Ricardo J. O. Tching de M. Coelho   Confiabilidade e Escalabilidade de Sistemas de Voz sobre IP

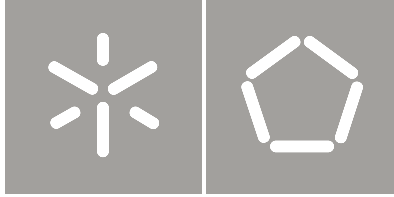
UMinho | 2010

Universidade do Minho
Escola de Engenharia

Ricardo Jorge Oliveira Tching de Magalhães Coelho

# Confiabilidade e Escalabilidade de Sistemas de Voz sobre IP

Outubro de 2010



Universidade do Minho
Escola de Engenharia

Ricardo Jorge Oliveira Tching de Magalhães Coelho

# Confiabilidade e Escalabilidade de Sistemas de Voz sobre IP

Tese de Mestrado
Engenharia Informática

Trabalho efectuado sob a orientação do
Professor Doutor Rui Carlos Oliveira

Outubro de 2010

# Agradecimentos

Ao apresentar esta dissertação quero agradecer a todos aqueles que, de alguma forma, contribuíram para a sua concretização particularmente:

- Ao meu orientador, o Professor Doutor Rui Oliveira pela orientação, acompanhamento e incentivo que me permitiram levar a cabo esta dissertação;

- Ao Engenheiro José Ferreira da Wavecom S.A. pela oportunidade de trabalhar num projecto que serviu de base a todo o trabalho desenvolvido nesta dissertação;

- Aos meus colegas do Laboratório de Sistemas Distribuídos pela discussão de ideias, entreajuda e apoio "moral";

- Aos meus pais pelo apoio e suporte fundamental ao longo de todo o meu percurso;

- À Rute, por toda a ajuda e apoio

# Resumo

## Confiabilidade e Escalabilidade de Sistemas de Voz sobre IP

No mercado das telecomunicações tornam-se cada vez mais evidentes as consequências da evolução tecnológica. Neste contexto, os sistemas tradicionais de telefonia começam a ficar cada vez mais obsoletos, dando azo à criação de novos sistemas de telefonia, mais vantajosos quer sob o ponto de vista económico quer funcional.

O surgimento e a evolução da Internet vieram provocar uma revolução tecnológica no panorama das telecomunicações. É neste contexto que surgem os sistemas de telefonia de voz sobre IP que, para além de possuírem funcionalidades não disponíveis nos sistemas tradicionais de telefonia, reduzem significativamente os custos das telecomunicações.

No entanto, e uma vez que os sistemas tradicionais de telefonia são conhecidos pela sua fiabilidade, os sistemas de voz sobre IP devem pois assegurar um alto nível de confiabilidade e escalabilidade, de modo a firmar a sua posição no mercado das telecomunicações. Para alcançar tudo isto, os sistemas de voz sobre IP devem não só apresentar resiliência ao nível dos servidores que prestam o serviço, como também devem ter a capacidade de suportar um elevado número de chamadas em simultâneo. Tais características asseguram que o utilizador seja capaz de efectuar chamadas, mesmo que ocorra uma interrupção das mesmas devido à falha de um dos servidores. Para além de ser garantida a confiabilidade do sistema, é também garantida simultaneamente a não ocorrência de congestionamento do sistema o que poderia, por sua vez, induzir quebras na prestação do serviço.

# Abstract


## Reliability and Scalability of Voice Over IP Systems

The telecommunications market is becoming increasingly aware of the consequences of technological evolution. As a result, traditional telephony systems are becoming obsolete giving rise to the creation of new telephone systems, more advantageous in terms of economy and functionality.

The widespread of Internet brought along a technological revolution into the telecommunications sector. This revolution gave birth to a new telephony based system, the Voice over IP, which not only is more cost efficient but also provides new and improved functionalities, thus becoming the natural successor of traditional telephony systems.

Since traditional telephony systems are known for their reliability, voice over IP systems must ensure a high level of reliability and scalability in order to assert its position in the telecommunications market. To achieve this voice over IP systems should, not only provide resilience at the service providing server, but also be capable of withstanding a large number of simultaneous calls. These features ensure that the user is able to perform phone-calls even if they abruptly end, due to failure of one of the providing servers. Besides guaranteeing the reliability of the system, it is also reassured that no congestion of the system will occur, thus, not allowing breaks in service delivery.

# Conteúdo


**1 Introdução** **1**

1.1 Objectivos do Projecto . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Estrutura da Dissertação . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Estado da Arte** **5**

2.1 História do SIP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.1.1 Session Invitation Protocol (SIPv1) . . . . . . . . . . . . . . . 5

2.1.2 Simple Conference Invitation Protocol (SCIP) . . . . . . . . . 6

2.1.3 Session Initiation Protocol (SIP) . . . . . . . . . . . . . . . . 6

2.2 Funcionalidades do SIP . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.3 Entidades SIP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3.1 User Agents . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3.2 Redirect Servers . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3.3 Proxy Servers . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.3.4 Registrars . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.3.5 Location Servers . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.4 Diferenças que distinguem o SIP de outros Protocolos . . . . . . . . . 11

2.4.1 O SIP no Toolkit da IETF . . . . . . . . . . . . . . . . . . . 11

2.4.2 Estabelecer/Descrever uma Sessão . . . . . . . . . . . . . . . . 12

2.4.3 Periferia Inteligente . . . . . . . . . . . . . . . . . . . . . . . 12

2.4.4 Reutilização de componentes . . . . . . . . . . . . . . . . . . . 12

2.5 Arquitectura Base de Sistemas de Telefonia baseados em SIP . . . . . 14

2.6 Aumento da confiabilidade . . . . . . . . . . . . . . . . . . . . . . . . 15

2.6.1 Baseado no cliente . . . . . . . . . . . . . . . . . . . . . . . . 15

2.6.2 Baseado em DNS . . . . . . . . . . . . . . . . . . . . . . . . . 16

2.6.3 Replicação de base de dados . . . . . . . . . . . . . . . . . . . 16

2.6.4 Assumir o endereço IP . . . . . . . . . . . . . . . . . . . . . . 18

2.6.5 Pooling de servidores . . . . . . . . . . . . . . . . . . . . . . . 18

2.6.6 Infrastutura de SIP usando *Content Addressable Networks* . . 19
2.7 Aumento de escalabilidade . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.7.1 Baseada em DNS . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.7.2 Baseado em Identificadores . . . . . . . . . . . . . . . . . . . . 20
2.7.3 *Network Address Translation (NAT)* . . . . . . . . . . . . . . 21
2.7.4 Múltiplos servidores com o mesmo endereço IP . . . . . . . . . 21
2.7.5 Arquitectura de duas fases . . . . . . . . . . . . . . . . . . . . 22
2.7.6 Optimização de uma infra-estrutura SIP usando *Content Addressable Networks* . . . . . . . . . . . . . . . . . . . . . . . . 22

**3 Solução VoIP da Wavecom S.A. 25**
3.1 Ferramentas de Software do iPBX . . . . . . . . . . . . . . . . . . . . 26
3.1.1 FreeRADIUS . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.1.2 OpenSER . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.1.3 Asterisk . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.1.4 MySQL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.1.5 FreePBX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.2 Tolerância a faltas na solução da Wavecom S.A. . . . . . . . . . . . . 28

**4 Arquitectura de *Failover* da Solução 31**
4.1 Transparência do sistema . . . . . . . . . . . . . . . . . . . . . . . . . 32
4.2 Problema de replicação do OpenSER e uma solução . . . . . . . . . . 33
4.3 Problema de replicação do Asterisk e uma solução . . . . . . . . . . . 38
4.3.1 Ficheiros e Directórios do Asterisk . . . . . . . . . . . . . . . . 39
4.3.2 Base de dados . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.3.3 Memória *cache* . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.3.4 Replicação de ficheiros do Asterisk . . . . . . . . . . . . . . . . 44
4.3.5 Replicação da base de dados . . . . . . . . . . . . . . . . . . . 49
4.3.6 Replicação de informação armazenada na memória *cache* . . . 49
4.4 Resumo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

**5 Arquitectura Hierarquizável da Solução 57**
5.1 Arquitectura de Hierarquização . . . . . . . . . . . . . . . . . . . . . 57
5.1.1 Hierarquia no OpenSER . . . . . . . . . . . . . . . . . . . . . . 61
5.1.2 Hierarquia no Asterisk . . . . . . . . . . . . . . . . . . . . . . . 62
5.2 Resumo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

**6 Conclusão** **65**

6.1 Conclusão e Trabalho Futuro . . . . . . . . . . . . . . . . . . . . . . . . 65

**Bibliografia** **70**

**A Módulo Criado** **71**

**B Scripts** **85**

# Lista de Figuras

2.1 Funcionamento de um Redirect Server . . . 9
2.2 Funcionamento de um Proxy Server . . . 10
2.3 Pedido de Registo . . . 11
2.4 Distribuição de Servidores . . . 13
2.5 Estabelecimento básico de uma sessão SIP . . . 15
2.6 Failover baseado em DSN . . . 17
2.7 Failover baseado em replicação de bases de dados . . . 17
2.8 Endereço de IP apoderado por parte do servidor secundário . . . 18
2.9 Pooling de servidores . . . 19
2.10 Distribuição de carga baseada em identificadores . . . 21
2.11 Arquitectura escalável e confiável baseado em duas fases . . . 23

3.1 Solução de VoIP da Wavecom S.A. . . . 26
3.2 Funcionamento do DRBD . . . 30

4.1 Funcionamento do LVS . . . 33
4.2 Replicação Síncrona . . . 36
4.3 Não há perda de informação quando um nodo falha . . . 36
4.4 Replicação Assíncrona . . . 37
4.5 Há perda de informação quando um nodo falha . . . 37
4.6 Problema de *Split Brain* . . . 37
4.7 Solução para o problema de *Split Brain* . . . 38
4.8 Directórios do Asterisk . . . 41
4.9 Directórios criados no sistema de ficheiros distribuído . . . 46
4.10 Script que detecta alterações em ficheiros e efectua recarregamento do Asterisk . . . 48
4.11 Pedido e processamento do dicionário . . . 54
4.12 Processamento do dicionário e do Buffer . . . 54

4.13 Solução Final do Failover do iPBX . . . . . . . . . . . . . . . . . . . . 55

5.1 Hierarquia do Sistema de VoIP da Wavecom . . . . . . . . . . . . . . 58
5.2 Registo dos telefones IP de acordo com o seu nível na hierarquia . . . 59
5.3 Exemplos de chamadas efectuadas entre níveis diferentes . . . . . . . 60
5.4 Exemplos de chamadas efectuadas para o exterior . . . . . . . . . . . 61
5.5 Níveis da Arquitectura Hierarquizável . . . . . . . . . . . . . . . . . . . 64

# Capítulo 1

# Introdução

Os sistemas de telefonia tradicionais estão a ficar cada vez mais ultrapassados uma vez que as necessidades de novas funcionalidades aumentaram. Ser capaz de reduzir os custos económicos das telecomunicações é crucial para os serviços de telefonia. Com a proliferação da Internet surgiram os serviços de telefonia na Internet com o objectivo de substituir os sistemas tradicionais de telefonia. É neste contexto que os sistemas de voz sobre IP (VoIP) prometem disponibilizar novas funcionalidades e serem mais competitivos.

Existem grandes diferenças entre os sistemas de telefonia tradicionais e os sistemas de VoIP. Numa chamada feita através dos sistemas tradicionais, os dados de voz passam por linhas telefónicas proprietárias, geridas por uma companhia telefónica e interligadas com linhas de outras companhias. Nestes sistemas são utilizadas linhas telefónicas dedicadas e, funcionalidades como chamada em espera ou identificador de chamadas, estão disponíveis com um custo extra. Estes podem ser actualizados ou expandidos recorrendo a novos equipamentos e a um aprovisionamento de linhas. Ao contrário dos sistemas tradicionais, os sistemas VoIP convertem voz em dados e enviam os mesmo em forma de pacotes via Internet. Para além disso, estes sistemas disponibilizam novas funcionalidades recorrendo praticamente a software. Relativamente a expansões nestes sistemas, é apenas necessária uma maior largura de banda e actualizações de software.

Os sistemas de VoIP nunca serão vistos como uma alternativa plausível aos sistemas tradicionais de telefonia se estes não assegurarem um correcto fornecimento do serviço, mesmo na presença de um falta. A perda da capacidade de fazer chamadas pode ser bastante nefasto para aqueles que dependem de tal serviço. Uma empresa de telemarketing, por exemplo, poderá sofrer perdas consideráveis se o sistema telefónico falhar.

Para evitar tais problemas, podem-se criar sistemas tolerantes a faltas replicando todos os componentes essenciais do sistema. Quando um componente falha, uma réplica assume o controlo garantindo a correcta prestação do serviço. Neste sentido, a implementação de sistemas tolerantes a faltas tornar-se-á crucial para garantir que os sistemas de VoIP se imponham no mercado e sejam capazes de substituir os sistemas tradicionais de telefonia.

Para além da criação de sistemas tolerantes a faltas, os sistemas de VoIP devem suportar um elevado número de chamadas em simultâneo. Se um sistema apenas consegue lidar com um número baixo de chamadas em simultâneo, então este pode facilmente ficar congestionado, impossibilitando a realização de uma nova chamada.

Muitas técnicas e arquitecturas podem ser aplicadas para melhorar a confiabilidade e escalabilidade de um sistema. No entanto, primeiro é necessário perceber e compreender a arquitectura do sistema alvo para saber quais as técnicas ou arquitecturas que possam ser implementadas. Sem este estudo inicial, não é possível alcançar a confiabilidade e escalabilidade que satisfaça as necessidades dos consumidores.

## 1.1 Objectivos do Projecto

Este projecto tem por base uma solução de VoIP cuja arquitectura compreende quatro componentes. Um destes componentes desenrola um papel crucial na solução sendo, por conseguinte, de extrema importância a sua análise para este projecto. De modo a tolerar falhas, este componente compreende duas máquinas: uma destas encontra-se em modo activo e a outra em modo passivo. Sempre que a máquina activa falha, a passiva assume o seu papel e disponibiliza o serviço de um modo correcto. Apesar de esta solução satisfazer o seu principal objectivo, esta arquitectura desperdiça uma recurso bastante valioso dado que uma máquina apenas entra em funcionamento quando ocorre uma falha na máquina activa.

Um dos objectivos deste projecto é o de melhorar a solução já existente de modo a que o sistema faça uso de todos os recursos disponíveis, criando assim uma solução na qual ambas as máquinas funcionem num modo activo fornecendo simultaneamente, garantias de confiabilidade. Assim, mesmo na ocorrência de uma falha por parte de uma das máquinas, o serviço de VoIP é assegurado de um modo correcto, causando a menor interferência possível aos utilizadores do sistema. Mesmo que a chamada seja terminada abruptamente quando uma máquina falha, o objectivo é garantir que quando o utilizar voltar a fazer a chamada, esta será feita sem qualquer percalço.

Para além da criação de uma solução que funcione no modo activo/activo,

pretende-se igualmente aumentar a escalabilidade do sistema. Este objectivo não deve ser alcançado através de optimizações dos componentes de software, mas sim apresentando uma arquitectura que permita ao sistema lidar e processar um elevado número de chamadas em simultâneo, sem perda significativa de performance. Esta arquitectura deverá fazer uso de novas máquinas como *cache*, de modo a evitar a sobrecarga do sistema.

## 1.2 Estrutura da Dissertação

O Capítulo 2 apresenta um breve resumo sobre o protocolo mais utilizado em sistemas de telefonia VoIP, bem como o seu funcionamento e características que o distinguem dos muitos outros protocolos criados para o mesmo efeito. Este capítulo descreve ainda algum do trabalho relacionado com os objectivos que se pretendem alcançar com este projecto. Esta descrição inclui algumas técnicas e arquitecturas, que podem ser implementadas com o objectivo de melhorar a escalabilidade e confiabilidade de um sistema. O Capítulo 3 descreve os detalhes e especificações da solução que serve de ponto de partida para o desenvolvimento deste projecto. Neste capítulo são descritos os componentes que constituem a solução. É também descrito o processo de tolerância a faltas já existente no sistema e o seu funcionamento no modo activo/passivo.

No Capítulo 4 é descrita e explicada a solução para o problema da confiabilidade do sistema. É realizada uma análise a todos os componentes de software, de modo a que todos os detalhes sejam considerados aquando a criação da solução. Após esta análise, é descrita a replicação de cada componente de software bem como as implicações que esta replicação entre duas máquinas no modo activo trazem para a solução. Neste capítulo é ainda descrito como se aumentou a transparência do sistema relativamente aos telefones IP.

O Capítulo 5 descreve a arquitectura proposta para resolver o problema da escalabilidade do sistema. Tal como no Capítulo 4, primeiro é realizada uma análise ao impacto que esta arquitectura poderá exercer sobre os componentes de software. Após esta análise, é explicada ao pormenor a solução considerando todos os detalhes dos componentes de software.

Por último, o Capítulo 6 sumariza as conclusões alcançadas com este projecto, sugerindo a prossecução deste trabalho num futuro próximo.

# Capítulo 2

# Estado da Arte

Este capítulo aborda a informação essencial acerca do *Session Initiation Protocol* (SIP), fazendo um resumo sucinto da bibliografia disponível no que a confiabilidade e escalabilidade dos sistemas de VoIP concerne.

## 2.1 História do SIP

### 2.1.1 Session Invitation Protocol (SIPv1)

Em 1996, Mark Handley e Eve Schooler desenharam e criaram o protocolo *Session Invitation Protocol*. Este protocolo possibilitava o convite de utilizadores para participarem em sessões de multimédia, permitindo assim a mobilidade do utilizador através do redireccionamento de convites para a localização actual do utilizador [12]. O SIPv1 utilizava o protocolo *Session Description Protocol* (SDP) [11] para descrever as sessões, e o protocolo *User Datagram Protocol* (UDP) para transporte.

Na altura em que surgiu, o conceito de registo em servidores de endereços de conferências foi proeminente. A partir do momento que um utilizador registasse a sua localização, um servidor de endereços seria capaz de redireccionar os pedidos de convite para o utilizador correcto, permitindo também uma certa mobilidade do utilizador. Caso este estivesse num outro local e desejasse receber convites para conferências, bastava registar a sua nova localização. Este protocolo apenas geria o estabelecimento de sessões. Assim que o utilizador se juntasse à sessão, a utilização do protocolo era terminada.

### 2.1.2 Simple Conference Invitation Protocol (SCIP)

O SCIP surgiu como mecanismo para convidar utilizadores para sessões *multicast* ou ponto-a-pronto. Criado em 1996 por Henning Schulzrinne, este protocolo combina aspectos do *Hypertext Transfer Protocol* (HTTP) e do *Simple Mail Transfer Protocol* (SMTP), permitindo assim a reutilização dos mecanismos de segurança presentes nestes e recorre ao *Transmission Control Protocol* (TCP) para protocolo de transporte. O SCIP utiliza endereços de e-mail como identificadores dos utilizadores, permitindo também reutilizar a infra-estrutura genérica do e-mail incluindo os registos MX de DNS e listas de endereços [25]. Ao contrário do SIPv1, a sinalização do SCIP continua após o estabelecimento da sessão permitindo, não só a troca de parâmetros entre sessões em utilização como também o fecho de sessões existentes.

### 2.1.3 Session Initiation Protocol (SIP)

O protocolo *Session Initiation Protocol* nasceu da combinação dos protocolos SCIP e SIPv1, durante o 35º e 36º encontro da *Internet Engineering Task Force* (IETF). O SIP teve por objectivo a criação, modificação e encerramento de sessões de multimédia [24], como por exemplo chamadas telefónicas através da Internet e conferências de multimédia. Os convites do SIP que permitem a criação de sessões, transportam descrições das mesmas que permitem aos utilizadores chegar a um acordo em relação aos parâmetros da sessão. Este protocolo é baseado em HTTP e pode utilizar UDP ou TCP como protocolo de transporte. Com o passar do tempo, o SIP foi-se tornando cada vez mais relevante na IETF, importância essa que resultou na criação de um grupo especificamente dedicado ao estudo e desenvolvimento do SIP. Após várias fazes de evolução, este mecanismo é utilizado, nos dias de hoje para convidar utilizadores para qualquer tipo de sessão de multimédia.

## 2.2 Funcionalidades do SIP

Como já mencionado anteriormente, o SIP estabelece, modifica e termina sessões de multimédia. Este protocolo é utilizado para convidar novos utilizadores para uma sessão já existente, ou a criarem uma nova. Para que uma sessão seja estabelecida é necessário que ambas as partes estabeleçam um acordo acerca dos parâmetros de media que serão usados. Todas estas operações recorrem a cinco funcionalidades disponibilizadas pelo SIP [24], que serão detalhadas de seguida.

**Localização do Utilizador** O SIP determina as localizações dos utilizadores através de um processo de registo, no qual um UA envia um pedido de registo indicando a sua localização actual. Este protocolo permite ainda que sejam efectuados vários registos de localizações diferentes, ou mesmo o registo de uma mesma localização em diferentes servidores, em simultâneo. Tal como um código de barras, cada utilizador possui um identificador único. Em SIP, os utilizadores possuem um SIP *Uniform Resource Locators* (URLs), sendo este composto por um nome de utilizador e um nome de domínio, muito à semelhança de um endereço de e-mail. Um exemplo deste tipo de identificadores poderá ser *SIP:UserA@domainA.pt*.

**Disponibilidade do Utilizador** Esta funcionalidade permite a um utilizador decidir se deseja receber ou não chamadas. Assim, caso um utilizador não atenda uma chamada, ele fica marcado no sistema como não estando disponível, sendo possível configurar o redireccionamento de chamadas quer para outros dispositivos telefónicos como para outras aplicações, como é o caso do *voicemail*.

**Capacidades do Utilizador** Para que uma sessão seja estabelecida é necessário que ambas as partes estabeleçam um acordo acerca dos parâmetros de media que serão usados. Esta troca de parâmetros é disponibilizada pelo SIP, uma vez que este faz uso de protocolos como o SDP para permitir a troca de informações relativas à sessão [22].

