating_table"    => false,
                    "csv_delimiter"         => ",",
                    "priceDenominator"      => 10000, // e.g. 1 Eur = 10000 units
                    "priceDecimalDigits"    => 4,     // how many digits to round the prices to
                    "minimumDurationCharged" => 0,    // Only calls greater than this duration will be charged
                    "durationPeriodRated"   => 60,    // the prices from the rating table are calculated per this
period
                    "trafficSizeRated"      => 1024,  // same as above but for data traffic
                    "reportMissingRates"    => 0,     // send email notifications if rates are missing from the
ratingEngine
                    "minimumDuration"       => 0,     // minimum duration to rate, if call duration is shorter the
price is zero
                    "allow"                 => array ('10.'),  // list with local network clients allowed to connect
                        "MaxSessionTime"    => 36000  // limit all prepaid calls to maximum 10 hours
                    );
# Normalize engine settings
$CDRTool['normalize']['defaultCountryCode']    = "351";

# Anti-fraud settings
# create group quota in OpenSER and deny calls to users in this group
# $UserQuota["default"]["traffic"]  = 5000;          // MBytes
```

```
$UserQuota["default"]["cost"]     = 1000;             // Euro
# CDRTool datasources
class DB_CDRTool extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "cdrtool";
  var $User     = "administrador";
  var $Password = "user";
  var $Halt_On_Error ="no";
}
class DB_Locker extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "cdrtool";
  var $User     = "locker";
  var $Password = "user";
  var $Halt_On_Error ="no";
}
class DB_radius extends DB_Sql {
  var $Host     = "locahost";
  var $Database = "radius";
  var $User     = "radius";
  var $Password = "senha";
  var $Halt_On_Error ="no";
}
class DB_subscribers extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "openser";
  var $User     = "openser";
  var $Password = "openserrw";
  var $Halt_On_Error ="no";
}
class DB_openser extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "openser";
  var $User     = "openser";
  var $Password = "openserrw";
  var $Halt_On_Error ="yes";
}
class DB_siptrace extends DB_Sql {
  var $Host     = "locahost";
  var $Database = "sip_trace";
  var $User     = "openser";
  var $Password = "openserrw";
  var $Halt_On_Error ="yes";
}
$replicated_databases=array('cluster1'=>array(
                                "db1"=>array('ip'      =>'127.0.0.1',
                                             'slave_of'=>'db-log2',
                                             'user'    =>'process',
                                             'password'=>'password',
                                             'replication_user'    =>'replication',
                                             'replication_password'=>'password',
                                             'active_master' => true
                                             ),
                                "db2"=>array('ip'      =>'127.0.0.1',
                                             'slave_of'=>'db-log1',
                                             'user'    =>'process',
                                             'password'=>'password',
```

```
                                        'replication_user'    =>'replication',
                                        'replication_password'=>'password'
                    )                    )
            );

class DomainAuthLocal extends DomainAuth {      // defined in phplib/local.inc
}
class PageLayoutLocal extends PageLayout {      // defined in phplib/local.inc
}
# To customize E164 normalization
class E164_custom extends E164 {      // defined in library/cdr_lib.phtml
}
# $CDRTool['normalize']['E164Class']          = "E164_custom";
$DATASOURCES=array(
"unknown"=>array(
                "class"             => "CDRS_unknown" // leave it here
                ),
"ser_radius"=>array(
                "name"              => "OpenSER",
                "class"             => "CDRS_ser_radius",
                "table"             => "radacct".date("Ym"),
                "db_class"          => array("DB_radius2","DB_radius"),
                "rating"            => "1",
                "normalizedField"    => "Normalized",
                "UserQuotaClass"     => "OpenSERQuota",
                "AccountsDBClass"    => "DB_openser",
                "UserQuotaNotify"    => "1",
                "purgeCDRsAfter"     => 120, // how many days to keep the CDRs
                "SIPProxies"         => array("127.0.0.1" => "node01"
                                        ),
                "db_class_siponline" => "DB_openser",
                 ),
"SIP"=>array("WEBName"=>"OpenSER"),
"sip_trace"   =>array(
                "name"              => "SIP trace",
                "db_class"          => "DB_siptrace",
                "table"             => "sip_trace",
                "enableThor"        => false,
                "purgeRecordsAfter"  => "7",
                "invisible"         => 1
                )
);
// load CDRTool libraries
$CDRToolModules=array("openser");
if ($_SERVER['REMOTE_ADDR']=="127.0.0.1") {
   //$verbose=1;
} else {
   // prevent set of verbose via post/get
   unset($verbose);  } ?>
```

Anexo C – Configuração de equipamento

1- Configuração do *Asterisk* para utilizar o *VoipBuster* como provedor

O driver do *asterisk* que controla o *hardware* é o *zaptel*. Para configuramos o *asterisk* para que reconheça a placa de fax modem, devemos editar o arquivo zaptel.conf e colocar no final do ficheiro as seguintes linhas:

```
fxsks=1
# Aqui especificamos o número de portas fxo, no caso 1. Notem que ao invés de usarmos fxoks=1 para placa fxo, usamos o contrário fxsks=1. Se tivéssemos uma placa fxs esse parâmetro ficaria fxoks=1

loadzone=us
defaultzone=us
# Aqui definimos a zona como US.
```

Configurado o *zaptel*, vamos especificar o funcionamento da placa no arquivo "*zapata.conf*". Cada interface FXO dentro do *Asterisk* deve ser configurada no arquivo "*zapata.conf*". É dentro deste que definimos o contexto de ligações, o nível de ruído, o volume das ligações. As seguintes linhas devem ser adicionadas no final do "*zapata.conf*"

```
# vi /etc/asterisk/zapata.conf
signalling=fxs_ks
# tipo de interface, lembrando se estamos usando uma fxo, aqui vai fxs_ks
language=en
rxgain=0.0
txgain=0.0
# podemos aumentar ou diminuir o parâmetro do rx e do tx para resolver problema de eco. Esse parâmetro vai de -8.0 até 8.0
echocancelwhenbridged=yes
echocancel=yes
context=581
# aqui especificamos o contexto que vai entrar as regras de marcação
channel => 1
# número do canal. Se tivesses por exemplo uma tdm400 com 4 fxo, para cada interface fxo eu deveria repetir as linhas acima, mudando apenas o channel. Por exemplo a fxo 1 ficaria channel =>1, a fxo 2 ficaria channel => 2.
```

Para que a FXO fique a funcionar é necessário activar com os seguintes comandos:

```
modprobe zaptel
```

(carrega o driver zaptel)

```
modprobe wcfxo
```

(carrega o driver da fxo)

```
ztcfg -vv
```

(configura a placa, deve aparecer a seguinte mensagem no ecrã: "Channel 01: FXS Kewlstart (Default) (Slaves: 01)")

Agora falta configurar os ficheiros "sip.conf" e "extensions.conf" para definir entradas e saídas [9]. O directório é "/etc/asterisk".

SIP.conf:

```
[general]
context=sipincoming
[sipproxy]
Type=peer
host=127.0.0.1
```

Extensions.conf

```
[general]
[globals]
[sipincoming]
exten=>_[0-9].,1,Dial(Zap/g1/${EXTEN:1})
exten=>_[0-9].,2,Hangup()
[sipoutgoing]
exten=s,1,Answer()
exten=s,2,Dial(SIP/${EXTEN}@sipproxy)
exten=s,3,Hangup()
```

2- Configuração do *Cisco 2061 Gateway*

```
voice class codec 1
codec preference 2 g711ulaw
interface Ethernet0/0
ip address 10.1.30.38 255.255.0.0
half-duplex
ip classless
ip route 0.0.0.0 0.0.0.0 10.1.0.1
no ip http server
ip pim bidir-enable
voice-port 1/0
voice-port 1/1
mgcp profile default
! The dial-peer pots commands will handle the calls coming from SIP !dial-peers. Any call matching 9 followed by any number of digits will be !forwarded to the PSTN with the 9 striped.
dial-peer voice 1 pots
destination-pattern 9T
port 1/0
dial-peer voice 2 pots
destination-pattern 9T
port 1/1
!The dial-peer voip commands will handle the calls coming from the pots !dial peers (PSTN). You can prefix a number (80 in this example) and send the DID number ahead.
dial-peer voice 123 voip
destination-pattern ....T
prefix 80
forward all
session protocol sipv2
session target ipv4:10.1.30.22
dtmf-relay sip-notify
```

Anexo D – Bases de dados

1- Bases de dados existentes

Usando o commando “mysql –u root –p –Bse “show databases”” é possível exportar para um ficheiro o nome das bases de dados existentes no servidor MySQL. A saída é a seguinte:

```
information_schema
cdrtool
mysql
openser
radius
```

2- Tabelas dentro das bases de dados

Usando o comando “mysql –u root –p NOME_da_BD –Bse “show tables”” é possível exportar para um ficheiro a lista de tabelas dentro de cada uma das bases de dados, apenas substituindo o campo “NOME_DA_BD” por um dos nomes vistos na secção anterior.

Em primeiro lugar, a tabela “*openser”.*

```
acc
active_watchers
address
alias
aliases
cpl
dbaliases
domain
domainpolicy
grp
gw
gw_grp
imc_members
imc_rooms
jsec_permission
jsec_role
jsec_role_permission_rel
jsec_user_permission_rel
jsec_user_role_rel
lcr
location
missed_calls
pdt
presentity
pua
puorg
re_grp
register_user
silo
```

sip_trace
speed_dial
subscriber
trusted
uri
usr_preferences
version
watchers
xcap_xml

Agora, a tabela "*radius*".

nas
radacct
radcheck
radgroupcheck
radgroupreply
radippool
radpostauth
radreply
usergroup

Por último, a tabela "*cdrtool*".

active_sessions
active_sessions_split
asterisk_cdr
auth_user
billing_customers
billing_enum_tlds
billing_holidays
billing_profiles
billing_profilesNGN
billing_rates
billing_ratesNGN
billing_rates_history
db_sequence
destinations
isdncause
lastquery
log
memcache
normalize_lock
prepaid
prepaid_cards
prepaid_history
quota_usage
settings
sip_status
spam