**Estabelecimento de uma Sessão** Este protocolo permite estabelecer sessões de multimédia entre dois ou mais utilizadores. O SIP é independente do tipo de sessão de multimédia e do mecanismo usado para a descrever, tendo apenas como objectivo a distribuição de descrições de sessões entre utilizadores. Após esta distribuição, o SIP apenas será novamente utilizado para renegociar e modificar os parâmetros da sessão, ou para a terminar.

**Gestão de uma Sessão** Ao utlizador é dada a capacidade de, durante uma sessão, renegociar os parâmetros da mesma. Assim, caso um utilizador pretenda adicionar uma componente de vídeo durante uma sessão de áudio, os parâmetros

da sessão são renegociados e esta é alterada.

## 2.3 Entidades SIP

Uma rede SIP é composta por quatro tipo de entidades SIP [3]. Cada uma possui características específicas que lhes permitem desempenhar um papel distinto em comunicações baseadas em SIP. De salientar que um mesmo dispositivo pode desempenhar o papel de mais do que uma entidade SIP.

### 2.3.1 User Agents

Esta entidade é responsável por iniciar e terminar as sessões através do envio e recepção de pedidos e respostas. Normalmente possui uma interface que é disponibilizada ao utilizador. Um exemplo de um *User Agent* (UA) é um programa com capacidade de realizar chamadas via Internet. O utilizador interage com a interface do programa e é o UA presente neste que envia e recebe pedidos e respostas. Existem dois tipos de UA: os *User Agent Client* (UAC) e os *User Agent Server* (UAS). Enquanto que os UAC são responsáveis pelo envio de novos pedidos, os UAS são as entidades que geram respostas aos pedidos recebidos. A resposta pode surgir na forma de aceitação, rejeição ou redirecionamento dos pedidos. Tal como definido em [24], um UA é uma entidade que desempenha tanto o papel de UAC como de UAS, representando um sistema *endpoint*.

### 2.3.2 Redirect Servers

Este tipo de servidores têm como objectivo ajudar os UAs a localizar um utilizador, fornecendo-lhes possíveis localizações onde o utilizador esteja contactável. Estas localizações não correspondem obrigatoriamente à localização exacta do utilizador podendo ser localizações de outros servidores a contactar. Os *Redirect Servers* não reencaminham o pedido para outros servidores, devolvendo apenas uma lista dos servidores ou utilizadores que devem ser contactados [3]. Assim, é da responsabilidade do UA do utilizador tentar localizar o utilizador desejado (Figura 2.1). O redireccionamento permite uma melhor escalabilidade dado que o esforço para descobrir a localização de um utilizador passa do núcleo da rede (servidores) para a periferia da mesma (UAs dos utilizadores) [24].

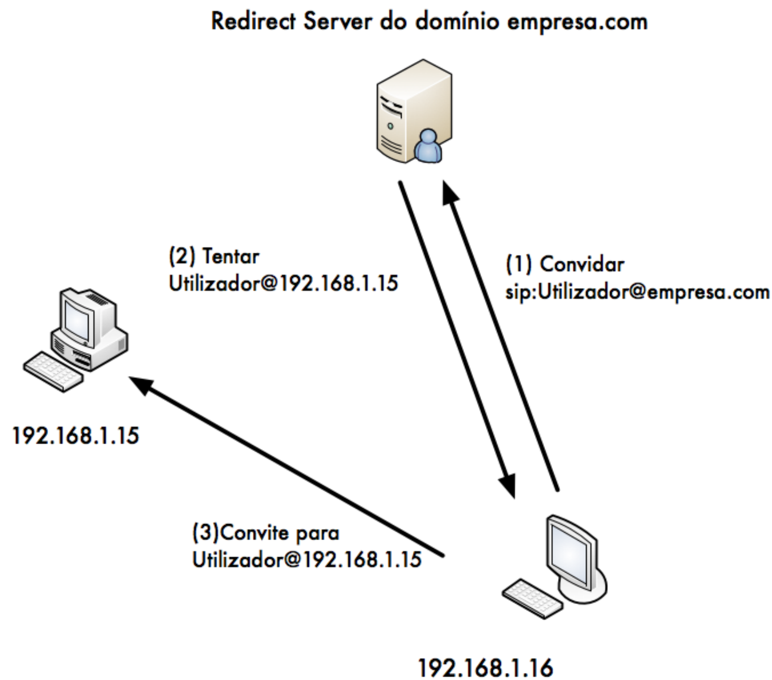



Figura 2.1: Funcionamento de um Redirect Server

Uma característica interessante dos *Redirect Servers* é o facto de ser possível implementar grupos de endereços. Se um pedido é enviado para um grupo de endereços, o servidor irá devolver diferentes localizações de acordo com as especificações do grupo. Um exemplo desta funcionalidade é o apoio ao cliente. Se um cliente contactar o serviço de manhã, o servidor irá devolver a localização de um UA. Se contactar de tarde, irá devolver a localização de outro UA.

### 2.3.3 Proxy Servers

Ao contrário dos *Redirect Servers*, quando um *Proxy Server* recebe um pedido, este tipo de servidores funcionam como um cliente e um servidor uma vez que efectuam pedidos em nome de outros clientes. Um *Proxy Server* interpreta e, se necessário, pode reescrever o pedido antes de o reencaminhar (Figura 2.2). Ao reencaminhar um pedido, se o *Proxy Server* tentar mais do que uma localização em simultâneo, o servidor é considerado um *Forking Proxy Server*. O pedido pode ser reencaminhado por este tipo de servidores de um modo sequencial ou paralelo. Tal como nos *Redirect Servers*, os *Proxy Servers* podem implementar grupos de endereços.

Existem três tipos de *Proxy Servers*: *Call Stateful*, *Stateful* e *Stateless* [24]. Um servidor *Call Stateful* está sempre entre as mensagens SIP enviadas entre os utili-

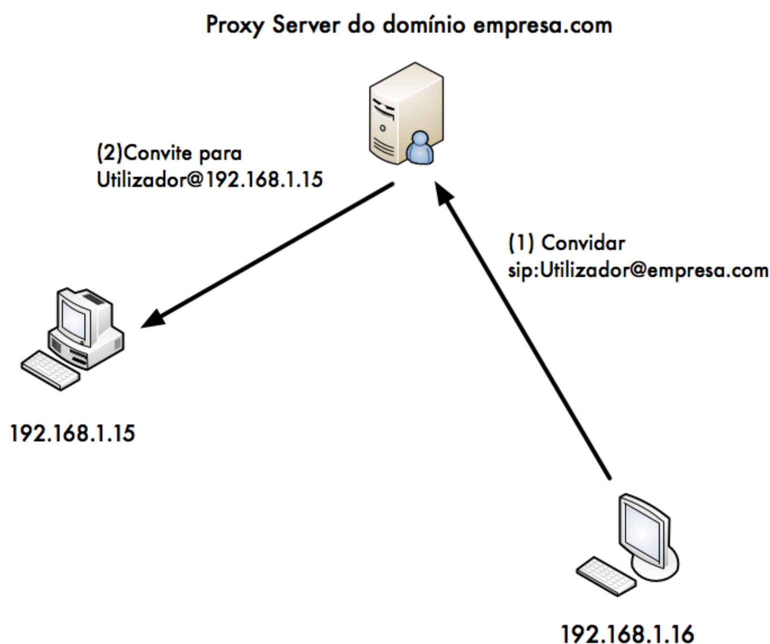



Figura 2.2: Funcionamento de um Proxy Server

zadores, fazendo sempre um registo do estado desde o momento em que a sessão foi estabelecida até ao momento que termina. Este tipo de servidores revelam-se úteis particularmente quando se pretende ser notificado aquando o término de uma chamada. Um servidor *Stateful* apenas guarda o estado relacionado com uma determinada transacção, ficando excluído do caminho seguido pelas mensagens SIP em transacções subsequentes. Um bom exemplo deste tipo de servidores são os *Forking Proxy Servers*. Quando são enviados pedidos de convites para diferentes localizações, este tipo de servidores requer o armazenamento de informação relativa ao estado de cada transacção de convite, de modo a saber que servidores já responderam. Um servidor *Stateless* não guarda qualquer tipo de estado. Após a recepção de um pedido, este tipo de servidores apenas o reencaminha eliminando de imediato qualquer informação relacionada com o estado.

### 2.3.4 Registrars

Um servidor *Registrar* é um UAS que aceita pedidos de registo de um utilizador, actualizando a informação relativaao mesmo num servidor de localização (Figura 2.3). Esta informação estará disponível e pronta a ser consultada tanto pelos *Proxy Servers* como pelos *Redirect Servers*. Este tipo de servidores está normalmente implementando em conjunto com *Proxy Servers* ou *Redirect Servers*.

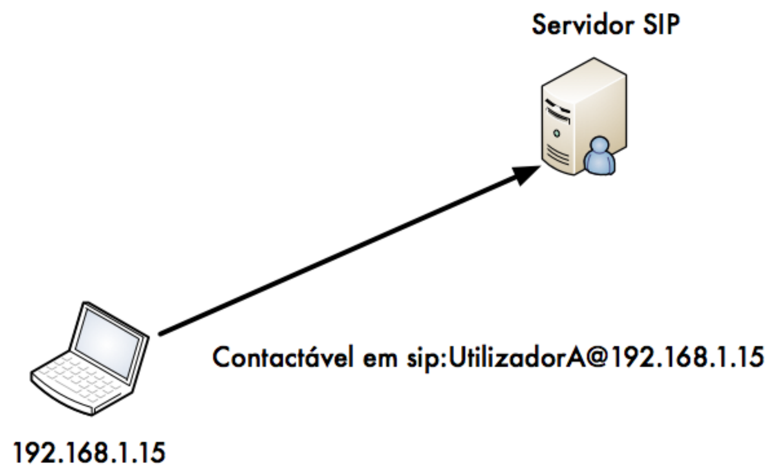



Figura 2.3: Pedido de Registo

### 2.3.5 Location Servers

Embora estes servidores não sejam considerados entidades SIP, são de extrema importância numa arquitectura que se baseie neste protocolo. Os *Location Servers* armazenam e devolvem possíveis localizações dos utilizadores quando pedido, informação esta que é normalmente obtida através dos servidores *Registrars*. Um detalhe importante é o facto de a comunicação entre os servidores de SIP e os *Location Servers* não ser feita recorrendo ao SIP, mas sim a protocolos como o *Lightweight Directory Access Protocol* (LDAP) [27].

## 2.4 Diferenças que distinguem o SIP de outros Protocolos

### 2.4.1 O SIP no Toolkit da IETF

O protocolo SIP apenas desempenha as funções para as quais foi pensado, recorrendo a outros protocolos e mecanismos para executar tarefas adicionais. Isto reproduz-se numa grande flexibilidade, uma vez que é possível estabelecer inúmeras combinações entre protocolos para realizar as tarefas desejadas. Qualquer novo protocolo criado pela IETF pode sempre ser usado em sistemas SIP sem a necessidade

de se efectuar modificações no SIP [3].

### 2.4.2 Estabelecer/Descrever uma Sessão

Quer o estabelecimento quer a descrição de uma sessão são conceitos claramente distinguidas pelo SIP. Relativamente ao estabelecimendo de uma sessão, ele localiza utilizadores e fornece meios que permitem uma troca de parâmetros, troca essa que definirá uma sessão. No que concerne a descrição de uma sessão, o SIP não não restringe nem o tipo de possíveis sessões a estabelecer nem o tipo de protocolos utilizados para as descrever. Este factor torna o SIP num protocolo capaz de interagir e cooperar com outros protocolos para o estabelecimento de sessões [3].

### 2.4.3 Periferia Inteligente

Uma vez que os servidores SIP apenas fazem o roteamento de mensagens SIP, são ignorantes relativamente ao conteúdo destas, responsabilizando os UA pelo processamento das mesmas. Isto torna os servidores SIP mais leves e eficientes, uma vez que não necessitam de armazenar o estado das transacções. O simplificar do núcleo do sistema, colocando toda a sua inteligência na periferia, torna o SIP num protocolo escalável uma vez que após efectuada a localização de um utilizador, é utilizada comunicação ponto-a-ponto entre os dispositivos dispersos pela periferia (UA).

Este princípio é aplicável à distribuição dos *Proxy Servers*. O núcleo do sistema está sujeito a uma maior carga, sendo por isso necessário torná-lo rápido e eficiente. Dado que servidores que armazenam estados são menos eficientes dos que os fazem, no núcleo apenas se encontram os servidores *stateless*. Já na periferia encontram-se os servidores *call stateful* e *stateful*, servidores estes que serão responsáveis por fazer o roteamento de mensagens baseando-se em variáveis que exijam um maior processamento, como por exemplo a identidade do utilizador que originou o pedido (Figura 2.4).

### 2.4.4 Reutilização de componentes

Vários são os componentes presentes noutros protocolos que são integrados no SIP. Este não se limita a integrar serviços, como também a disponibilizá-los aos utilizadores. Esta reutilização de componentes é uma das características mais importantes

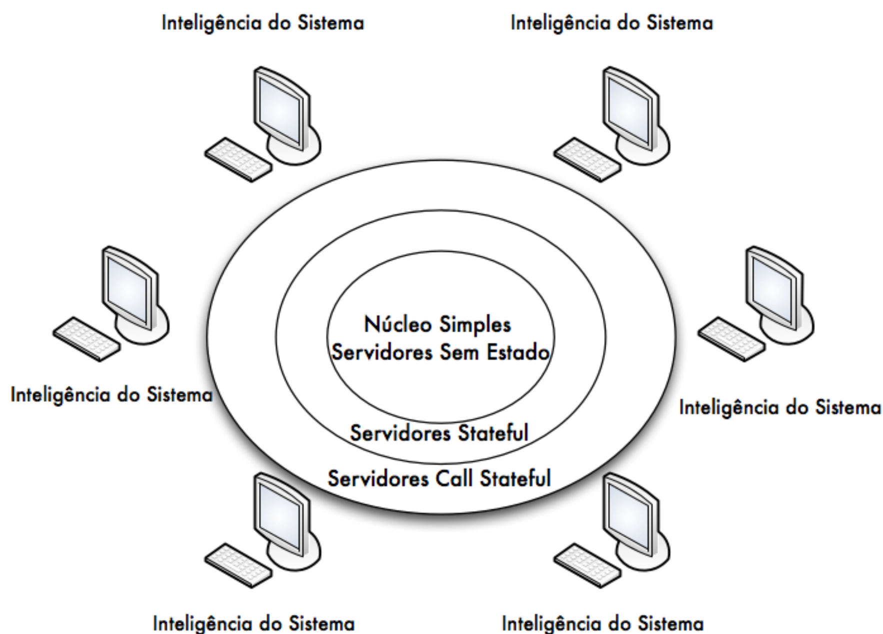



Figura 2.4: Distribuição de Servidores

do SIP e faz com que este seja completamente essencial quando a unificação das comunicações é um objectivo.

### O SIP e o HTTP

A implementação do SIP tem por base a implementação do protocolo *Hypertext Transfer Protocol* (HTTP) [24], regendo-se assim por um modelo de transacções pedido/resposta. Cada transacção consiste num pedido, que invoca uma função específica no lado do servidor, e numa resposta. A forma de codificar mensagens do protocolo é semelhante à do HTTP. Em vez de serem definidos novos mecanismos de segurança específicos ao protocolo SIP, este reutiliza sempre que possível modelos de segurança existentes derivados dos mecanismos de segurança do HTTP e do *Simple Mail Transfer Protocol* (SMTP) [15]. Estas semelhanças permitem a reutilização de código na implementação de uma aplicação. Por exemplo, um telefone móvel baseado em SIP com funcionalidades na Internet certamente permitirá a reutilização de código.

### Uso do conceito de roteamento do SMTP

O roteamento de mensagens SIP é extremamente semelhante ao roteamento de mensagens de correio electrónico [3]. Embora o protocolo SIP não tenha sido de-

senhado para transportar grandes quantidades de informação, é possível recorrer ao *Multipurpose Internet Mail Extension* (MIME) [4] para tal. O protocolo SIP foi desenhado para entregar mensagens instantâneas, urgentes e com pouca informação. Estas semelhanças tornam o SIP num protocolo adequado para sistemas de presença.

#### Reutilização de Infra-estruturas

Uma vez que os servidores de SIP ignoram o conteúdo das descrições das sessões, qualquer novo serviço que venha a ser implementado apenas necessita que os UA o suportem. A infra-estrutura torna-se, assim, automaticamente qualificada a fornecer esse novo serviço sem a necessidade de aplicar alterações à rede.

## 2.5 Arquitectura Base de Sistemas de Telefonia baseados em SIP

Uma arquitectura genérica de um sistema de telefonia baseado em SIP é constituída por um servidor de registo (*Registrar*), um *Proxy Server* ou *Redirect Server* e um servidor de localização. O servidor de registo é um servidor SIP que recebe pedidos de registo de modo a actualizar uma base de dados de localizações, contendo informações do utilizador presentes do pedido de registo. O servidor de localizações guarda e, quando é feito um pedido de localização, devolve possíveis localizações de utilizadores. Aquando a recepção de um pedido de convite, um *Redirect Server* pergunta a outro servidor SIP sobre a localização de um *User Agent*, devolvendo ao respectivo UA todas as possíveis localizações. Ao contrário deste, um *Proxy Server* actuará como cliente e irá contactar o UA especificado em nome do UA que originou o pedido.

Todos os utilizadores que pretendam usufruir do serviço necessitam de registar os seus equipamentos através do envio de um pedido REGISTER para o servidor *Registrar*. Deste modo, quando um cliente pretende contactar outro, o UA do equipamento do cliente envia um pedido de convite para um servidor SIP (*Proxy* ou *Redirect*). Este último envia um pedido ao servidor de localização acerca da localização do cliente a contactar. Se o servidor SIP for um *Proxy Server*, então este contacta as localizações devolvidas pelo servidor de localização. Se o servidor SIP for um *Redirect Server*, então este devolve as possíveis localizações ao UA que efectou o pedido. Uma vez alcançado o utilizador destino, os dois UA envolvidos na comunicação negoceiam os parâmetros a serem usados na sessão. Finda esta ne-

gociação, um protocolo ponto-a-ponto é usado para transmitir os pacotes, pacotes estes contendo o tipo de media acordado durante a negociação dos parametros da sessão (Fig. 2.5). O protocolo SIP só volta a ser usado quando um utilizador pretende alterar o tipo de sessão (por exemplo, adicionar vídeo à sessão) ou quando um utilizador a pretende terminar [19].

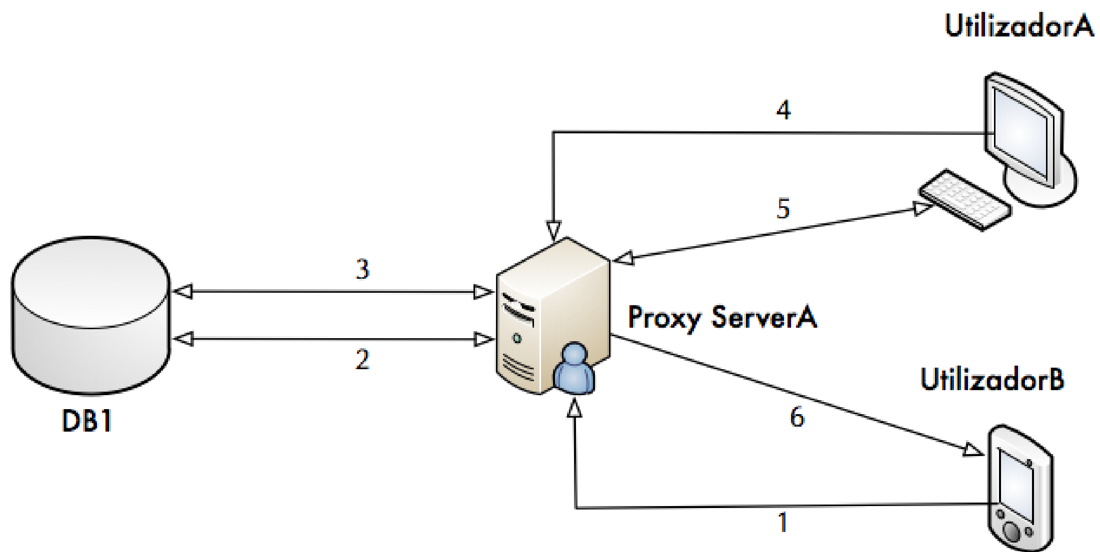



Figura 2.5: Estabelecimento básico de uma sessão SIP

## 2.6 Aumento da confiabilidade

Componentes vitais fazem parte integrante de qualquer sistema. Quando um destes falha, o sistema não será capaz de fornecer o serviço correctamente aos utilizadores. De modo a melhorar a confiabilidade do sistema, servidores substitutos serão incluídos na arquitectura deste sistema. Em [30] estão descritas inúmeras técnicas e arquitecturas com o objectivo de melhorar a confiabilidade do sistema, arquitecturas estas que serão especificadas sumariamente nas subsecções seguintes.

### 2.6.1 Baseado no cliente

Neste tipo de arquitectura, a lógica de *Failover* é colocada do lado do cliente. O equipamento do cliente conhece os endereços de IP do servidor primário e do secundário e, em vez de se registar apenas num destes, o equipamento envia a

mensagem de REGISTER para ambos os servidores. Quando é feito um pedido de convite e no caso de falha do servidor primário, o mesmo pedido de convite é enviado para o servidor secundário que irá substituir o servidor primário.

Esta arquitectura pode fazer uso do DNS para permitir a adição ou substituição de servidores sem recorrer a modificações na configuração do equipamento manualmente. No entanto, quer esta arquitectura como todas as outras que colocam a lógica de *Failover* no lado do cliente, eliminam a independência do servidor sobre a configuração do cliente. Um outro problema com este tipo de arquitecturas prende-se com os equipamentos dos clientes, dado que, nem todos os equipamentos suportam configurações que enviam múltiplos pedidos de registo para vários servidores. Este tipo de arquitecturas não pode, então, ser implementado em todos os sistemas de telefonia baseados em SIP.

### 2.6.2 Baseado em DNS

Quando um utilizador deseja contactar alguém, o seu equipamento adquire um registo DNS SRV [9] para obter o endereço IP dos servidores primários e secundários. Este SRV possui informação acerca da prioridade de cada servidor, de modo a que o cliente saiba que servidor contactar primeiro (Fig 2.6) [23]. Na figura 2.6 é apresentado um exemplo do funcionamento desta arquitectura. Aqui, o equipamento do cliente deverá primeiro contactar o servidor de SIP P1 uma vez que este possui um valor de prioridade mais baixo. Já o servidor secundário deve monitorizar o servidor primário de modo a que quando este falhe, o servidor secundário possa actualizar o registo SRV.

### 2.6.3 Replicação de base de dados

Quer o registo em dois servidores quer a obtenção de um registo DNS SRV podem não ser suportados por todos os equipamentos. O equipamento deve apenas registar-se no servidor primário, que depois irá propagar o pedido de registo para todos os servidores secundários.

Quando são utilizadas bases de dados para guardar os registos dos utilizadores, o servidor primário recebe o pedido de registo, guarda-o na base de dados, e depois replica-a de modo a garantir que todos os servidores possuam os mesmos registos. Se o servidor primário falha, o secundário tem acesso à base de dados replicada que contém todos os registos dos utilizadores, permitindo assim que o serviço de telefonia seja disponibilizado correctamente (Fig. 2.7).

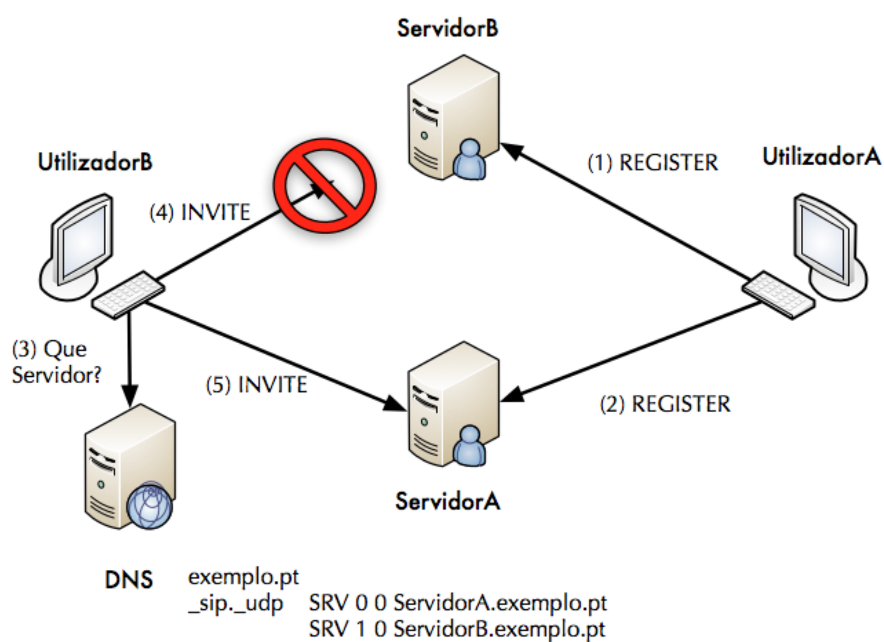



Figura 2.6: Failover baseado em DSN

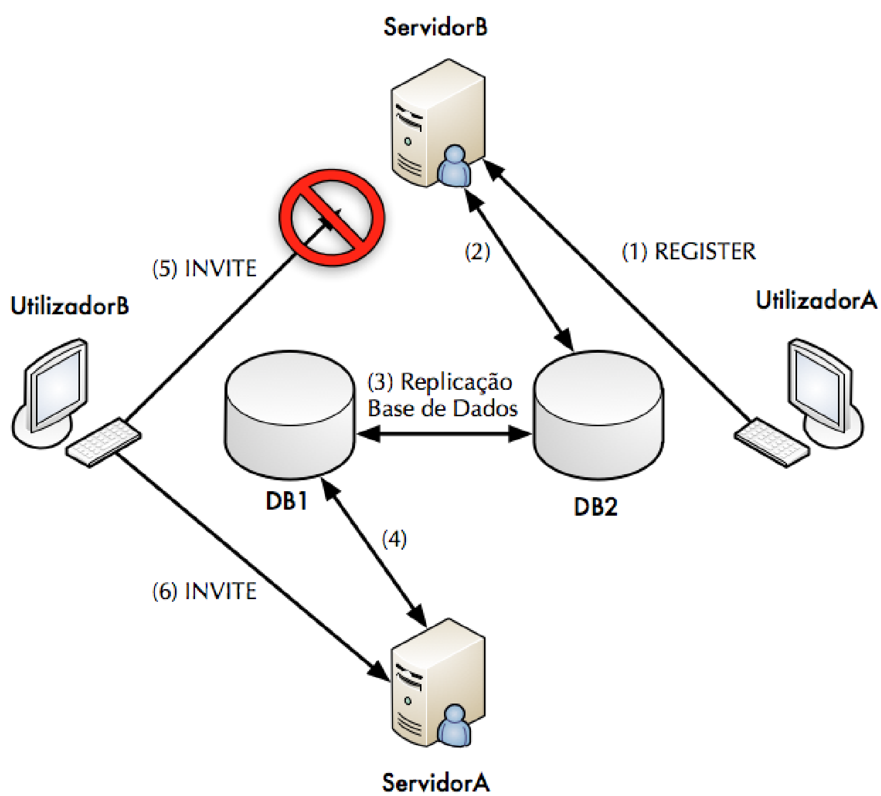



Figura 2.7: Failover baseado em replicação de bases de dados

### 2.6.4 Assumir o endereço IP

Todos os servidores correm em diferentes *hosts*, mas na mesma rede *Ethernet*, com a mesma configuração e com o acesso à mesma base de dados. O servidor secundário monitoriza o primário e quando este falha, o secundário assume o endereço de IP do primário. Desta forma, o equipamento contacta sempre o mesmo endereço IP, apesar de ser um servidor diferente que receberá os pedidos (Fig. 2.8).

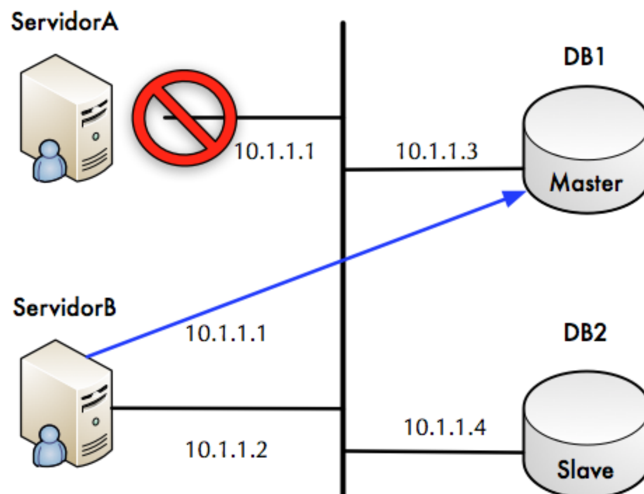



Figura 2.8: Endereço de IP apoderado por parte do servidor secundário

### 2.6.5 Pooling de servidores

Nesta arquitectura, o equipamento do cliente é visto como um utilizador de uma *pool* (PU) e, quer o servidr primário quer o secundário são considerados elementos de uma *pool* (PE) na "Pool de Servidores SIP". Já as bases de dados são consideradas elementos de uma *pool* na "Pool de Base de dados". Tanto os servidores SIP como as bases de dados são registados num *Name Server* (NS) que os supervisiona e, em caso de falha, remove o PE respectivo da *pool*.

Quando um equipamento deseja contactar um servidor SIP, primeiro interroga o *Name Server* para obter a lista de servidores SIP disponíveis. Esta lista contém os endereços IP dos servidores, bem como as prioridades (tal como na arquitectura baseada no DNS). Quando um servidor SIP é contactado, este interroga o NS responsável pelas bases de dados e obtém o endereço IP das mesmas (Fig. 2.9).

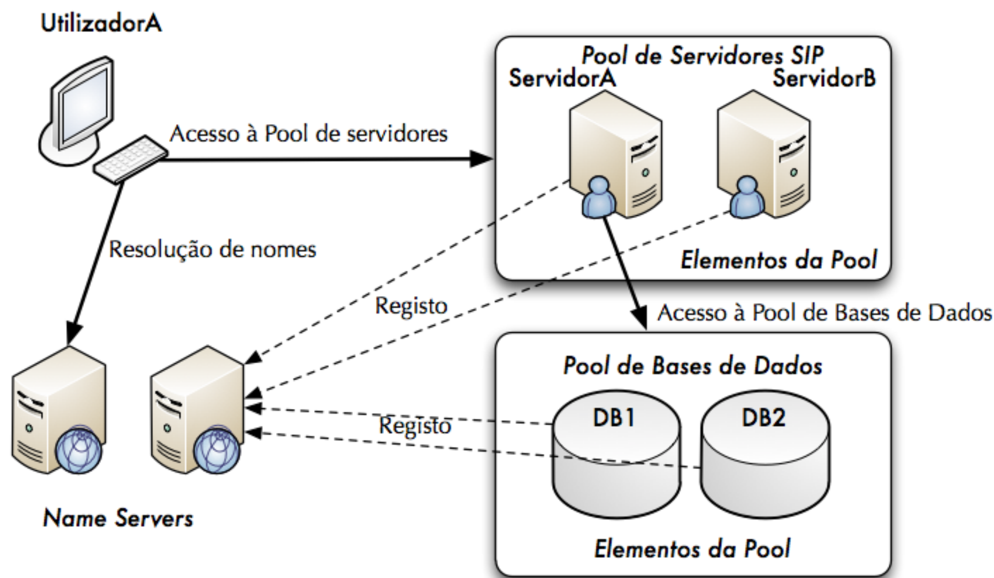



Figura 2.9: Pooling de servidores

### 2.6.6 Infrastutura de SIP usando *Content Addressable Networks*

Em [2], os autores descrevem uma arquitectura onde um conjunto de servidores SIP formam uma *Content Addressable Network* (CAN). As mensagens SIP são roteadas usando tabelas de *hash* distribuídas baseadas em endereços SIP. Cada servidor é responsável por uma sub-região única do espaço de *hash* e possui uma base de dados privada, que é utilizada para guardar os registos dos utilizadores. Para além de serem responsáveis por uma sub-região, cada servidor possui uma lista que contém informações sobre os seus vizinhos na rede. Quando um servidor falha, o seu vizinho assume a responsabilidade pela sub-região, permitindo que futuros pedidos possam ser processados sem problemas.

Quando um UA envia um pedido SIP, este segue para um dos nodos na rede. Este nodo verifica se é responsável pela zona que contém o valor de *hash* do UA destino. Caso seja responsável, o nodo realiza as operações usuais de um servidor SIP, quer reencaminhando o pedido para o UA destino ou retornando uma mensagem de erro por não existir um registo do utilizador destino. Caso o nodo não seja responsável, este reencaminha o pedido para o vizinho cuja zona possua a menor distância cartesiana para o UA destino.

## 2.7 Aumento de escalabilidade

Um factor importante que afecta a escalabilidade de um sistema é a taxa de solicitação. Se o número de pedidos dos utilizadores feitos simultâneamente é elevado, a capacidade do servidor em os processar é degradada.

De modo a permitir um balanceamento da carga, todos os servidores redundantes necessitam de trabalhar em conjunto. Deste modo, a escalabilidade do sistema melhora e este será capaz de processar mais pedidos dos utilizadores. Arquitecturas como as baseadas em DNS, em identificadores, em *network address translation*, em múltiplos servidores com o mesmo endereço IP, ou numa arquitectura de duas fases são também descritas em [30], e tentam melhorar a escalabilidade do sistema. Algumas arquitecturas foram modificas a partir das arquitecturas supramencionadas para melhorar a confiabilidade do sistema. Alguns exemplos destas arquitecturas serão expostos de seguida.

### 2.7.1 Baseada em DNS

Registos SRV e um *Name Authority Pointer* (NAPTR) podem ser utilizados para balanceamento de carga. Em vez de apenas fazer uso do campo de prioridade dos registos SRV, esta arquitectura inclui nestes registos o campo peso. Um exemplo destes registos é apresentado em baixo:

*abc.com*
*_sip._udp 0 50 server1.abc.com*
*0 50 server2.abc.com*
*1 0 backup.somewhere.com*

Neste exemplo, os servidores *server1* e *server2* possuem a mesma prioridade e cada um deles irá receber 50% da carga do sistema. O servidor *backup* será apenas utilizado como substituto destes.

### 2.7.2 Baseado em Identificadores

Para implementar uma arquitectura de balanceamento de carga usando identificadores, o espaço dos utilizadores é dividído em grupos que não se sobrepõem. Recorrendo a uma função de *hash*, é possível mapear os identificadores dos utilizadores com o grupo que processa o registo do utilizador. Por exemplo, o primeiro servidor SIP processará os pedidos que possuam como primeira letra do identificador letras de 'a' a 'h' (Fig. 2.10).

Para implementar esta arquitectura é necessário adicionar um servidor com elevada capacidade de processamento, para reencaminhar os pedidos para cada servidor de acordo com o identificador. Ademais, cada servidor possui uma base de dados privada sem a necessidade de interacção com outras bases de dados.

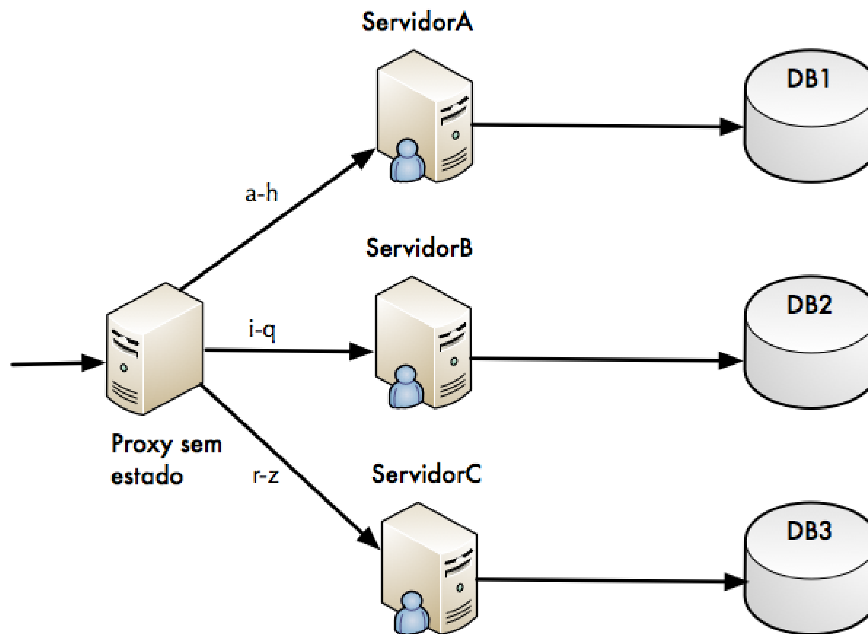



Figura 2.10: Distribuição de carga baseada em identificadores

Para alcançar uma distribuição uniforme de pedidos de chamadas, uma função de *hash* como o SHA1 deve ser utilizada.

### 2.7.3 *Network Address Translation (NAT)*

Um NAT pode ser utilizado para distribuir uma chamada recebida para um dos muitos *hosts* privados correndo servidores SIP. Uma vez que os servidores SIP possuem uma natureza de estado de transação, é necessário assegurar que todos as retransmissões adjacentes são processadas pelo mesmo servidor interno, forçando o NAT a manter um estado de transações durante a transação.

### 2.7.4 Múltiplos servidores com o mesmo endereço IP

Todos os servidores redundantes devem estar na mesma rede *Ethernet* e usar o mesmo endereço IP. Um router da rede é configurado para reencaminhar os pacotes recebidos para um dos endereços MAC dos servidores. Múltiplos algoritmos podem

ser utilizados para reencaminhar pacotes para o servidor com o menor índice de carga, como por exemplo o *round robin* ou tempo de resposta do servidor. Esta arquitectura é apenas recomendada quando os servidores SIP não possuem estado (não guardam qualquer estado transaccional dos pedidos).

### 2.7.5 Arquitectura de duas fases

Esta arquitectura é constituída pela combinação das arquitecturas baseadas em DNS e baseadas em identificadores. Desta forma, consequentemente obtendo uma arquitectura altamente escálável e confiável.

Tal como na arquitectura baseada em identificadores, esta possui uma primeira fase de reencaminhadores. A arquitectura baseada em DNS é utilizada para definir que reencaminhador da primeira fase será contactado. Usando uma função de *hash*, o reencaminhador envia o pedido para um aglomerado de servidores. É depois utilizada de novo a arquitectura baseada em DNS para definir que membro deste aglomerado é responsável por processar o pedido. Todas as bases de dados do aglomerado são replicadas (Fig. 2.11).

Apesar desta arquitectura adicionar uma primeira fase de reencaminhadores, tal facto não afecta a transmissão de qualquer tipo de media, uma vez que esta é efectuada ponto-a-ponto. A utilização do DNS permite a co-localização dos servidores.

### 2.7.6 Optimização de uma infra-estrutura SIP usando *Content Addressable Networks*

A arquitectura descrita na subsecção 2.6.6 pode ser optimizada de modo a melhorar a escalabilidade do sistema. Para tal, é possível utilizar uma técnica de *caching* denominada *zone caching* [2]. Esta técnica guarda em *cache* tanto as coordenadas da zona como o endereço IP do nodo responsável por esta zona. Estas coordenadas são gravadas pelo nodo presente no cabeçalho *Via* da mensagem SIP. Aquando a recepção de um pedido SIP, o nodo contactado aplica a função de *hash* sobre o destinatário do pedido, procurando na *cache* por uma entrada contendo a zona para a qual o pedido SIP deve ser reencaminhado. Se tal entrada for encontrada, o servidor reencaminha o pedido de SIP directamente para o nodo presente na entrada. Se não for encontrada nenhuma entrada, então o pedido SIP será reencaminhado utilizando as regras normais de reencaminhamento da arquitectura *Content Addressable Networks*.

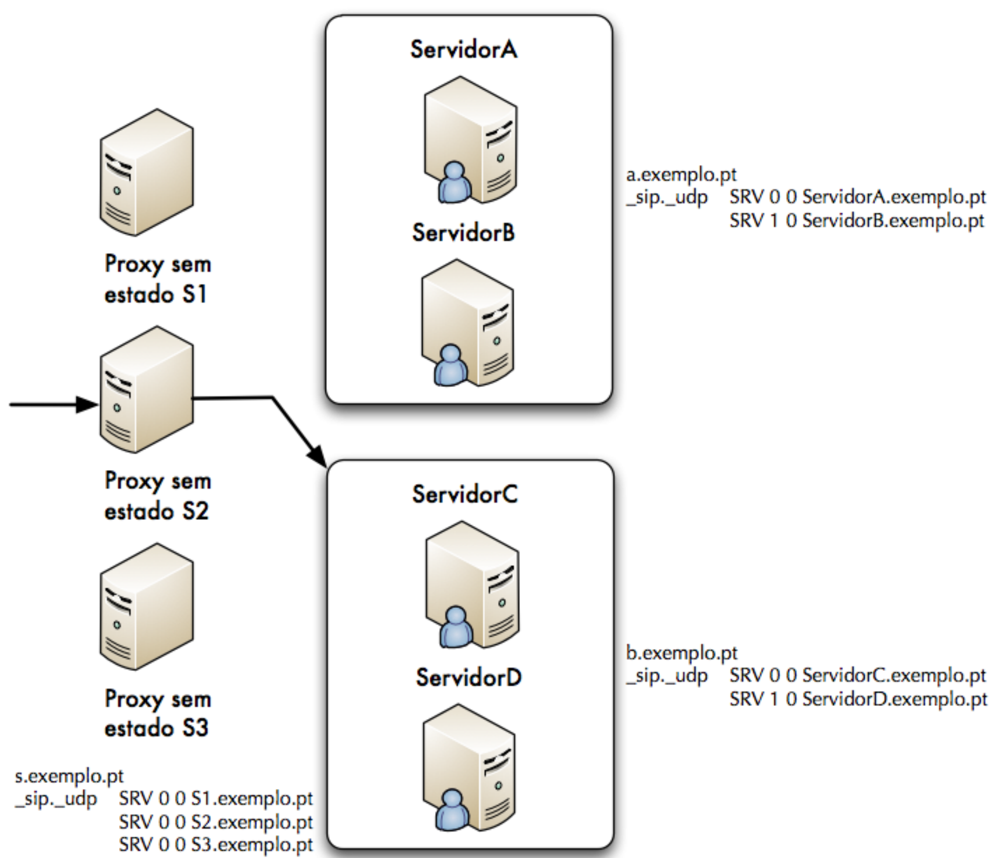



Figura 2.11: Arquitectura escalável e confiável baseado em duas fases

# Capítulo 3

# Solução VoIP da Wavecom S.A.

A Wavecom S.A. [32] é uma empresa na área da Engenharia de Comunicações, nascida em 2000, que disponibiliza uma infra-estrutura baseada em SIP permitindo a migração dos sistemas tradicionais de telefonia para sistemas de voz sobre IP de um modo transparente. Esta solução é baseada em ferramentas open source, suporta até 7500 chamadas em simultâneo envolvendo quatro componentes principais: *Media Gateway* (MGW), *Accounting, Billing and Quality of Service Report Server* (SABQR), *Session Border Controller* (SBC) e o *IP Private Branch Exchange* (iPBX) (Figura 3.1).

A MGW estabelece a ponte entre o sistema tradicional de telefonia, nomeadamente linhas *public switched telephone network* (PSTN) ou centrais telefónicas tradicionais, e a telefonia VoIP. Este componente é responsável pela conversão de chamadas de voz ou fax entre a rede pública e a rede IP. A conversão inclui compressão/descompressão de voz, roteamento de chamadas e controlo de sinalização.

O SABQR é responsável por todas as operações relacionadas com contabilidade, facturação e análise de qualidade do serviço relativamente a todas as comunicações.

Todo o tráfego e toda a sinalização de VoIP que flui de e para fora das redes geridas pelo sistema passa pela SBC. O principal objectivo deste componente é assegurar a qualidade do serviço e proteger a rede VoIP. É a SBC que decide se uma chamada pode ser realizada, controlando o fluxo de dados, sendo instalada como fronteira entre a rede privada da instituição e todas as outras redes VoIP.

O iPBX é um sistema de telefonia que permite comunicações de voz sobre uma rede de pacotes e é capaz de operar com a rede convencional de telefonia. Visto que as funcionalidades do iPBX são baseadas em software, torna-se barato e fácil adicionar novas funcionalidades. Uma vez que os sistemas convencionais de telefonia tendem a ser substituídos pelos sistemas de VoIP, a MGW deixará de ser necessária

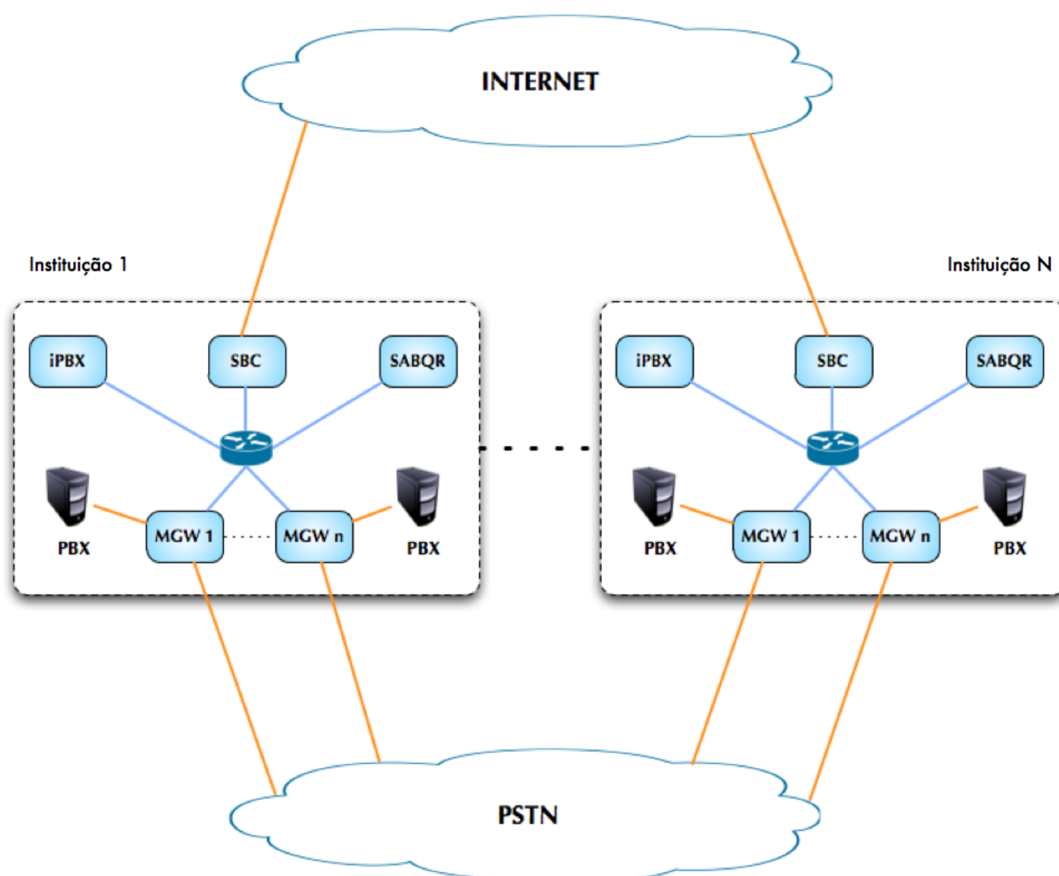



Figura 3.1: Solução de VoIP da Wavecom S.A.

nesta solução. Para além disto e tendo em conta que o tráfego VoIP passa pelo iPBX, este último poderá comprometer o sistema a nível de performance. É por estas razões que será este componente estudado e redesenhado para oferecer uma alta confiabilidade e escalabilidade.

## 3.1 Ferramentas de Software do iPBX

Um iPBX compreende cinco diferentes ferramentas de software: o FreeRADIUS, o OpenSER, uma base de dados MySQL, o Asterisk e o FreePBX.

### 3.1.1 FreeRADIUS

O *Remote Authentication Dial In User Service* (RADIUS) é um protocolo de rede que para além de possibilitar a gestão de contas, permite também a autorização

e autenticação de computadores que pretendam usufruir de um serviço disponível na rede. Seguir um modelo cliente/servidor, segurança de rede, mecanismos de autenticação flexiveis e a extensabilidade do protocolo são funcionalidades chave do RADIUS [21]. Embora este protocolo tenha sido criado inicialmente apenas com o objectivo de autorização e autenticação, foi mais tarde adaptado para permitir a gestão de contas [20].

O FreeRADIUS é uma implementação modular e de alta performance de um servidor RADIUS. Uma vez que suporta todos os protocolos de autenticação mais comuns, esta ferramenta é responsável pela autenticação dos utilizados do sistema. O FreeRADIUS possui ainda uma ferramenta web de gestão baseada em PHP.

### 3.1.2 OpenSER

O OpenSER surgiu em 2005 como um *Proxy Server* de SIP, roteador de chamadas e servidor de registo de SIP. Esta ferramenta é muito utilizada em sistemas de VoIP e de *Instant Messaging*. A performance e robustez do OpenSER permite utilizá-lo para gerir e servir milhões de utilizadores [8]. Esta ferramenta possui um modelo de plug-ins bastante flexível para aplicações de terceiros. Assim, podem-se criar novas funcionalidades e acrescentá-las com facilidade ao OpenSER, como por exemplo, os módulos RADIUS, PRESENCE, SMS, entre outros[8]. O OpenSER é assim flexível, portável e extensível. Uma vez que foi desenvolvido em *ANSI C*, esta poderosa ferramenta pode ser portada para qualquer plataforma.

Embora o OpenSER tenha sido desenvolvido e pensado para desempenhar o papel de *Proxy Server* de SIP, a adição de novos módulos permitiu o uso de novas funcionalidades e novas tarefas que o OpenSER é perfeitamente capaz de executar. Assim, esta ferramenta pode desempenhar o papel de balanceador de carga, firewall de SIP, servidor de presença, instant messaging e de *Network Address Translation* (NAT) transversal.

O OpenSER foi descontinuado em 2008, originando dois novos servidores: o Kamailio e o OpenSIPS. Apesar disto, na solução da Wavecom, o OpenSER continua a desempenhar o papel de *Proxy Server* de SIP e de servidor de registo de SIP.

### 3.1.3 Asterisk

O Asterisk é uma implementação de software de um *Private Branch Exchange* (PBX) telefónico. Tal como qualquer outro PBX, o Asterisk permite que os telefones

ligados ao sistema possam efectuar chamadas entre si e possam conectar-se a outros serviços telefónicos como a PSTN e o VoIP.

Este software inclui bastantes funcionalidades tais como correio de voz, chamada em conferência, resposta de voz interactiva e distribuição automática de chamadas. Para além disso suporta também uma ampla gama de protocolos de vídeo e de VoIP, incluindo o SIP, o *Media Gateway Control Protocol* e o H.323. O Asterisk pode interagir com a maioria dos telefones SIP, desempenhando o papel tanto de *Registrar* como de *gateway* entre os telefones IP e a PSTN.

Uma vez que aplicações como voicemail, conferências, fila de espera de chamadas e music on hold fazem parte do Asterisk como aplicações padrão, esta ferramenta funciona como PBX, servidor aplicacional e servidor de média na solução da Wavecom S.A. Para além de desempenhar estes papéis, o Asterisk também funciona como servidor de registo de SIP, sendo esta ferramenta responsável por manter o estado de cada extensão do sistema.

### 3.1.4 MySQL

O MySQL é um sistema de gestão de bases de dados relacionais que é executado como um servidor, fornecendo acesso às bases de dados a múltiplos utilizadores. Este sistema de gestão é open source e é dos mais populares. Tanto o Asterisk como o OpenSER podem escrever/ler informação para/de uma base de dados MySQL.

### 3.1.5 FreePBX

Gerir um sistema com ferramentas de software como o Asterisk e o OpenSER pode ser oneroso quando na presença de milhares de extensões e utilizadores. O objectivo do FreePBX é facilitar este processo através de uma interface gráfica que permita ao administrador uma melhor gestão do sistema.

## 3.2 Tolerância a faltas na solução da Wavecom S.A.

O *Distributed Replicated Block Device* (DRBD) é um sistema de armazenamento distribuído que garante tolerância a faltas espelhando conjuntos de blocos via rede [10], podendo ser então visto como um sistema de *Redundant Array of Independent Disks* em modo 1 (RAID-1) baseado em rede. O DRBD espelha a informação

em tempo real e de um modo transparente. Esta pode ser espelhada quer de um modo síncrono ou assíncrono. No primeiro, a aplicação responsável pela escrita é notificada apenas quando ambos os dispositivos efectuaram tais alterações. Já no modo assíncrono, a aplicação responsável pela escrita é notificada mal as alterações sejam escritas num dispositivo, embora estas ainda não tenham sido propagadas para o outro dispositivo.

No DRBD cada recurso ou desempenha um papel primário ou um papel secundário. Qualquer tipo de operações de escrita ou leitura podem ser efectuadas num dispositivo se este for primário. Caso seja secundário, é negada qualquer possibilidade de executar tanto operações de escrita como de leitura. Este tipo de dispositivos apenas poderá receber a informação espelhada e enviada pelo dispositivo primário (Figura 3.2). O papel desempenhado pelos dispositivos pode ser alterado manualmente ou recorrendo a uma aplicação de gestão de *clusters* que seja responsável por esta troca de papéis.

Existem dois modos de funcionamento do DRBD: o modo primário único e o modo duplo primário. No modo primário único existe apenas um primário, o que garante que não irão ocorrer problemas de concorrência, podendo ser utilizado com qualquer sistema de ficheiros convencional. No modo duplo primário podem existir vários recursos primários, o que poderá resultar num problema de concorrência. Este modo requer a utilização de um sistema de ficheiros partilhado que implemente um gestor de bloqueio distribuído, como por exemplo o *Global File System*.

Relativamente ao iPBX, a solução da Wavecom S.A. é composta por duas máquinas e já oferece garantias de confiabilidade através do uso do DRBD no modo primário único. O iPBX no modo activo é considerado o primário, e todas as alterações realizadas neste iPBX são reflectidas de um modo síncrono no iPBX secundário, que se encontra num estado de espera. Quando o iPBX primário falha, o secundário é promovido e assegura a continuação da prestação correcta do serviço.

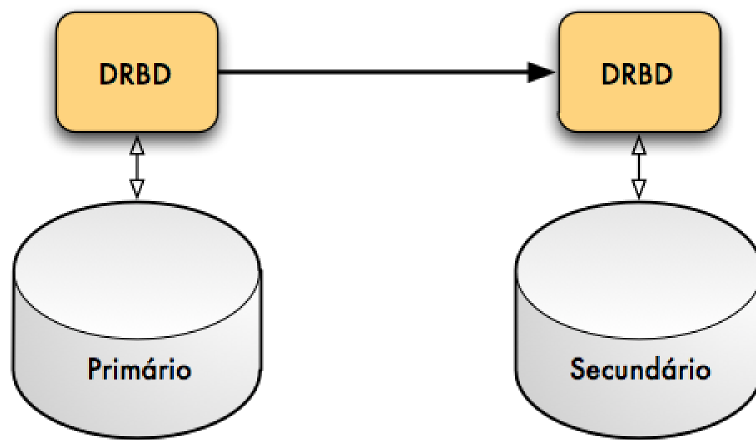



Figura 3.2: Funcionamento do DRBD

# Capítulo 4

# Arquitectura de *Failover* da Solução

A solução da Wavecom S.A. utiliza o DRBD em modo primário com replicação síncrona para oferecer garantias de confiabilidade do sistema, o que implica que uma máquina esteja sempre em estado de espera. Ter uma solução que funcione em modo activo/passivo é um desperdício de recursos quando o número de máquinas disponíveis é baixo. Ao modificar o sistema de modo a permitir uma execução em modo activo/activo é possível aproveitar ao máximo todos os recursos existentes no sistema, garantindo ainda que a ocorrência de uma falha numa máquina não afectará nem a disponibilidade nem a prestação correcta do serviço.

Para além das garantias de confiabilidade, é também pretendido que a solução apresentada seja o mais transparente possível para os clientes (telefones IP). Quando uma máquina falha, os telefones IP têm de ser capazes de contactar e usufruir do serviço sem a necessidade de realizar qualquer esforço ou tarefa extra.

O FreeRADIUS é utilizado para autenticar os utilizadores e para tal necessita apenas de aceder à informação disponível no OpenSER. Garantindo que todos os nodos OpenSER têm acesso à mesma informação, resolve-se o problema de *Failover* no que ao FreeRADIUS concerne. Já o FreePBX necessita de aceder à informação disponibilizada nos ficheiros de configuração do Asterisk. Assim, tal como o FreeRADIUS, assegurando que todos os nodos Asterisk possuem os mesmos ficheiros de configuração garante-se o *Failover* relativamente ao FreePBX. Em suma, resolvendo os problemas de *Failover* do OpenSER e do Asterisk, resolvem-se, simultâneamente, os problemas de *Failover* do FreePBX e do FreeRADIUS.

Ao longo deste capítulo irá ser apresentada uma solução para o problema do *Failover* do iPBX. Em primeiro lugar irá ser abordada a transparência do sistema,

seguindo-se da solução que permite a replicação de informação relativamente ao OpenSER e *a posteriori* ao Asterisk.

## 4.1 Transparência do sistema

A solução apresentada centra-se no *Linux Virtual Server* (LVS) [28] para garantir a transparência do sistema relativamente aos telefones IP. O LVS é um servidor de alta confiabilidade e escalabilidade cuja implementação é realizada num cluster de servidores reais, com um balanceador de carga a operar num sistema Linux (*Linux Director*). A arquitectura do cluster de servidores é completamente transparente para os clientes, interagindo com o cluster como se trata-se de um único servidor de alta performance.

Entre as várias topologias de redes disponíveis para implementar o LVS, escolheu-se aquela que combina o funcionamento de *Linux-Directors* (máquinas que executam o LVS) e servidores reais na mesma máquina (Figura 4.1). Assim, apenas com duas máquinas é possível alcançar alta confiabilidade do sistema bem como um balanceamento de carga. Durante a prestação do serviço de telefonia VoIP, apenas uma das máquinas funciona como *Linux-Director* activo, enquanto a outra máquina está num estado de espera. O *Linux-Director* activo aceita tráfego recebido através de um endereço IP virtual, endereço esse que será contactado por todos os telefones IP. As duas máquinas monitorizam-se uma à outra recorrendo a uma ferramenta de gestão de clusters (como é o exemplo do *heartbeat* [13]) e, caso o *Linux-Director* activo falhe, o *Linux-Director* no estado de espera assume o endereço IP virtual e assegura a prestação do serviço. As ligações são sincronizadas entre a máquina activa e a máquina em espera assegurando que as ligações existentes não são quebradas em caso de *Failover*, desde que o servidor real responsável por elas continue em execução.

Quando uma conexão originada de um cliente é recebida por um *Linux-Director*, este escolhe que servidor real (ele próprio ou outra máquina) será responsável por processar essa conexão. Durante o tempo de vida dessa conexão todos os pacotes serão reencaminhados para o mesmo servidor real, de modo a manter a integridade da conexão.

Assim, caso o iPBX responsável pelo *Linux-Director* primário falhe, outro iPBX irá assumir este papel, permitindo aos utilizadores continuarem a contactar sempre o mesmo endereço IP. A implementação LVS na solução permite o uso de qualquer tipo de telefone IP, colocando assim a lógica de *Failover* no lado dos servidores e

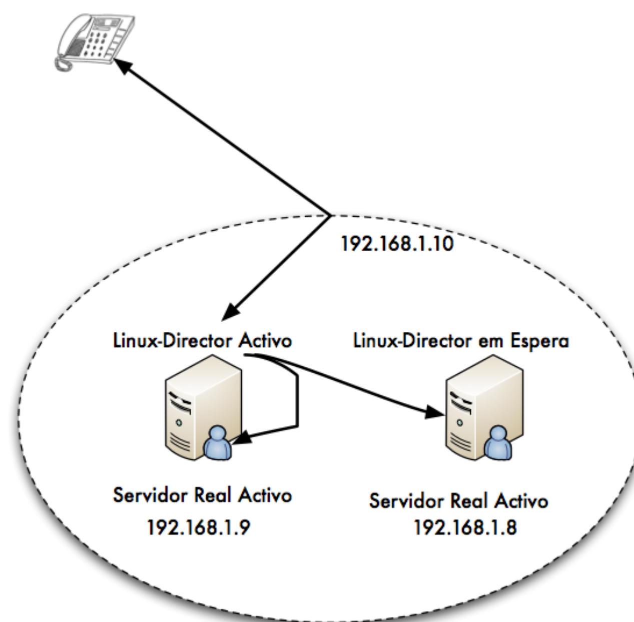



Figura 4.1: Funcionamento do LVS

não do lado dos clientes.

## 4.2 Problema de replicação do OpenSER e uma solução

Enquanto *Proxy Server* de SIP, o OpenSER possui ficheiros de configuração que instruem o modo como cada chamada deve ser reencaminhada [8]. O ficheiro *openser.cfg* contém os parâmetros globais de configuração, especifica que módulos devem ser carregados e possui ainda uma script que será executada quando receber um pedido. Este ficheiro é estático e é obrigatório que seja igual em qualquer iPBX do cluster.

Uma vez que o OpenSER também desempenha o papel de servidor de registo de SIP e de modo a permitir a persistência dos dados, esta ferramenta de software pode armazenar informação numa base de dados onde estará protegida contra falhas de energia e reinicializações da máquina. O OpenSER suporta quatro modos de acesso: esquema de memória, esquema *write-through*, esquema *write-back* e esquema de base de dados. Quando o esquema de memória é utilizado, os contactos registados não sobreviverão a uma reinicialização da máquina, dado que nada é armazenado e escrito na base de dados garantindo assim a persistência da informação. No esquema

*write-through* todas as alterações na memória são reflectidas imediatamente na base de dados. Apesar desta operação tornar este esquema bastante lento, aumenta simultaneamente a confiabilidade do sistema. Se se utilizar o esquema *write-back*, todas as alterações são feitas na memória e a sincronização com a base de dados é efectuada ciclicamente com a ajuda de um temporizador. Este temporizador elimina todos os contactos expirados e descarrega todos os novos contactos ou que foram modificados para a base de dados. Este esquema é bastante útil caso se verifiquem altos picos de carga no sistema, sendo necessário processá-los o mais rápido possível. Por último, o esquema de base de dados não faz uso de memória e todas as operações são efectuadas directamente sobre a base de dados.

Tendo em conta que o ficheiro de configuração do OpenSER é estático, basta copiá-lo de uma máquina para outra antes de se disponibilizar o serviço de telefonia VoIP. Relativamente à restante informação e uma vez que o OpenSER na solução funciona recorrendo ao esquema de base de dados, a solução para o problema da replicação do OpenSER passa por implementar a replicação de bases de dados. O problema básico que é resolvido através deste tipo de replicação é o de manter a informação armazenada sincronizada com outras bases de dados de outros servidores. A replicação do MySQL permite configurar um ou mais servidores como réplicas de outro servidor. Assim, permite resolver problemas como [26]:

- Distribuição da Informação

  A replicação de bases de dados é útil quando se pretende manter uma cópia da informação num local geograficamente distante, como por exemplo, num centro de dados diferente;

- Balanceamento de carga

  A utilização da replicação do MySQL permite distribuir os pedidos de leitura da base de dados por vários servidores, o que é importante quando existem aplicações com leituras intensivas sobre a base de dados;

- Alta confiabilidade e *Failover*

  A replicação impede o MySQL de ser um ponto único de falha do sistema. Um bom sistema de *Failover* que involva réplicas de bases de dados ajuda a reduzir o tempo em que o sistema está em baixo.

Existem dois tipos de replicação de bases de dados MySQL, a replicação assíncrona e a replicação síncrona. Na replicação síncrona, uma transacção não termina no servidor primário até que um ou mais servidores termine a transacção [26] (Figura 4.2). Isto significa que uma réplica não pode ficar atrasada em relação ao primário e se uma transacção é abortada na réplica, então o primário também tem de a abortar. A versão utilizada para a replicação síncrona é o MySQL Cluster [6], uma versão do MySQL adaptada para ser de alta confiabilidade e disponibilidade num ambiente de computação distribuída. A replicaçao no MySQL Cluster baseia-se no mecanismo *two-phase commit* para garantir que a informação é escrita em múltiplos nodos. Uma outra solução que implementa a replicação síncrona é o MySQL Galera [5]. O que difere o MySQL Galera do MySQL Cluster é o facto de o primeiro recorrer a um protocolo de comunicação em grupo para propagar as alterações à base de dados. Teoricamente, a replicação síncrona apresenta algumas vantagens sobre a replicação assíncrona:

- Nível maior de confiabilidade

  Não ocorre a perda de informação quando um dos nodos falha e a informação replicada encontra-se sempre num estado coerente (Figura 4.3);

- Transacções em paralelo

  As transacções podem ser executadas em paralelo em todos os nodos;

- Garantias de causalidade em todo o cluster de base de dados

  Um pedido *SELECT S* seguido de uma transacção T irá devolver sempre os resultados da transacção mesmo que o pedido seja executado sobre outro nodo.

Na prática, a replicação síncrona é mais lenta do que a replicação assíncrona, uma vez que o cliente só recebe a resposta final quando a transação é efectuada em todos os nodos. Este factor leva a uma maior demora no processamento dos pedidos, o que pode resultar numa perda de performance do sistema.

Replicação assíncrona implica que, em algum momento no futuro, a réplica ficará com o estado igual ao do primário no presente, não esperando contudo que a réplica possua esse estado [18] (Figura 4.4). O tempo que demora a propagar

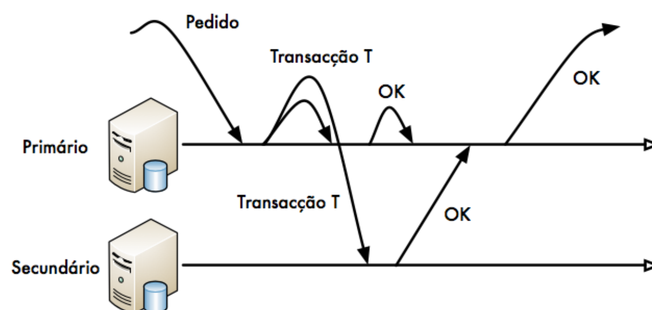



Figura 4.2: Replicação Síncrona

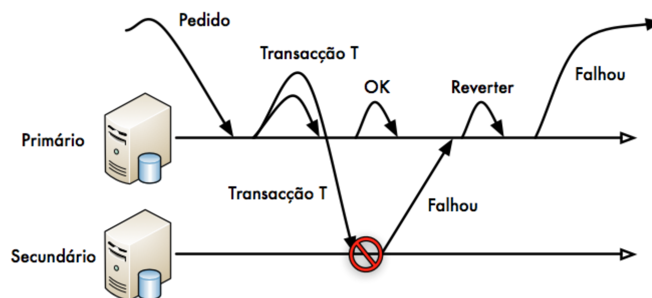



Figura 4.3: Não há perda de informação quando um nodo falha

essas alterações da base de dados depende de alguns factores tais como rede, tipo de actualização do estado, carga a que os nodos estão sujeitos aquando o pedido, entre outros [18]. O servidor primário mantém um *log* com as actualizações -*binary log*- que regista os eventos. Estes possuem informação que a réplica irá utilizar para executar a actualização da mesma maneira que o primário executou. Para além dos eventos, é também guardada alguma meta-informação que servirá para a réplica recriar o contexto da actualização, de modo a que esta devolva os mesmo resultados devolvidos pelo primário [18]. A réplica obtém estas informações conseguindo assim proceder à actualização do estado.

A replicação assíncrona promove uma maior capacidade no processamento de pedidos. No entanto, poderá ocorrer perda de informação se o nodo que executou a transacção falhe antes de a propagar para os outros nodos (Figura 4.5).

A solução proposta para o problema da replicação do OpenSER implementa a replicação síncrona de MySQL, através do MySQL Cluster. Embora este tipo de replicação introduza um declínio de performance no sistema, o facto de não haver perda de informação quando uma das máquinas falha faz compensa esse mesmo declínio. Um dos requisitos mínimos do MySQL Cluster é a existência de três máquinas. Esta terceira máquina actua como nodo de gestão dos nodos do MySQL

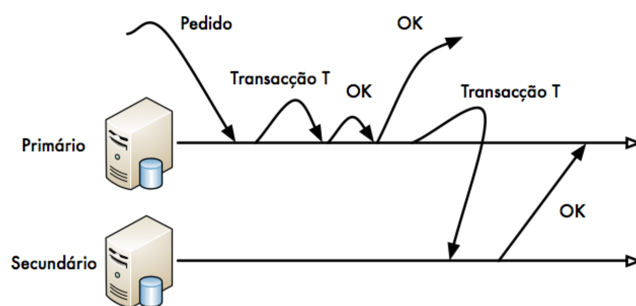



Figura 4.4: Replicação Assíncrona

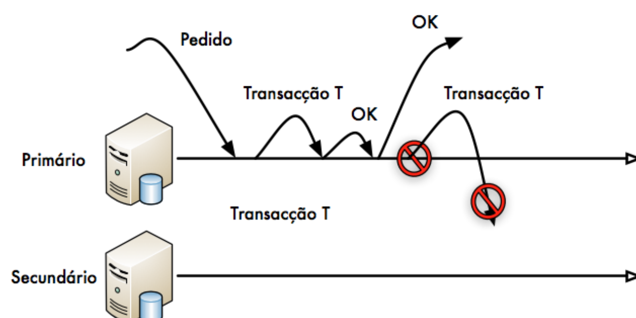



Figura 4.5: Há perda de informação quando um nodo falha

Cluster, possuindo a capacidade de desligar um dos nodos para garantir que o agrupamento de nodos MySQL continue a funcionar correctamente. Um dos problemas que poderá ocorrer durante o uso do MySQL Cluster é denominado de *Split Brain*. Este problema ocorre quando o meio de comunicação entre duas máquinas falha mas estas continuam activas. Cada nodo irá processar pedidos como se fosse o único nodo activo (Figura 4.6). Quando o meio de comunicação é restabelecido, poderão ocorrer problemas na reconciliação dos nodos MySQL. O objectivo do nodo de gestão é monitorizar os dois nodos e, caso ocorra uma situação de *Split Brain*, este tem a responsabilidade e a capacidade de escolher uma máquina e desligá-la.

Figura 4.6: Problema de *Split Brain*

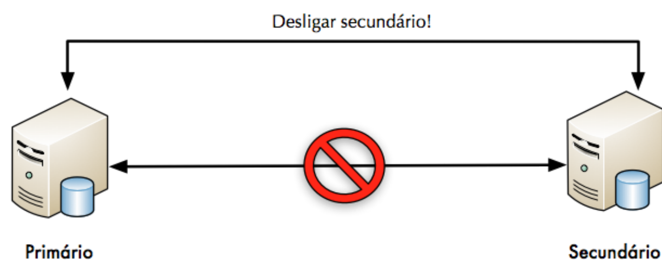



Figura 4.7: Solução para o problema de *Split Brain*

Uma solução para ultrapassar este problema passa por especificar no sistema que apenas quando existe uma maioria de máquinas a funcionar, o serviço poderá continuar a ser prestado. Um ambiente que compreende cinco máquinas apenas tolera a falha de duas. Quando ocorre a falha de uma máquina ou uma partição na rede, as que conseguem comunicar entre si verificam o número de máquinas consideradas activas. Caso este número constitua uma maioria, então estas continuam a prestação do serviço. Caso contrário, estas máquinas também param e o serviço deixa de ser disponibilizado. No ambiente de trabalho em causa existem apenas duas máquinas, tornando-se necessária outra solução para o problema do *Split Brain*. A solução para este problema passa por criar redundância nas comunicações entre as máquinas. Assim, caso um meio de comunicação entre estas falhe, o outro meio é utilizado para realmente verificar se alguma máquina falhou. Se realmente ocorreu a falha de uma máquina, então a outra continua a prestar o serviço. Caso nenhuma máquina tenha falhado, então a considerada primária envia um sinal que obriga a máquina secundária a desligar-se (Figura 4.7).

## 4.3 Problema de replicação do Asterisk e uma solução

O problema da replicação do Asterisk envolve três tipos distintos de dados a replicar. Para escrever/ler a informação necessária para o seu correcto funcionamento, o Asterisk faz uso de directórios e ficheiros, bases de dados e memória *cache*. A cada tipo de dados será necessário aplicar diferentes técnicas e ferramentas para permitir o *Failover* correcto por parte do Asterisk.

### 4.3.1 Ficheiros e Directórios do Asterisk

O Asterisk faz uso de vários directórios (Figura 4.8) para gerir vários aspectos do sistema, como por exemplo gravações de voicemail, ficheiros de configuração, e comandos de voz [17]. Todas estes directórios são criados durante a instalação do Asterisk e podem ser configurados no ficheiro *asterisk.conf*.

**Directórios**

- /etc/asterisk/

  O directório */etc/asterisk/* contém os ficheiros de configuração do Asterisk. É neste, que se encontram ficheiros de configuração como o *asterisk.conf*, *extensions.conf*, *sip.conf* e o *modules.conf*. O primeiro ficheiro de configuração não só especifica quais os vários directórios que o Asterisk irá usar para guardar não só ficheiros de configuração, ficheiros de *logs* e *scripts*, como também várias opções para a linha de comandos do Asterisk (CLI). Já ficheiro de configuração *extensions.conf* contém o *dial plan* do Asterisk. O *dial plan* é o plano principal de controlo de fluxo e de execução que informa o Asterisk como lidar e reencaminhar chamadas. É neste ficheiro que se configura o comportamento de todas as conexões que passam pelo iPBX. O *sip.conf* possui os parâmetros e configurações relativamente ao acesso de clientes SIP. Para que estes possam fazer ou receber chamadas usando o servidor Asterisk, é necessário configurar os clientes neste ficheiro. Por último, o ficheiro de configuração *modules.conf* indica que módulos iram ser carregados pelo Asterisk quando este arrancar. Muitos outros ficheiros de configuração do Asterisk encontram-se neste directório e cada um deles contém informação essencial para o Asterisk;

- /usr/lib/asterisk/modules

  Neste directório estão contidos todos os módulos que o Asterisk poderá carregar, assim como várias aplicações e codecs utilizados pelo Asterisk. Embora seja possível configurar que módulos serão carregados, alguns deles ou são essenciais para o funcionamento do Asterisk ou são dependências de outros módulos essenciais. Iniciar o Asterisk sem carregar estes módulos resultará num erro que impedirá o arranque do Asterisk.

- /var/lib/asterisk

  Este directório contém o ficheiro *astdb* e várias outras subdirectorias. O ficheiro *astdb* é uma base de dados que contém a informação local do Asterisk. Esta base de dados é uma simples implementação baseada na primeira versão da base de dados *Berkeley*. O *Asterisk Database* (AstDB) proporciona um mecanismo simples para guardar informação que poderá ser utilizada no *dial plan*. Dentro deste directório existem ainda subdirectorias que contêm ficheiros como imagens, ficheiros de som para a funcionalidade *Music on Hold*, ficheiros de som para os comandos de voz, scripts, entre outros.

- /var/spool/asterisk

  O directório */var/spool/asterisk* contém vários subdirectórios como o *meetmet/*, *outgoing/*, *voicemail*, *tmp/*, entre outros. O Asterisk monitoriza o subdirectório *outgoing/* em busca de ficheiros de chamadas. Estes, permitem a geração automática de uma chamada por parte do Asterisk e contêm informação importante como o contexto e a extensão pretendida. Tais ficheiros são úteis em situações em que se pretenda efectuar chamadas automaticamente a uma determinada hora, como por exemplo, o serviço de *wake-up call* dos hotéis. Sempre que uma mensagem de correio de voz é criada, o Asterisk guarda-a dentro do subdirectório *voicemail*. Se um utilizador pretende gravar uma conferência, o ficheiro correspondente é guardado no subdirectório *meetme/*. O *tmp/* é utilizado para guardar informação temporária. Algumas aplicações do Asterisk necessitam de um directório onde possam armazenar informação temporária. O uso deste subdirectório, impede a ocorrência de problemas de concorrência entre dois processos, processos esses que poderão desejar escrever e/ou ler um ficheiro ao mesmo tempo.

- /var/run

  É neste subdirectório que se encontra o identificador do processo (PID) do Asterisk. Este directório não apresenta qualquer problema para a solução activa/activa.

- /var/log/asterisk

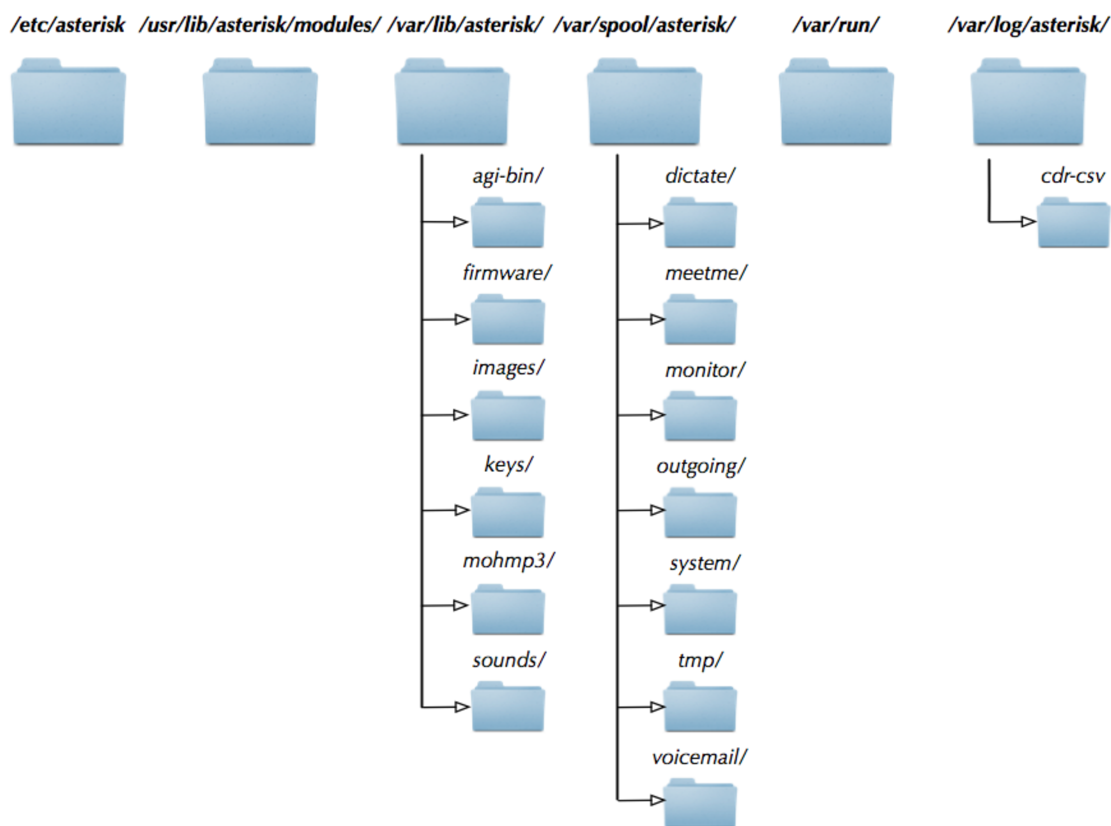



Figura 4.8: Directórios do Asterisk

Neste subdirectório é guardada, em ficheiros, toda a informação registada relacionada com erros ou avisos do Asterisk, o que por conseguinte facilita a depuração o sistema. Para além destes ficheiros, existe ainda uma subdirectoria *cdr-csv/* onde são guardados todos os *Call Detail Records* (CDRs). Estes, são criados sempre que uma chamada é efectuada ou terminada e contêm informação como a sua duração, a sua origem e destino, a data, a sua descrição, etc. Os CDRs possuem o formato *comma-separate-value* (CSV) e são depois enviados para o SABQR para efeitos de facturação, podendo ser também guardados em base de dados.

### Grupos e comportamento dos ficheiros a replicar

Apesar de existirem imensos ficheiros e directorias, é possível agrupá-los de acordo com o comportamento que cada ficheiro possui. Para tal, é necessário ter em atenção os problemas que poderão surgir aquando da replicação dos mesmos. É necessário

também considerar a forma como são acedidos e como são criados e geridos pelo Asterisk de modo a evitar problemas, nomeadamente problemas de concorrência. A criação de grupos tem como objectivo facilitar a desenvolvimento de uma solução para a replicação dos ficheiros do Asterisk.

- Configuração

  Este grupo é composto por todos os ficheiros de configuração. Com a excepção do ficheiro *asterisk.conf*, todos os outros são alterados por uma ferramenta de gestão como o FreePBX. Se, na presença de um *cluster* de iPBX dois administradores alterarem as configurações do Asterisk simultaneamente através do FreePBX, poderão eventualmente ocorrer problemas de concorrência, problemas esses que levarão a um comportamento errado por parte do Asterisk. Outro parâmetro a ter em consideração é o facto de que, sempre que o FreePBX altera os ficheiros de configuração do Asterisk, é efectuado um recarregamento das configurações por parte do Asterisk de modo a que este reflicta as alterações efectuadas às configurações. Ao realizar esta operação, o FreePBX força o Asterisk a efectuar este recarregamento das configurações a nível local. Quer isto dizer que apenas um dos nodos Asterisk (aquele que pertencer à máquina que possui o FreePBX utilizado para alterar as configurações) irá realizar a operação de recarregamento, tornando-se consciente das alterações nas configurações. Todos os outros nodos Asterisk não serão notificados de tais alterações e continuarão a execução seguindo as configurações antigas.

- Despoletadores

  Fazem parte deste grupo todos os ficheiros que actuam como despoletadores de acções por parte do Asterisk. Os ficheiros de chamada despoletam a criação de uma chamada automática por parte do Asterisk fazendo, por isso, parte deste grupo. Enquanto os ficheiros pertencentes ao grupo de Configuração necessitam de estar sempre disponíveis e replicados, os ficheiros do grupo de Despoletadores apenas necessitam de estar sempre disponíveis mas nunca replicados. Se os ficheiros deste grupo forem replicados então, por cada ficheiro de chamada, serão realizadas $n$ chamadas sendo $n$ o número de iPBX do *cluster*. Recordando o exemplo do hotel atrás mencionado, este problema

resultaria em várias chamadas para despertar o mesmo cliente. Sendo assim, a solução para este grupo passa por garantir que os ficheiros estarão sempre disponíveis mas nunca presentes em mais do que um iPBX.

- Simples

  Ficheiros como módulos, bibliotecas e algumas subdirectorias do */var/spool/asterisk* fazem parte deste grupo, devendo apresentar-se sempre disponíveis e replicados. Como estes são estáticos, não existem conflitos de concorrência quando são replicados. Existe, no entanto, um ficheiro em que poderão ocorrer problemas de concorrência: o *astdb*. É necessário garantir um mecanismo de controlo de concorrência para evitar eventuais problemas com este ficheiro. Neste grupo não são incluídos os ficheiros de correio de voz e conferências uma vez que possuem uma característica especifica que os distingue deste grupo.

- Exclusão

  Tal como os ficheiros de módulos e bibliotecas, os ficheiros deste grupo necessitam de estar sempre replicados e disponíveis. É aqui que se inserem os ficheiros de correio de voz e de conferência. Uma vez que todos os ficheiros detêm um identificador único, não ocorrem problemas de concorrência. No entanto, estes possuem uma característica única: o facto de que quando um dos nodos do Asterisk apaga um destes ficheiros, então todos os nodos necessitam também de o apagar. Caso tal não seja efectuado, os clientes iram acabar por ter mensagens de correio de voz duplicadas.

- Logs

  Este grupo é constituído pelos ficheiros que contêm os CDRs. Uma vez que o SABQR apenas requer e aceita um ficheiro com o registo dos CDRs, se todos os nodos Asterisk escreverem os CDRs para o mesmo ficheiro, é possível que se originem não só problemas de concorrência, como também problemas ao nível da performance do sistema.

### 4.3.2 Base de dados

O Asterisk, como já mencionado anteriormente, pode armazenar informação numa base de dados como por exemplo os CDRs. Informação pertencente a um utilizador

pode também ser escrita/lida a partir de uma base de dados. O *Asterisk Realtime Architecture* (ARA) é um método que permite guardar os ficheiros de configuração numa base de dados. Existem dois modos em tempo real: estático e dinâmico. O modo estático é muito semelhante ao método de ler um ficheiro de configuração. A alteração de informação estática obriga a um recarregamento por parte do Asterisk, tal como se a leitura fosse realizada a partir de um ficheiro. Já o modo dinâmico é usado para correio de voz e para guardar informação de utilizadores, informação essa que é automaticamente carregada e actualizada sempre que necessário. Neste modo, a informação é monitorizada e lida pelo Asterisk sempre que necessário, não precisando assim de a recarregar. O modo de tempo real é configurado no ficheiro *extconfig.conf*, que indica qual a informação que o Asterisk deve escrever/ler de ficheiros e qual deve escrever/ler de base de dados.

### 4.3.3 Memória *cache*

De modo a melhorar a performance, o Asterisk guarda em memória *cache* toda a informação que esteja relacionada com estados, como por exemplo, os dos dispositivos ou das extensões. Os estados das extensões são necessários para que um nodo Asterisk saiba como agir. Assim, se uma extensão estiver num estado ocupado e alguém tentar contactar a mesma, o Asterisk saberá como processar essa chamada e que acções tomar. Todas as alterações na memória *cache* relacionadas com os estados são guardadas na base de dados *astdb*. Uma vez que na solução da Wavecom S.A. este ficheiro é replicado através do DRBD, sempre que um iPBX passe do estado de espera para o estado activo, o Asterisk vai recuperar toda a informação sobre os estados ao *astdb*, assegurando deste modo que o serviço continua a ser disponibilizado correctamente. Na solução será necessário garantir que todos os nodos Asterisk têm acesso à mesma informação relativamente aos estados, com o intuíto de garantir um correcto funcionamento e *Failover* do sistema.

### 4.3.4 Replicação de ficheiros do Asterisk

A solução para o problema da sincronização de ficheiros do Asterisk passa pelo uso de um sistema de ficheiros distribuído. Um sistema de ficheiros distribuído é uma implementação distribuída do modelo clássico de *time-sharing* de um sistema de ficheiros, onde múltiplos utilizadores partilham ficheiros e recursos [29]. O objectivo destes sistemas é o de suportar o mesmo tipo de partilha quando os ficheiros estão fisicamente dispersos entre nodos de um sistema distribuído. Em vez de existir um

repositório de dados centralizado, o sistema possui múltiplos dispositivos de armazenamento independentes. Idealmente, um sistema de ficheiros distribuído deve ser visto pelos clientes como um sistema de ficheiros convencional e centralizado. Isto significa que a multiplicidade e dispersão de servidores e dispositivos de armazenamento deve ser invisível ao utilizador, impossibilitando a uma interface do cliente de um sistema de ficheiros distribuído distinguir entre ficheiros locais e remotos. Os clientes não possuem directo bloco de armazenamento subjacente, interagindo então via Internet usando um protocolo. O uso do protocolo permite a restrição do acesso ao sistema de ficheiros dependendo de listas ou capacidades de acesso.

O controlo de concorrência torna-se essencial nos sistemas de ficheiros distribuídos, dado que mais do que um utilizador terá acesso aos mesmos ficheiros. Actualizações de um ficheiro feitas por um cliente não podem interferir nem com o acesso nem com actualizações por parte de outro cliente. O controlo de concorrência poderá então estar integrado tanto no sistema de ficheiros distribuído como no protocolo.

Existem três tipos de sistemas de ficheiros distribuídos: os tolerantes a faltas, os paralelos e os tolerantes a faltas paralelos. Os sistemas tolerantes a faltas replicam a informação entre nodos para obter, não só uma elevada confiabilidade e disponibilidade, como para permitir trabalhar em modo desconectado. Já os sistemas paralelos dividem a informação entre vários servidores de modo a aumentar o desempenho do sistema. Os sistemas tolerantes a faltas paralelos resultam numa combinação dos dois anteriores e são maioritariamente utilizados em *clusters* altamente disponíveis, confiáveis e de alta performance.

Existem vários exemplos de sistemas de ficheiros distribuídos paralelos tolerantes a faltas. A solução apresentada neste projecto faz uso do GlusterFS [7]. Este é um sistema de ficheiros distribuído *open-source* capaz de lidar com milhares de clientes. É relativamente simples de configurar e utilizar, possuindo ainda funcionalidades que são importantes para a solução de replicação dos ficheiros do Asterisk, como por exemplo:

- Auto-Reparação

  O tempo de inactividade induzido por um *filesystem check* (fsck) provoca grandes problemas a nível de confiabilidade e performance do sistema. O GlusterFS não possui fsck, restaurando-se de um modo transparente e com um impacto reduzido na performance do sistema;

- Replicação Automática

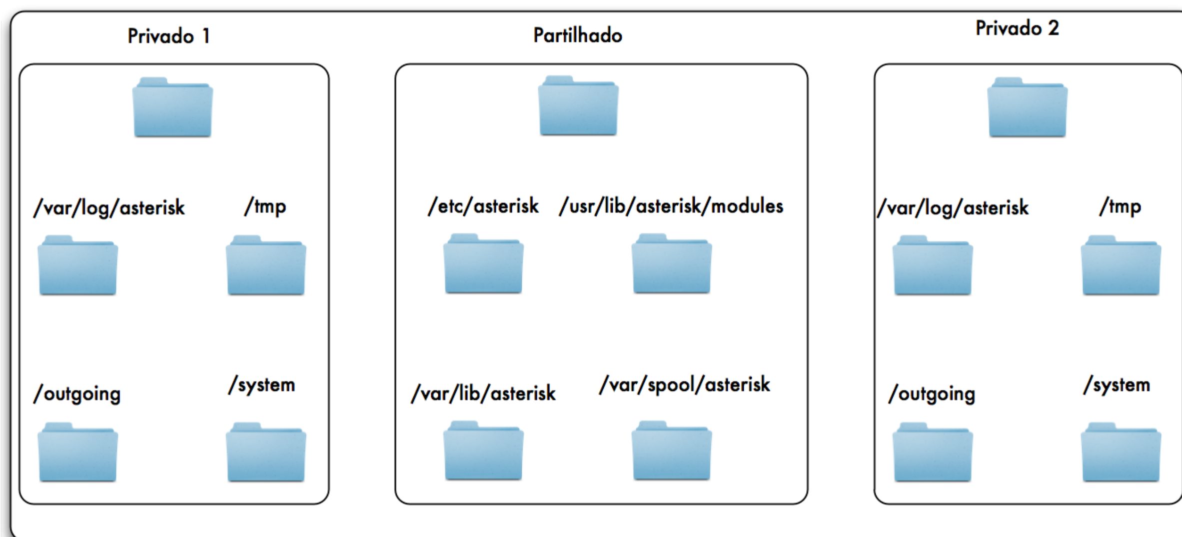



Figura 4.9: Directórios criados no sistema de ficheiros distribuído

A replicação automática de ficheiros do GlusterFS replica todo o *I/O* em tempo real, conseguindo assim ultrapassar e suportar falhas de hardware nas máquinas;

- Transporte de Informação

  O GlusterFS suporta redes baseadas em TCP/IP como *Ethernet*, *GigE*, e *10 GigE*, permitindo também o *Remote Direct Memory Access* baseado em *Infiniband*;

- Mecanismo de *Distributed Locking*

  Suporta todas as funcionalidades do mecanismos de *distributed locking* POSIX.

Uma vez que é possível configurar o Asterisk para utilizar directórios diferentes, o ficheiro *asterisk.conf* foi alterado de modo a que o Asterisk utilize directórios criados no sistema de ficheiros distribuído. Cada nodo Asterisk tem à sua disposição um directório que será apenas utilizado pelo mesmo. Para a solução foi também criado um directório que é partilhado por todos os nodos Asterisk (Figura 4.9). Recapitulando os grupos de ficheiros discutidos anteriormente:

- Configuração

Uma vez que todos os nodos Asterisk necessitam de aceder aos mesmos ficheiros de configuração torna-se necessário que, praticamente todos os ficheiros, sejam guardados no directório partilhado do sistema de ficheiros distribuído. O único ficheiro que é armazenado no sistema de ficheiros local de cada máquina é o ficheiro *asterisk.conf*, dado este ser completamente estático. Com o intuito de ultrapassar o problema do recarregamento do Asterisk, feito localmente pelo FreePBX, foi criada uma *script* baseada na ferramenta *inotify* [14] que está presente em todos os iPBX. Esta script surge com o objectivo de monitorizar o subdirectório onde se encontram os ficheiros de configuração. Sempre que a *script* detecta alterações efectuadas nestes ficheiros, ela é responsável por executar o recarregamento do Asterisk no nodo (Figura 4.10).

De modo a resolver os problemas de concorrência, fruto da gestão simultânea do sistema por dois administradores, o código do FreePBX foi alterado. O objectivo é implementar um mecanismo de *locking* recorrendo à criação de uma tabela na base de dados. Sempre que um administrador entre no sistema, a tabela sofre uma alteração de valor (passa de 0 para 1). Se na tabela estiver evidenciado o número 1, está explícito que um administrador se encontra, naquele momento, a gerir o sistema. Caso na tabela esteja evidenciado o número 0 então está indicado que ninguém, naquele momento, se encontra a administrar o sistema. Ademais e tendo por base a premissa de que dois administradores não poderão gerir o sistema em simultâneo, sempre que na tabela se encontrar o número 1 então, qualquer tentativa de acesso por parte de um segundo administrador será negada pelo FreePBX. Na eventualidade de falha do iPBX enquanto um administrador se encontrar ligado a este, o segundo iPBX irá executar uma *script* que removerá o bloqueio na base de dados (o valor passa de 1 para 0) permitindo, assim, o acesso de um novo administrador ao sistema.

- Despoletadores

Para evitar a ocorrência de múltiplas chamadas para a mesma extensão, os ficheiros deste grupo não são replicados e estão armazenados nos directórios privados no sistema de ficheiros distribuído. Quando um dos iPBX falha, o segundo é responsável por executar uma *script* que irá mover todos estes ficheiros do directório privado do iPBX que falhou para directório privado do iPBX em funcionamento. Assim, estes ficheiros estão sempre disponíveis

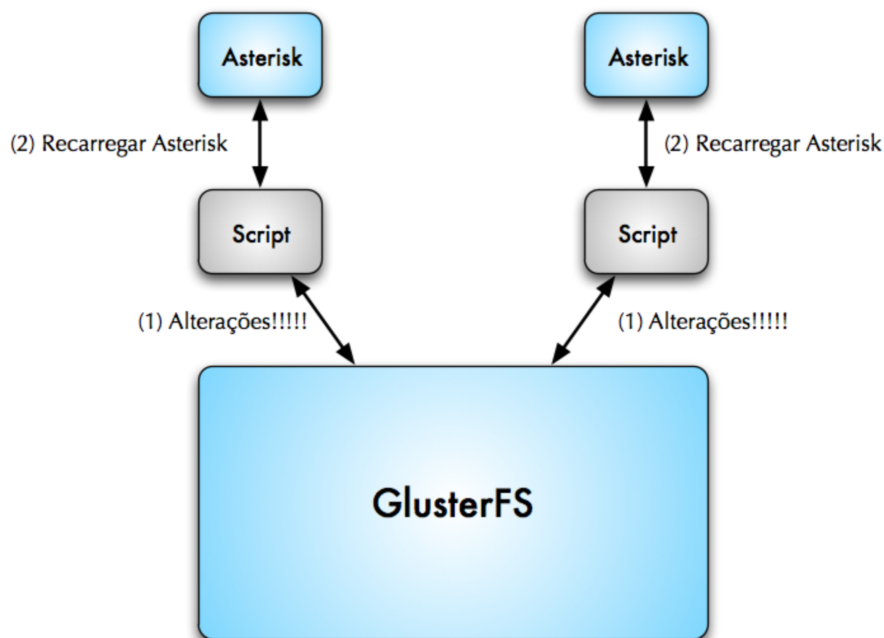



Figura 4.10: Script que detecta alterações em ficheiros e efectua recarregamento do Asterisk

e é assegurado que as chamadas automáticas serão efectuadas de um modo correcto.

- Simples

  Na solução da Wavecom S.A, o ficheiro *astdb* já é acedido e armazenado na base de dados, evitando assim problemas de concorrência. A resiliência da informação contida no *astdb* é assegurada pela replicação de base de dados já existente como solução para o problema da replicação do OpenSER. Uma vez que todos os restantes ficheiros são estáticos e iguais para todos os nodos Asterisk, estes são armazenados no directório partilhado no sistema de ficheiros distribuído.

- Exclusão

  Apesar dos ficheiros pertencentes a este grupo possuírem um comportamento e características diferentes do grupo Simples, o seu solucionamento é igual. Isto acontece pois todos os ficheiros possuem um identificador único não existindo,

assim, problemas de concorrência. Estes ficheiros são então armazenados no directório partilhado do sistema de ficheiros distribuído.

- Logs

  De modo a evitar não só problemas de concorrência como também perda de performance, todos os ficheiros de registos são armazenados nos directórios privados de cada nodo Asterisk, no sistema de ficheiros distribuído. Esta solução permite também uma melhor depuração de erros do sistema, uma vez que é facilitada a análise destes ficheiros. Sempre que o SABQR necessitar dos ficheiros que contêm os CDRs, o iPBX considerado primário agrupa todos os ficheiros num só e envia para o SABQR.

### 4.3.5 Replicação da base de dados

A replicação do MySQL, já implementada como solução para o problema da replicação do OpenSER, garante que a informação que o Asterisk armazena na base de dados é replicada, resolvendo assim o problema da replicação do Asterisk relativamente às bases de dados.

### 4.3.6 Replicação de informação armazenada na memória *cache*

A solução encontrada para o problema da replicação de informação armazenada na memória *cache* é a mais complexa de toda a arquitectura de *Failover*. O Asterisk armazena nessa memória os estados das extensões do sistema. Estes estados estão em constante modificação, o que obriga a que a replicação seja eficiente. Este problema será solucionado recorrendo à criação de um novo módulo que requer a utilização de duas ferramentas: o *Spread* e a *Asterisk Manager Interface* (AMI).

#### *Spread*

O *Spread* é uma ferramenta que disponibiliza um sistema de troca de mensagens e comunicação em grupo escalável e confiável. Esta ferramenta disponibiliza ainda vários tipos de serviços de troca de mensagens, fornecendo garantias de ordenação e confiabilidade [31]. Existem vários tipos de garantias de ordenação fornecidas pelo *Spread*:

- Sem garantia

  Não é dada nenhuma garantia de ordenação de mensagens. Se primeiro foi enviada uma mensagem *Ma* e só depois uma mensagem *Mb* usando este tipo de garantia, então *Mb* pode ser entregue antes ou depois de *Ma*;

- *First In First Out* (FIFO) do Remetente

  Todas as mensagens enviadas por um remetente serão entregues por ordem *FIFO* (primeira enviada, primeira entregue);

- Ordem Causal (Lamport)

  Todas as mensagens enviadas por todos os remetentes serão entregues por ordem coerente com a definição de ordem causal de Lamport [16];

- Ordem Total (Coerência Causal)

  Todas as mensagens enviadas por todos os remetentes são entregues exactamente pela mesma ordem a todos os destinatários.

Relativamente às garantias de confiabilidade, existem três tipos:

- Falível

  Não existem garantias relativamente à entrega da mensagem, podendo esta ser descartada ou perdida na rede. O *Spread* não recupera a mensagem caso tal aconteça;

- Confiável

  As mensagens enviadas com este tipo de garantia serão entregues de forma confiável a todos os destinatários pertencentes ao grupo para o qual foi enviada. O *Spread* recupera a mensagem de modo a ultrapassar qualquer perda na rede;

- Seguro

A mensagem apenas será entregue a um destinatário se foi igualmente entregue a todos os destinatários.

O *Spread* garante a confiabilidade de mensagens mesmo na presença de falhas de máquinas, erros nos processos e partições de rede. Uma vez que se baseia em algoritmos distribuídos, não existe um ponto central e singular de falha que possa perturbar o funcionamento do sistema. Para além de permitir o envio de mensagens entre nodos, com garantias de confiabilidade e ordenação, o *Spread* também notifica quando um processo ou máquina falha.

A ferramenta *Spread* possui ainda uma arquitectura que tem por base o modelo cliente/*daemon* [1]. Toda a comunicação física é gerida e processada por este *daemon*. Cada *daemon* controla os processos que residem na sua máquina e que participam numa comunicação em grupo. Esta informação é partilhada entre todos os *daemons* do sistema. Esta arquitectura origina um aumento de performance e possui ínumeras vantagens:

- O algoritmo de conjunto de membros apenas é invocado sempre que existe uma alteração no *daemon* de grupo. Tal não acontece quando um processo entra ou sai de um grupo. Nesta situação, o *daemon* do Spread envia uma mensagem de notificação aos outros *daemons*;

- A ordenação é mantida ao nível dos *daemons* e não ao nível dos grupos. Para sistemas com mais do que um grupo, a ordenação de mensagens é mais eficiente em termos de latência. Ademais, a ordenação de mensagens inter-grupos é trivial pois apenas é mantida a ordenação global a nível dos *daemons*;

- O controlo de fluxo é feito a nível dos *daemons*. O facto de o fluxo ser controlado a nível dos *daemons* resulta numa melhor performance em sistemas com mais do que um grupo.

#### Asterisk Manager Interface (AMI)

A AMI é uma interface que permite a programas de terceiros o controlo e a monitorização do sistema Asterisk. Esta interface possibilita que um programa considerado cliente se conecte ao Asterisk e execute comandos ou leia eventos enviados via TCP/IP. Tal permite monitorizar os estados dos dispositivos telefónicos e as extensões do sistema, permitindo também redirecionar pedidos baseados em regras criadas pelo administrador [17]. A AMI permite que os programas não só sejam notificados quando ocorre um determinado evento, como também que estes possam

proceder à alteração dos estados que o Asterisk armazena em memória *cache*. De modo a ser possível uma aplicação de terceiros comunicar através da AMI, é preciso criar uma conta no ficheiro */etc/asterisk/manager.conf*. Para tal é necessário um nome de utilizador e um segredo para autenticar a aplicação de terceiros. Neste ficheiro pode-se ainda estabalecer que endereços IP estão autorizados ou proibidos de se ligaram à AMI, estabelecendo-se ainda uma lista de permissões de operações possíveis para cada conta.

Os únicos eventos relevantes para a resolução do problema da replicação da memória *cache* do Asterisk, são os relacionados com a alteração do estado das extensões. A aplicação que se liga à AMI, recebe a notificação destes eventos sempre que o estado das extensões é alterado. Para obter estes eventos, é necessário alterar o ficheiro *extensions.conf* e adicionar *hints* ao *dial plan*, permitindo assim um mapeamento entre canais e extensões. Além destas alterações, também foi aplicado um *patch* [1] ao Asterisk que adiciona um novo comando à CLI do Asterisk. Este novo comando -*devstate change*- permite a alteração do estado de uma extensão directamente na CLI do Asterisk.

### Novo módulo

Para tornar possível a replicação dos estados das extensões, foi criado um módulo que combina estas duas ferramentas, resultando numa sincronização de estados de extensões entre todos os nodos Asterisk. Para além de permitir esta sincronização, este módulo é também responsável por activar e desactivar a interface da rede utilizada para a gestão do sistema. Um dos pré-requisitos estipulados para solução é de que apenas um dos iPBX possua esta interface activada. As mensagens entregues via *Spread* são enviadas para todos os membros, incluíndo o remetente da mensagem (*multicast*). Relativamente às garantias que o *Spread* oferece para o envio de mensagens, este módulo faz uso das garantias de ordem total e de segurança.

Existem dois modos de funcionamento deste módulo: normal e recuperação de estado. Quando o módulo arranca e se junta ao grupo criado no *Spread*, é verificada a existência de algum membro no grupo. Caso este não exista, o módulo arranca em modo normal e a interface de rede de administração do sistema é activada. Caso já exista algum membro no grupo, o módulo arranca em modo de recuperação e a interface de rede de administração é desactivada.

No modo de funcionamento normal, as ferramentas de monitorização dos iPBX

[1] https://issues.asterisk.org/view.php?id=15818

estão activadas. Sempre que o estado de uma extensão é alterado, o módulo recebe uma notificação enviada pela AMI e remete essa informação para os membros do grupo através do *Spread*. Quando o remetente recebe a mensagem que enviou, é guardado num dicionário o identificador dessa extensão e o seu novo estado. Ao receberem a mensagem, os outros membros do grupo, não só armazenam essa informação nos seus dicionários, como também procedem à alteração do estado da extensão no Asterisk através da AMI (recorrendo ao novo comando *devstate change*). Estes dicionários são necessários para a execução do algoritmo de recuperação de estado, que permitirá aos nodos Asterisk recuperarem toda a informação relacionada com os estados das extensões.

Já no modo de recuperação de estado, as ferramentas de monitorização do iPBX estão desligadas. Assim, a ferramenta de monitorização a correr no nodo em modo normal irá considerar o nodo em modo de recuperação como desligado, consequentemente não permitindo o estabelecimento de uma conexão entre os telefones IP e o iPBX em modo de recuperação. Neste modo, o nodo envia uma mensagem via *Spread* a pedir o envio dos estados das extensões ao outro nodo (Figura 4.11). A partir do momento em que o remetente recebe o seu próprio pedido, todas as futuras mensagens recebidas contendo informação sobre a alteração do estado de uma extensão são guardadas num *buffer*. Ao receber a mensagem de pedido de recuperação de estado, o destinatário envia o dicionário ao remetente. Quando o nodo em recuperação recebe o dicionário, processa-o e toda a informação é actualizada na memória *cache* do Asterisk via AMI. Caso o nodo em recuperação receba alguma mensagem contendo informação sobre a alteração do estado de uma extensão, enquanto processa o dicionário, o nodo guarda essa informação no *buffer*. Após ter processado o dicionário e actualizado a memória *cache* do Asterisk, o nodo verifica se existe informação no *buffer* e, caso exista, esta é também processada e a memória cache do Asterisk é de novo actualizada. Findo o processamento do dicionário e do *buffer*, o nodo em recuperação muda para o modo de funcionamento normal e activa tanto a interface de rede como a ferramenta de monitorização dos iPBX (Figura 4.12).

Para além do envio de mensagens, o *Spread* também notifica o módulo quando uma máquina falha. Estas notificações são de extrema importância pois, na ocorrência da falha de uma máquina, o módulo executa as *scripts* responsáveis quer pela remoção do bloqueio dos ficheiros geridos pelo FreePBX, quer pela transferência dos ficheiros do grupo Despoteladores para o directório privado do iPBX em funcionamento.

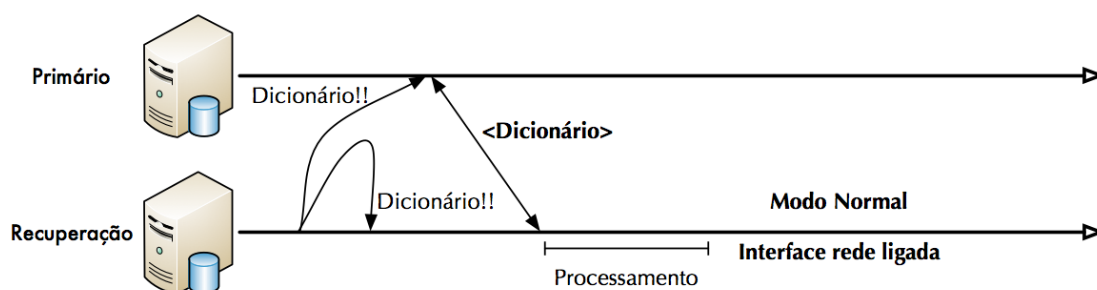



Figura 4.11: Pedido e processamento do dicionário

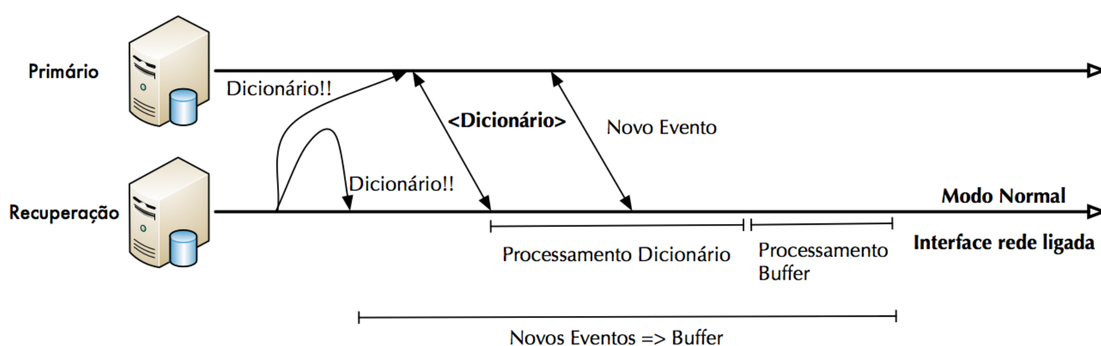



Figura 4.12: Processamento do dicionário e do Buffer

Este módulo poderia recorrer ao envio de mensagens via TCP em vez de recorrer ao *Spread*. No entanto, este último fornece garantias que são imprescindíveis para o bom funcionamento da replicação dos estados das extensões. Um outro detalhe importante a ter em consideração é a ordem pela qual os componentes de software são iniciados. Quando uma máquina é reiniciada e está no modo de recuperação, é necessário assegurar que o módulo criado é o último a arrancar. Assim, é garantido que a ferramenta de monitorização do iPBX apenas será activada após a execução do algoritmo de recuperação do estado do Asterisk.

## 4.4 Resumo

Este capítulo fornece a solução para o problema do *Failover* do iPBX. Foram estudados todos os componentes de software e, para cada um deles, foi realizada uma análise sobre os detalhes e as características a ponderar na criação da solução. A solução final assenta não só na configuração de ferramentas (como a replicação

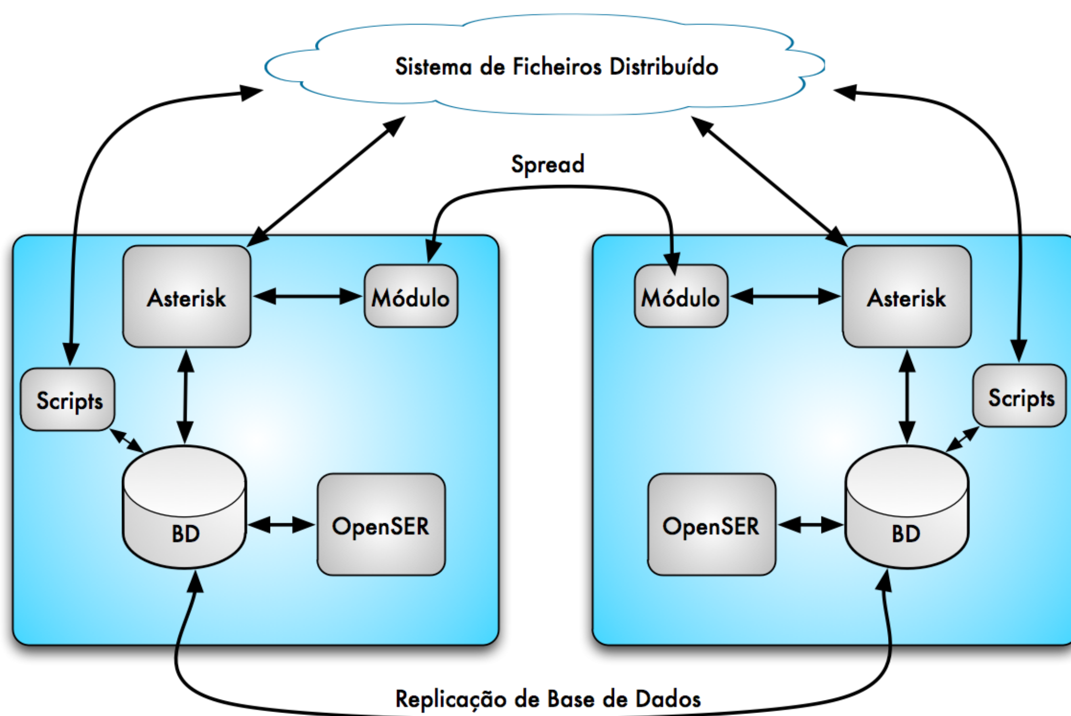



Figura 4.13: Solução Final do Failover do iPBX

do MySQL e o sistema de ficheiros distribuído), como também na criação de um módulo e de *scripts* que garantam a replicação de informação crucial para o Asterisk, impedindo consequentemente a ocorrência de problemas que poderiam afectar severamente o funcionamento correcto do sistema (Figura 4.13).

# Capítulo 5

# Arquitectura Hierarquizável da Solução

Um sistema de telefonia de VoIP deve ser capaz de suportar um número elevado de chamadas em simultâneo sem a ocorrência de uma perda considerável de performance. Na solução da Wavecom S.A. existem apenas dois níveis de hierarquia: o nível correspondente ao iPBX e o nível correspondente aos telefones IP (Figura 5.1). Todos estes telefones registam-se e ficam sob a alçada de um único iPBX, obrigando a que todo o fluxo de dados passe por este iPBX.

Nesta capítulo irá ser apresentada e descrita uma arquitectura que visa aumentar a escalabilidade do sistema. Na arquitectura em questão, todos os pressupostos relacionados com *Failover* e tolerância a faltas são postos de lado. Na secção seguinte irá ser descrita uma possível arquitectura para aumentar a escalabilidade do sistema, bem como as implicações que esta arquitectura traz tanto para o sistema como para os componentes de software que compõe o iPBX.

## 5.1 Arquitectura de Hierarquização

Uma das possíveis soluções para o aumento da escalabilidade do sistema passa por adicionar um novo nível de hierarquia ao sistema. Sob a alçada do iPBX primário, existem não só telefones IP como também existe um iPBX secundário. Este tipo de arquitectura é relevante e útil em vários cenários, como é o exemplo de uma universidade. As comunicações internas de uma universidade compreendem ligações entre diferentes departamentos (inter-departamentos) assim como dentro dos próprios departamentos (intra-departamentos). Assumindo que exista apenas um iPBX responsável por estes dois tipos de ligações, e que este esteja aleatoriamente localizado,

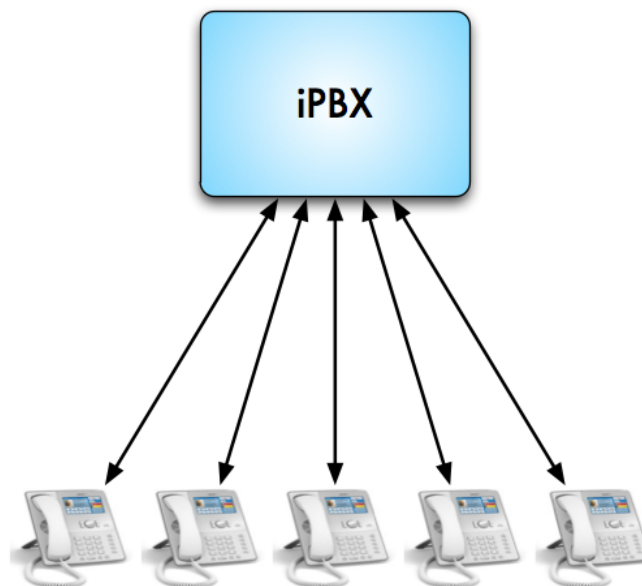



Figura 5.1: Hierarquia do Sistema de VoIP da Wavecom

estas não irão ter outra alternativa senão passar por este para efectuar chamadas. Isto coloca um problema pois, no caso de este iPBX falhar, é perdida a capacidade de efectuar chamadas inter e intra-departamento. Uma alternativa para tentar solucionar esta falha passa não só por aliviar a carga de chamadas no iPBX responsável, como também de tornar as chamadas intra-departamento não dependentes do iPBX principal. É também expectável que uma arquitectura hierarquizável produza o menor impacto possível, quer ao nível da configuração do iPBX primário como ao nível dos telefones IP.

Um iPBX secundário difere do primário ao nível de componentes de software. Para que o iPBX se torne mais leve, apenas são necessários os componentes essenciais para que se possam efectuar chamadas: o OpenSER, o Asterisk, uma base de dados MySQL e o FreeRADIUS. Uma vez que o componente FreePBX apenas tem como objectivo a gestão do sistema, não é necessário colocá-lo no iPBX secundário. No entanto, esta decisão implica resolver problemas relacionados com a modificação dos ficheiros presentes no iPBX primário.

Os telefones IP pertencentes ao novo nível da hierarquia registam-se em ambos os iPBX, enquanto que os restantes apenas se registam no iPBX primário (Figura 5.2). Isto resulta num conhecimento, por parte do iPBX primário, das localizações de todos os telefones IP, garantindo assim uma maior transparência ao nível deste iPBX. Se este não possuir as localizações de todos os telefones, sempre que se adicionasse

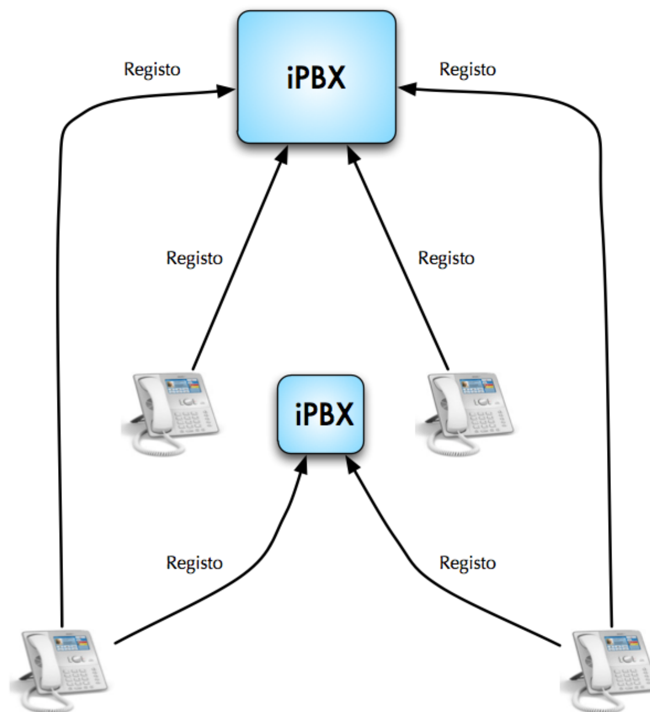



Figura 5.2: Registo dos telefones IP de acordo com o seu nível na hierarquia

um novo iPBX secundário, seria necessário alterar a configuração dos componentes de software do iPBX primário. Relativamente ao registo em vários iPBX, existem telefones que suportam esta funcionalidade e, para os que não suportam efectuar duplo registo, então será necessário recorrer à replicação de SIP relativamente aos pedidos de registo. Esta replicação é realizada ao nível do OpenSER recorrendo à função *t_replicate("other-proxy","port")* . Assim, sempre que é efectuado um pedido de registo num dos nodos OpenSER, este envia esse mesmo pedido para os outros nodos OpenSER. Existem três tipos de chamadas: chamadas efectuadas no mesmo nível, chamadas entre níveis diferentes (Figura 5.3) e chamadas para o exterior (Figura 5.4).

- Chamadas efectuadas no mesmo nível

  Uma vez que o iPBX responsável por processar este tipo de chamadas possui a localização do telefone que se pretende contactar, sendo processadas pelo iPBX de um modo normal.

- Chamadas efectuadas entre níveis diferentes

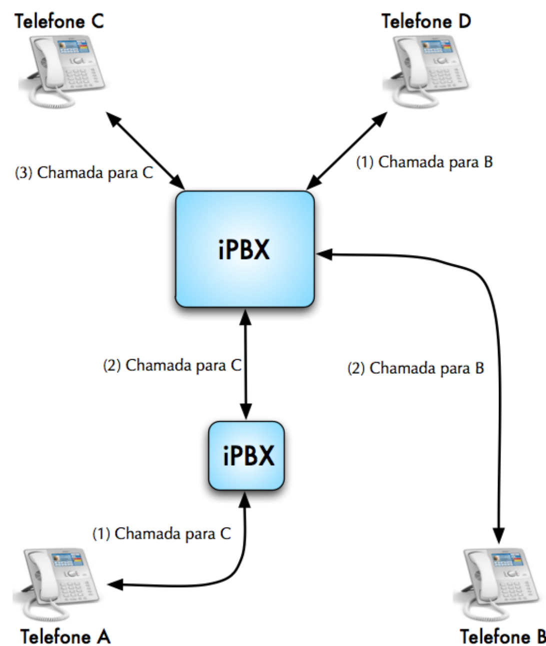



Figura 5.3: Exemplos de chamadas efectuadas entre níveis diferentes

Neste tipo de chamadas existem dois cenários possíveis, nos quais ou o iPBX responsável pelo processamento da chamada é o primário ou é o secundário. Caso o iPBX responsável seja o secundário, então ao receber um pedido para efectuar uma chamada, o iPBX secundário vai tentar localizar o destinatário. Uma vez que o iPBX secundário não possui o registo de localização do telefone de destino, encaminha o pedido para o iPBX primário. Como este já possui a localização do telefone destino, a chamada é então processada normalmente, sendo que o fluxo de dados passa entre os dois iPBXs.

Caso o iPBX responsável por processar a chamada seja o iPBX primário, então a chamada é processada de um modo normal visto este possuir os registos de localizações de todos os telefones (Figura 5.3).

- Chamadas para o exterior

Apenas o iPBX primário pode comunicar e encaminhar chamadas para a SBC. Assim, se o iPBX secundário for responsável por processar uma chamada, este encaminha-a para o iPBX primário, que por conseguinte a encaminhará para a SBC. Caso o iPBX responsável por processar uma chamada for o iPBX primário, então essa será imediatamente encaminhada para a SBC (Figura 5.4).

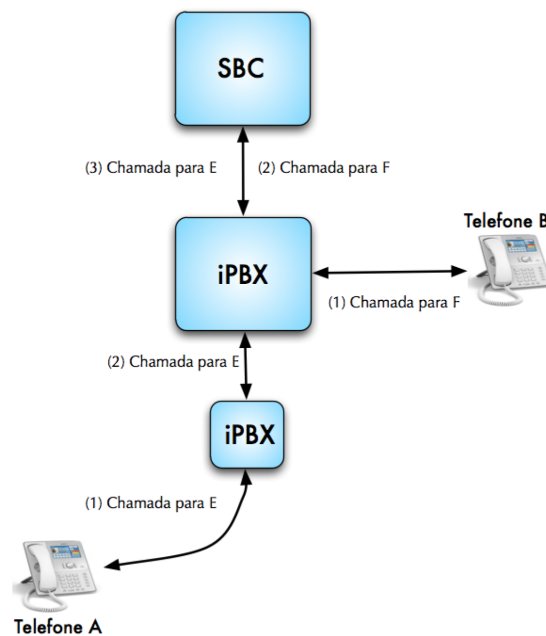



Figura 5.4: Exemplos de chamadas efectuadas para o exterior

### 5.1.1 Hierarquia no OpenSER

De modo a permitir que o iPBX secundário encaminhe a chamada para o iPBX primário, é primeiro necessário aplicar algumas alterações ao ficheiro de configuração do OpenSER presente no iPBX secundário. Existem duas possíveis soluções para este problema: a primeira passa por obrigar a que as extensões dos telefones IP que se registam no iPBX secundário sejam iniciadas por um determinado número. A segunda, processa-se quando o iPBX secundário não encontra um registo de localização, sendo a chamada é encaminhada para o iPBX primário. Relativamente à primeira solução, e considerando que as extensões dos telefones IP que se registam no iPBX secundário começam pelo número 5, acrescenta-se a seguinte informação ao ficheiro *openser.cfg* do iPBX secundário:

```
if(!(uri= "sip:5[0-9]+@")) {
rewritehostport("openserB_hostname");
t_relay();
exit;
}
```

A primeira linha verifica se o destino da chamada pertence ao mesmo nível da

hierarquia. A função *rewritehostport("hostname")* reescreve o campo *host* e o campo *port* de um *Request URI*, indicando assim o endereço para onde o pedido deve ser encaminhado. A função *t_relay* faz o encaminhamento do pedido de acordo com o *Request URI*.

Relativamente à segunda solução, a informação a adicionar ao ficheiro *openser.cfg* do iPBX secundário é semelhante aquela já apresentada, apenas alterando a primeira linha:

```
if(!lookup("location"))
rewritehostport("openserB_hostname");
t_relay();
exit;
}
```

A função *lookup("location")* tenta obter o registo de localização de uma extensão e, caso esse não seja encontrado, o pedido é encaminhado para outro nodo OpenSER. Foi esta a solução adoptada nesta hierarquia uma vez que não implica a atribuição fixa de números das extensões dos telefones IP registados no iPBX secundário.

### 5.1.2 Hierarquia no Asterisk

Para que o fluxo de dados passe pelos nodos do Asterisk, alterações aos seus ficheiros de configuração, nomeadamente o ficheiro *sip.conf* e o ficheiro *extensions.conf*, são necessárias. Um exemplo das alterações necessárias a efectuar em cada ficheiro é de seguida apresentado.

Existem dois nodos Asterisk, o nodo AsteriskA e o nodo AsteriskB. No ficheiro *sip.conf* do nodo AsteriskA acrescenta-se a seguinte informação:

```
[general]
register => AsteriskA:welcome@192.168.2.202/AsteriskB
[AsteriskB]
type=friend
secret=welcome
context=AsteriskB_incoming
host=dynamic
disallow=all
```

*allow=ulaw*

A linha *register* permite ao nodo AsteriskA registar-se no AsteriskB, de modo a que o primeiro possa encaminhar chamadas para o nodo AsteriskB. As restantes linhas fazem parte do bloco de autorização utilizado para controlar chamadas recebidas e efectuadas pelo nodo AsteriskB. O tipo *friend* permite receber e realizar chamadas, o campo *secret* é a palavra-passe utilizada no processo de autenticação dos nodos, o campo *context* indica onde as chamadas recebidas são processadas no *dialplan* (ficheiro *extensions.conf*), o campo *host* com o valor *dynamic* indica que o nodo AsteriskB irá registar-se no nodo AsteriskA, indicando o seu endereço IP para que se possam efectuar chamadas. Os últimos dois campos identificam que *codecs* podem ser usados. O ficheiro *sip.conf* no nodo AsteriskB é semelhante ao exposto para o nodo AsteriskA, apenas trocando os nomes dos nodos Asterisk.

Relativamente ao ficheiro *extensions.conf*, é acrescentada informação ao *dialplan* para processar tanto as chamadas internas como as chamadas entre nodos Asterisk.

Ao não incluir o FreePBX no iPBX secundário, poderá ocorrer um problema de coerência dos ficheiros de configuração, dado que um administrador do sistema pode alterar os ficheiros de configuração do Asterisk no iPBX primário mas não no secundário. A criação de uma solução que implicasse a replicação de ficheiros de um modo síncrono (recorrendo, por exemplo, ao uso de um sistema de ficheiros distribuídos), derrotaria o propósito da criação de uma solução com o objectivo de aumentar a escalabilidade do sistema. A solução passaria a utilizar uma ferramenta que permitisse a todos os nodos Asterisk, possuir os mesmo ficheiros de configuração de um modo assíncrono. Uma *script* é inicializada no iPBX primário que monitoriza os ficheiros de configuração do Asterisk. Quando são detectadas alterações nestes, é utilizada uma ferramenta como o *rsync* [1] para garantir que todos os nodos Asterisk possuem os mesmo ficheiros de configuração do Asterisk. No lado do iPBX secundário, é inicializada uma *script* que monitoriza os ficheiros de configuração. Quando são detectadas diferenças, é efectuado um recarregamento do Asterisk. No entanto, e uma vez que as alterações nos ficheiros de configuração acima descritas implicam a existência de ficheiros diferentes para cada nodo Asterisk, estas alterações são realizadas num ficheiro à parte que é importado nos ficheiro principais *sip.conf* e *extensions.conf*. Estes ficheiros adicionais não são monitorizados pela script nem são enviados através do *rsync* para todos os nodos.

[1] http://samba.anu.edu.au/rsync/

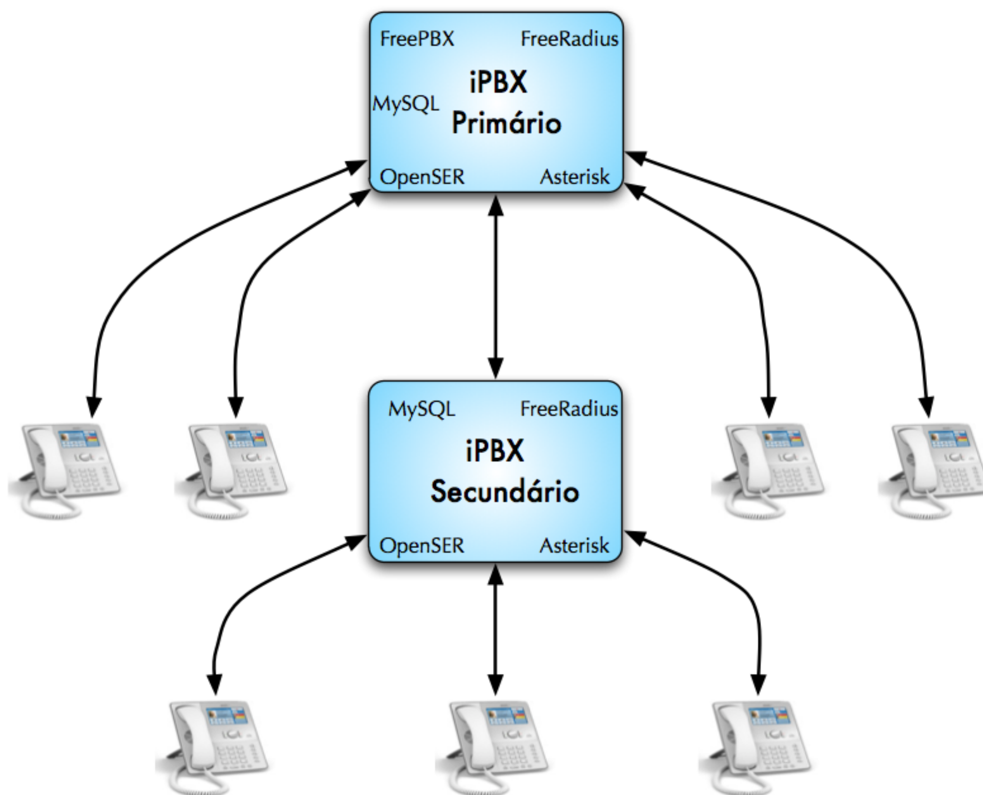



Figura 5.5: Níveis da Arquitectura Hierarquizável

A desvantagem que a replicação assíncrona de ficheiros apresenta face à síncrona prende-se com o facto de, enquanto na replicação síncrona a alteração de um ficheiro é imediatamente reflectida no iPBX secundário, na replicação assíncrona tal não acontece. Isto impossibilita aos telefones registados no iPBX secundário de efectuarem chamadas para uma nova extensão criada. No entanto, este revés é menos prejudicial para uma solução que tem por objectivo o aumento da escalabilidade do sistema.

## 5.2 Resumo

Neste capítulo é apresentada uma arquitectura hierarquizável que visa o aumento da escalabilidade do sistema (Figura 5.5). A adição de um iPBX secundário, sob a alçada de um iPBX primário, resulta num menor congestionamento deste, permitindo uma melhoria da sua performance. Este novo nível na hierarquia impede, também, que a falha do iPBX primário resulte numa impossibilidade de comunicação entre telefones do mesmo nível.

# Capítulo 6

# Conclusão

## 6.1 Conclusão e Trabalho Futuro

A proliferação dos sistemas telefónicos VoIP apresenta-se, no panorama actual das comunicações de voz, mais vantajoso e barato quando comparados com sistemas de telefonia tradicionais. Os conceitos de confiabilidade e de escalabilidade têm de estar constantemente presentes em qualquer sistema de telefonia VoIP, sob o risco de estes não vingarem no mercado das telecomunicações.

Este trabalho foi desenvolvido tendo por base o estudo e a análise de um componente de software, o iPBX por este ser o componente sujeito a um maior fluxo de informação e por desempenhar um papel crucial na solução VoIP de uma empresa na área das comunicações, a Wavecom S.A.. Embora a solução da Wavecom S.A já ofereça tolerância a faltas, este sistema funciona em modo activo/passivo, no qual apenas uma das máquinas disponibiliza o serviço de telefonia (modo activo), enquanto que a outra máquina permanece num estado de espera (modo passivo). Esta última funciona como cópia de segurança e, na ocorrência de uma falha na máquina primária, esta passa de um estado de espera para o estado activo, permitindo a continuação serviço. Quando na presença de um sistema com um baixo número de máquinas disponíveis, o uso de uma solução que funcione apenas em modo activo/passivo representa um desperdício de recursos extremamente úteis.

A solução adoptada para dotar a arquitectura da Wavecom S.A. com um serviço de *Failover* no modo activo/activo baseou-se, não só num conjunto de ferramentas de software com capacidade para replicar diferentes tipos de informação, como também na criação de um módulo que vise assegurar a correcta replicação dos esta-

dos das extensões do Asterisk. Uma vez que um iPBX possui diversas ferramentas de software foram, numa fase inicial, analisados os detalhes do seu funcionamento que poderiam causar problemas numa solução em modo activo/activo. Após esta fase inicial, verificou-se que o conjunto de ferramentas de software requer a replicação de três tipos de informação: informação guardada em ficheiros, em bases de dados e em memória. Para executar estes três tipos de replicação, recorreu-se ao uso de ferramentas como o *MySQL Cluster* (replicação de bases de dados) e o *GlusterFS* (replicação de ficheiros). De modo a resolver o problema da replicação da informação guardada em memória, foi criado um módulo que recorre ao uso do *Spread* e da *AMI*. Este módulo é ainda responsável por executar *scripts* que resolvem pequenos detalhes como por exemplo, o problema de mover ficheiros quando um dos iPBX falha.

A garantia de transparência da solução relativamente aos telefones IP foi possível graças à implementação de um *LVS* que coloca toda a inteligência de *Failover* no lado dos servidores. Para além de garantir a transparência do sistema, o *LVS* também efectua um balanceamento de carga para que ambos os iPBX possam processar pedidos.

O uso das ferramentas supramencionadas e do módulo criado resultou numa solução capaz de funcionar no modo activo/activo possuindo não só garantias de confiabilidade, como também garantias de que a solução é compatível com qualquer telefone IP que suporte o protocolo SIP.

Relativamente ao aumento da escalabilidade do sistema, foi tomada a decisão de não recorrer à optimização de qualquer componente de software, mas sim de apresentar uma arquitectura hierarquizável que permita a diminuição do fluxo de informação a que um iPBX primário poderá estar sujeito. Esta arquitectura implica a existência de uma replicação de ficheiros, uma vez que a gestão do sistema é efectuada apenas no iPBX primário. O uso de sincronismo nesta arquitectura derrotaria por completo o objectivo da mesma já que implicaria uma perda de performance por parte dos iPBX. Para este efeito foi utilizada uma ferramenta que permite a replicação de ficheiros de um modo assíncrono. Um dos objectivos considerados na criação desta solução foi o da transparência da arquitectura ao nível do iPBX primário. Ao ser efectuado um duplo registo por parte dos telefones IP que se registam no iPBX secundário, evita-se proceder à alteração dos ficheiros do OpenSER do iPBX primário sempre que se adicionar um novo iPBX secundário.

A solução apresentada nesta dissertação para resolver o problema de confiabilidade da solução da Wavecom S.A. assenta em metodologias já comprovadas, como é a replicação de base de dados e a replicação de ficheiros recorrendo a sistemas

de ficheiros distribuídos. Estes métodos, vulgarmente utilizados em muitas outras soluções de replicação, apresentam-se credíveis, cumprindo todos os objectivos a que se propõem. Ademais, estes métodos não interferem com o funcionamento de ferramentas de software como o Asterisk e o OpenSER. Relativamente ao módulo criado para replicar os estados das extensões do Asterisk, foram executados testes que validam o seu correcto funcionamento. No entanto, este módulo é passível de sofrer actualizações com o intuito de aumentar o seu desempenho enquanto ferramenta crucial nesta solução.

No que concerne a solução para resolver o problema da escalabilidade da solução da Wavecom S.A, foram efectuados testes preliminares, que corroboram o funcionamento da mesma. No entanto e uma vez que a implementação desta solução num sistema real para testes é ainda algo em processo, não foi possível executar os mesmo na sua totalidade o que, por conseguinte, se traduz numa ausência de resultados finais. Assim, é expectável a prossecução dos testes de modo a verificar o aumento real de performance ao implementar a solução aqui apresentada para o problema da escalabilidade.

Como qualquer *work in progress*, esta solução está sujeita a actualizações quer devido à implementação de novas funcionalidades quer pelo upgrade das mesmas como resposta a um mercado cada vez mais vanguardista, competitivo e em constante mudança.

# Bibliografia

[1] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The spread toolkit: Architecture and performance, 2004.

[2] R. Balasubramanian, I. Rhee, and J. Kang. A scalable architecture for sip infrastructure using content addressable networks. pages 1314–1318, 2005.

[3] G. Camarillo. *SIP Demystified*. McGraw-Hill, 2009.

[4] N. Free and N. Borenstein. Multipurpose internet mail extensions (mime) part one: Format of internet message bodies, 1996.

[5] MySQL Galera. http://www.codership.com/products/mysql_galera.

[6] MySQL Galera. http://www.mysql.com/products/cluster.

[7] GlusterFS. http://www.gluster.com/products/gluster-platform.php.

[8] F Gonçalves. *Building Telephony Services with OpenSER*. PACKT, 2008.

[9] A. Gulbrandsen, P. Vixie, and L. Esibov. A dns rr for specifying the location of services (dns srv), 2000.

[10] F. Haas, P Reisner, and L. Ellenberg. The drbd user's guide, 2009.

[11] M. Handley and V. Jacobson. Sdp: Session description protocol, 1998.

[12] M Handley and E Schooler. Session invitation protocol, 1996.

[13] Heartbeat. http://www.linux-ha.org/wiki/heartbeat.

[14] Inotify. http://linux.die.net/man/7/inotify.

[15] J. Klensin. Simple mail transfer protocol, 2001.

[16] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[17] J. Meggelen, L. Madsen, and J. Smith. *Asterisk: The Future of Telephony.* O'Reilly, 2005.

[18] D. Pachev. *Understanding MySQL Internals.* O'Reilly, 2007.

[19] Radvision. Sip protocol overview, 2001.

[20] C Rigney. Radius accounting, 2000.

[21] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote authentication dial in user service (radius), 2000.

[22] J. Rosenberg and Schulzrinne. An offer/answer model with the session description protocol (sdp), 2002.

[23] J. Rosenberg and H. Schulzrinne. Session initiation protocol (sip): Locating sip servers, 2002.

[24] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol, 2002.

[25] H Schulzrinne. Simple conference invitation protocol, 1996.

[26] B. Schwartz, P. Zaitsev, V. Tkachenko, J. Zawodny, A. Lentz, and D. Balling. *High Performance MySQL.* O'Reilly, 2008.

[27] J. Sermersheim. Lightweight directory access protocol (ldap): The protocol, 2006.

[28] Linux Virtual Server. http://www.linuxvirtualserver.org.

[29] A. Silberschatz, G. Gagne, and Peter Galvin. *Operating System Concepts (7th Edition).* John Wiley & Sons.Inc, 2005.

[30] K. Singh and H. Schulzrinne. Failover, load sharing and server architecture in sip telephony. *Comput. Commun.*, 30(5):927–942, 2007.

[31] J Stanton. A users guide to spread, 2002.

[32] Wavecom. http://www.wavecom.pt.

# Apêndice A

# Módulo Criado

Segue-se o código fonte do módulo desenvolvido:

## Main.java

```
package spread;

import java.io.IOException;
import org.asteriskjava.manager.AuthenticationFailedException;
import org.asteriskjava.manager.ManagerConnection;
import org.asteriskjava.manager.ManagerConnectionFactory;
import org.asteriskjava.manager.TimeoutException;

public class Main {

    public static void main(String[] args) throws
        SpreadException, InterruptedException,
        IllegalStateException, IOException,
        AuthenticationFailedException, TimeoutException {
         SpreadConnection connection = new SpreadConnection();
         connection.connect(null, 5000, args[0], true, true);

         ManagerConnection managerConnection;
         ManagerConnectionFactory factory = new
             ManagerConnectionFactory(args[1],"admin","wavecom");
         managerConnection = factory.createManagerConnection();
```

```
        SpreadGroup group = new SpreadGroup();
        group.join(connection, "SD");


        SpreadMessage message = new SpreadMessage();

        Spread spread = new Spread(connection,managerConnection,
            group,message,args[0]);
        spread.run();
    }
```

## Spread.java

```
package spread;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.asteriskjava.manager.AuthenticationFailedException;
import org.asteriskjava.manager.ManagerConnection;
import org.asteriskjava.manager.ManagerConnectionFactory;
import org.asteriskjava.manager.ManagerEventListener;
import org.asteriskjava.manager.TimeoutException;
import org.asteriskjava.manager.action.CommandAction;
import org.asteriskjava.manager.action.ExtensionStateAction;
import org.asteriskjava.manager.action.OriginateAction;
import org.asteriskjava.manager.action.StatusAction;
import org.asteriskjava.manager.event.ExtensionStatusEvent;
import org.asteriskjava.manager.event.ManagerEvent;
```

```
import org.asteriskjava.manager.response.CommandResponse;
import org.asteriskjava.manager.response.ManagerResponse;


public class Spread implements AdvancedMessageListener,
    ManagerEventListener {

     private ManagerConnection managerConnection;
     private SpreadConnection connection;
     private SpreadGroup group;
     private SpreadMessage message;
     private HashMap<String,String> dictionary;
     private HashMap<String,String> copy;
     private HashMap<String,String> bufferedMessages;
     private String myName;

     private int recoveryMode;
     private Iterator it;
     private int startBuffering; //adicionado depois
     private int hasInitiated = 0;

public Spread(SpreadConnection connection,ManagerConnection
    managerConnection,SpreadGroup group,SpreadMessage message,
    String myName){

   this.connection = connection;
   this.managerConnection= managerConnection;
   this.group = group;
   this.message = message;
   this.myName = myName;

   bufferedMessages = new HashMap<String,String>();

   this.dictionary = new HashMap<String,String>();
}
```

```
public void run() throws SpreadException, InterruptedException,
    IllegalStateException, IOException,
    AuthenticationFailedException, TimeoutException{


  managerConnection.addEventListener(this);
  managerConnection.login();
  managerConnection.sendAction(new StatusAction());

  connection.add(this);

  while(true){
    Thread.sleep(10000);
  }
}

public void regularMessageReceived(SpreadMessage sm) {
  try{

    Object received = sm.getObject();

    if(received.getClass().toString().contains("String")){
      String msgReceived = (String) received;

      if(sm.getSender().toString().contains(myName)){
        if(msgReceived.contains("recovery")){

          startBuffering = 1;

        }

        if(msgReceived.contains("/")){

          this.insert(msgReceived,0);

        }
      }
```

```
else{
  if(msgReceived.contains("recovery")){

    try {
      SpreadMessage send = new SpreadMessage();
      copy = dictionary;
      send.setObject(copy);
      send.addGroup(group);
      send.setSafe();
      this.connection.multicast(send);



    } catch (SpreadException e2) {
      Logger.getLogger(Spread.class.getName()).log(Level
          .SEVERE, null, e2);
    }

  }

  if(msgReceived.contains("/")){

    if(recoveryMode == 1){
      if(startBuffering == 1){
        this.updateBuffer(msgReceived);
      }

    }

    else{
      this.insert(msgReceived,1);
    }


  }
 }

}
```

```
else{

  if(sm.getSender().toString().contains(myName)){
  }

  else{
    try {
      dictionary = (HashMap<String, String>) sm.getObject
          ();
      } catch (SpreadException ex) {
        Logger.getLogger(Spread.class.getName()).log(Level
            .SEVERE, null, ex);
      }

      it = dictionary.entrySet().iterator();
      while(it.hasNext()){
        Map.Entry pairs = (Map.Entry) it.next();
        int saux2 = Integer.parseInt(pairs.getValue().
            toString());
        String clicmd;
        switch(saux2){
          case 0: clicmd = "NOT_INSUSE";break;
          case 1: clicmd = "INUSE";break;
          case 2: clicmd = "BUSY";break;
          case 4: clicmd = "UNAVAILABLE";break;
          case 8: clicmd = "RINGING";break;
          case 16: clicmd = "ONHOLD";break;
          default: clicmd = "UNKNOWN";break;


        }

        CommandAction ca = new CommandAction("devstate
            change Custom:"+new String(pairs.getKey().
            toString())+" "+ clicmd);
        try {
```

```
    CommandResponse cr = (CommandResponse)
        managerConnection.sendAction(ca);
    } catch (Exception ex) {
      Logger.getLogger(Spread.class.getName()).log(
          Level.SEVERE, null, ex);
    }

}
if(bufferedMessages.isEmpty()){
  recoveryMode=0;
  String cmd = "/etc/init.d/heartbeat start";
  Process child = Runtime.getRuntime().exec(cmd);
}

else{
  it = bufferedMessages.entrySet().iterator();
  while(it.hasNext()){
    Map.Entry pairs = (Map.Entry) it.next();
    int saux2 = Integer.parseInt(pairs.getValue().
        toString());
    String clicmd;
    switch(saux2){
      case 0: clicmd = "NOT_INSUSE";break;
      case 1: clicmd = "INUSE";break;
      case 2: clicmd = "BUSY";break;
      case 4: clicmd = "UNAVAILABLE";break;
      case 8: clicmd = "RINGING";break;
      case 16: clicmd = "ONHOLD";break;
      default: clicmd = "UNKNOWN";break;



    }

    CommandAction ca = new CommandAction("devstate
        change Custom:"+new String(pairs.getKey().
        toString()+" "+ clicmd);
    try {
```

```
                    CommandResponse cr = (CommandResponse)
                        managerConnection.sendAction(ca);
                    } catch (Exception ex) {
                      Logger.getLogger(Spread.class.getName()).
                          log(Level.SEVERE, null, ex);
                    }

                  }
                }
              }




            }
          }
          catch(Exception e){
            System.out.println(e.toString());
          }

        }


public void membershipMessageReceived(SpreadMessage sm) {

  if(sm.getMembershipInfo().isCausedByDisconnect() || sm.
      getMembershipInfo().isCausedByLeave()){

    try {

      System.out.println("alguem saiu");
      String cmd = "ifconfig eth1 up";
      Process child = Runtime.getRuntime().exec(cmd);

      cmd = "sh /usr/src/move.sh";
      child = Runtime.getRuntime().exec(cmd);

      cmd = "sh /usr/src/unlock.sh";
      child = Runtime.getRuntime().exec(cmd);
```

```
    } catch (Exception ex) {
      Logger.getLogger(Spread.class.getName()).log(Level.
          SEVERE, null, ex);
    }


  }
  else{
    if(hasInitiated == 1){
    }
    else{
      int len = sm.getMembershipInfo().getMembers().length;
      if(len > 1){

        try{

          SpreadMessage send = new SpreadMessage();
          send.setObject("recovery");
          send.addGroup(group);
          send.setSafe();
          this.connection.multicast(send);
          recoveryMode = 1;

        }
        catch(Exception e){System.out.println(e.toString());}
      }
      else{

        try{

          String cmd = "ifconfig eth1 up";
          Process child = Runtime.getRuntime().exec(cmd);
          recoveryMode = 0;

          cmd = "/etc/init.d/heartbeat start";
          child = Runtime.getRuntime().exec(cmd);
```

```
          }
          catch(Exception e){System.out.println(e.toString());}

        }

        hasInitiated = 1;
      }
    }
  }

public void onManagerEvent(ManagerEvent event) {
  try {

    String type = event.toString();

    if(type.contains("ExtensionStatusEvent")){
      ExtensionStatusEvent e = (ExtensionStatusEvent) event;

      message.setObject((myName +"/" + e.getExten()+"/"+e.
          getContext()+"/"+ e.getStatus()+"/").toString());
      message.addGroup(group);
      message.setSafe();
      this.connection.multicast(message);

    }



    } catch (Exception e) {
      System.out.println("ERRO:" + e.toString());

    }


  }

public synchronized void updateBuffer(String msgReceived){
  char aux[] = msgReceived.toCharArray();
```

```
ArrayList<Character> sender = new ArrayList<Character>();
ArrayList<Character> ext = new ArrayList<Character>();
ArrayList<Character> context = new ArrayList<Character>();
ArrayList<Character> status = new ArrayList<Character>();
int i;
for(i=0;i<aux.length;i++){
  while(aux[i] != '/'){
    sender.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    ext.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    context.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    status.add(aux[i]);
    i++;
  }
}

char[] sndr = new char[sender.size()];
for(int j = 0; j< sndr.length;j++){
  sndr[j] = sender.get(j);
}

try {
  char[] extension = new char[ext.size()];
  for (int j = 0; j < extension.length; j++) {
    extension[j] = ext.get(j);
  }
  char[] cont = new char[context.size()];
```

```
    for (int x = 0; x < cont.length; x++) {
      cont[x] = context.get(x);
    }
    char[] stat = new char[status.size()];
    for (int y = 0; y < stat.length; y++) {
      stat[y] = status.get(y);
    }
    int s = Integer.parseInt(new String(stat));
    String clicmd;
    switch(s){
      case 0: clicmd = "NOT_INSUSE";break;
      case 1: clicmd = "INUSE";break;
      case 2: clicmd = "BUSY";break;
      case 4: clicmd = "UNAVAILABLE";break;
      case 8: clicmd = "RINGING";break;
      case 16: clicmd = "ONHOLD";break;
      default: clicmd = "UNKNOWN";break;


    }



    String extensionAux = new String(extension);

    bufferedMessages.put(extensionAux, Integer.toString(s));
  }
  catch(Exception e){
    System.out.println(e.toString());
  }
}

 public synchronized void insert(String msgReceived,int
    updateAst){

  char aux[] = msgReceived.toCharArray();
  ArrayList<Character> sender = new ArrayList<Character>();
  ArrayList<Character> ext = new ArrayList<Character>();
  ArrayList<Character> context = new ArrayList<Character>();
```

```
ArrayList<Character> status = new ArrayList<Character>();
int i;
for(i=0;i<aux.length;i++){
  while(aux[i] != '/'){
    sender.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    ext.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    context.add(aux[i]);
    i++;
  }
  i++;
  while(aux[i] != '/'){
    status.add(aux[i]);
    i++;
  }
}

char[] sndr = new char[sender.size()];
for(int j = 0; j< sndr.length;j++){
  sndr[j] = sender.get(j);
}

try {
  char[] extension = new char[ext.size()];
  for (int j = 0; j < extension.length; j++) {
    extension[j] = ext.get(j);
  }
  char[] cont = new char[context.size()];
  for (int x = 0; x < cont.length; x++) {
    cont[x] = context.get(x);
  }
```

```
    char[] stat = new char[status.size()];
    for (int y = 0; y < stat.length; y++) {
      stat[y] = status.get(y);
    }
    int s = Integer.parseInt(new String(stat));
    String clicmd;
    switch(s){
      case 0: clicmd = "NOT_INSUSE";break;
      case 1: clicmd = "INUSE";break;
      case 2: clicmd = "BUSY";break;
      case 4: clicmd = "UNAVAILABLE";break;
      case 8: clicmd = "RINGING";break;
      case 16: clicmd = "ONHOLD";break;
      default: clicmd = "UNKNOWN";break;
    }


    String extensionAux = new String(extension);
    dictionary.put(extensionAux, Integer.toString(s));

    if(updateAst == 1){
      CommandAction ca = new CommandAction("devstate change
          Custom:"+new String(extensionAux)+" "+ clicmd);
      CommandResponse cr = (CommandResponse) managerConnection.
          sendAction(ca);
    }
  }

  catch (Exception e) {
    Logger.getLogger(Spread.class.getName()).log(Level.SEVERE,
        null, e);
  }

}
```

# Apêndice B

# Scripts

Seguem-se as scripts criadas e a aplicação que remove o *lock* do FreePBX:

## watch.sh

```
#!/bin/sh
 while inotifywait -qq –exclude .*[.][c][o][n][f].+ -e modify /mnt/glusterfs/shared/etc/asterisk; do
 asterisk -rx reload
```

## move.sh

```
#!/bin/sh
 mv $1 $1
```

## unlock.sh

```
#!/bin/sh
 java -jar "/usr/src/removeLock.jar"
```

## removeLock.java

```
import java.sql.*;

   public class RemoveLock
   {
       public static void main (String[] args)
       {
           Connection conn = null;

           try
           {
               String userName = "testuser";
               String password = "testpass";
               String url = "jdbc:mysql://localhost/freepbx";
               Class.forName ("com.mysql.jdbc.Driver").
                  newInstance ();
               conn = DriverManager.getConnection (url, userName
                  , password);

               Statement s = conn.createStatement ();
         s.executeQuery ("SELECT * FROM conc");
         ResultSet rs = s.getResultSet ();

         while (rs.next ())
         {
              int lock = rs.getInt ("num");

          if(lock == 0)
          {

          }

          else{

            String mineIP = rs.getString("serverIP");
            if(mineIP == args[0]){
```

```
            }

            else{

            s = conn.createStatement ();
            String command =
              s.executeUpdate ("UPDATE conc SET num = 0 WHERE
                 num = 1;");

            }

          }

         }

         rs.close ();
         s.close ();

           }
           catch (Exception e)
           {
               System.err.println ("Cannot connect to database
                  server");
           }

      }
  }
```