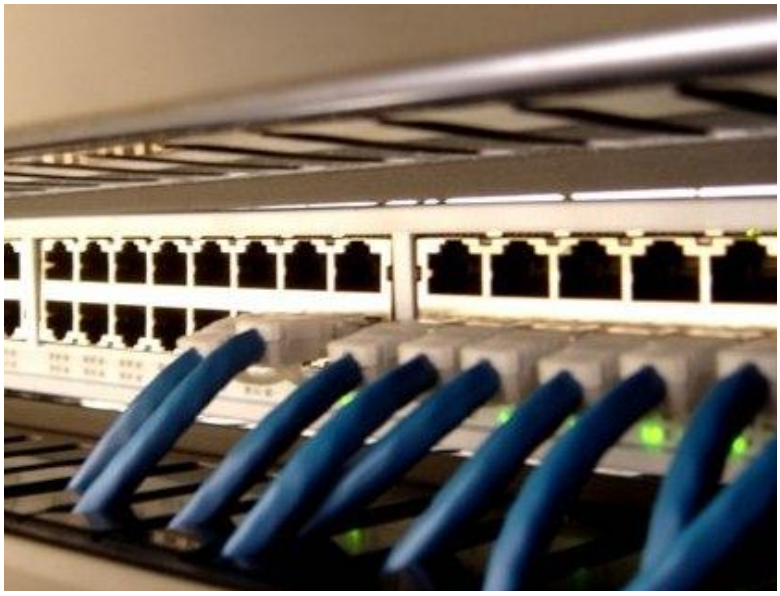
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

ISEL



Sistema de Gestão e Controlo da Atribuição de Endereçamento de Rede

CARLOS JORGE DINIS ABRANTES
(Bacharel)

Trabalho de projecto realizado para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientadores:
Professor-adjunto Pedro Ribeiro, ISEL

Júri:
Presidente: Professor-adjunto Vítor Almeida, ISEL
Vogais:
Professor-adjunto João Ascenso, ISEL
Professor-adjunto Pedro Ribeiro, ISEL

Novembro de 2011

Agradecimentos

Ao Professor Pedro Ribeiro, por orientar deste trabalho, pelas discussões tidas, pelos seus conselhos e pela sua disponibilidade.

À minha família por todo o incentivo prestado, em especial à minha mãe por me ter transmitido os valores que tenho e por ter tido a força necessária quando a vida assim o exigiu.

À minha namorada pelo apoio transmitido e pelas inúmeras tarefas somadas.

Aos meus amigos e colegas de curso Leandro Nunes e Vítor Santos que me acompanharam ao longo do curso em trabalhos de grupo, sem os quais seria certamente muito mais difícil.

Ao meu amigo Pedro Piedade pelo apoio e opiniões trocadas.

À minha amiga Isabel Amsellem pelo apoio e sugestões dadas.

À minha chefia directa João Luis pelo apoio demonstrado.

Resumo

Com a evolução tecnológica, a informática assumiu um papel fulcral na nossa vida actual tanto numa vertente doméstica como para as mais variadíssimas organizações. As redes IP estão disseminadas por todo o lado, escritórios, habitações, estabelecimentos de ensino, espaços comerciais, com as mais diversas topologias e dimensões, que como qualquer recurso devem ser bem geridas.

O presente projecto insere-se neste contexto, pretendendo criar um sistema para dar apoio no controlo e gestão de endereçamento IP dados por um servidor DHCP. A problemática central que este projecto pretende tratar é a detecção da configuração forçada de endereços IP por parte dos utilizadores, notificando tais situações aos administradores de rede. O sistema é constituído por vários componentes como Linux, MySQL, ISC DHCP, SNMP, PHP, APACHE e irá interagir com equipamentos de rede que possuam tabelas de endereços para que a informação, nestas encontradas, seja comparada com a informação dos endereços fornecidos pelo servidor DHCP. O sistema dá suporte a uma página *web* que pretende auxiliar algumas configurações e fornecer informações estatísticas sobre a rede.

Palavras-chave

DHCPd; Protecção IP forçado; ISC DHCP; OMAPI; dhcpctl.

Abstract

With technological developments, the computer assumed a central role in our present life both domestic and to the most different organizations. IP networks are used everywhere, offices, home, schools, shopping centers with the most topologies and very varied dimensions as any other resource should be well managed.

This project fits into this context with the purpose of building a support system to control and management of IP address given by the DHCP server. The central problem that this project wants to treat is the detection of forced IP configuration by users and report such situations to the networks administrators. The system will consist of various components such as Linux, MySQL, ISC DHCP, SNMP, PHP, APACHE interacting with network devices that have address tables to cross check this information with the DHCP server lease information. The system supports a web page with the purpose of helping some settings and provide statistical information of the network.

Key words

DHCPd; forced IP protection; ISC DHCP; OMAPI; dhcpctl.

Índice

Índice 5
Índice de Figuras 7
Glossário e definições 9
1. Introdução 10
1.1. Problemática central 13
1.2. Objectivos Propostos 14
1.3. Organização do relatório 16
2. Análise dos dados existentes 17
3. Análise do cruzamento dos dados 20
4. Solução do sistema 26
4.1. Interacção com servidor DHCP 26
4.2. Interacção com os equipamentos de rede 28
4.3. Implementação 29
4.3.1. Ambiente de desenvolvimento 29
4.3.2. Configuração do sistema 31
4.3.3. Base de dados 34
4.3.4. *Logs* produzidos 42
4.3.5. Interacção com a base de dados 45
4.3.6. Recepção de pedidos DNS update 48
4.3.7. Consulta via OMAPI de uma *lease* 58
4.3.8. Processamento das *leases* activas 62
4.3.9. Normalização do formato dos endereços MAC 66
4.3.10. Processamento das entradas ARP 67
4.3.10.1. Leitura e inserção entradas ARP 69
4.3.10.2. Processamento das excepções 70
4.3.10.3. Renovação das situações anormais 71
4.3.10.4. Verificação das novas entradas via OMAPI 72
4.3.10.5. Rotação das entradas ARP 73
4.3.11. Notificação 73
4.3.12. Página de gestão 76
5. Testes realizados 86
6. Dificuldades encontradas 90

7. Melhoramentos.....92
8. Problemática IPv6 94
9. Conclusão 96
10. Bibliografia 98

Índice de Figuras

Figura 1 – Fluxo do pedido de IP 11
Figura 2 – Fluxo da renovação do IP 11
Figura 3 – Fluxo da libertação do IP.....12
Figura 4 – ARP Table.....17
Figura 5 - ARP *table* por SNMP 18
Figura 6 – Verificação Falso Positivo com TI 21
Figura 7 – Verificação com Falso Negativo com TI.....22
Figura 8 – Verificação com Falso Positivo 23
Figura 9 – Verificação sem erros.....24
Figura 10 – Verificação sem erros II.....25
Figura 11 – Interacção com o servidor DHCP.....27
Figura 12 – Interacção com equipamentos de rede.....28
Figura 13 – Interface Virtual 29
Figura 14 – Network device list 30
Figura 15 – Criação da base de dados 34
Figura 16 – Tabela active_lease 35
Figura 17 – Tabela History 36
Figura 18 – Tabela arp 37
Figura 19 – Tabela exception.....37
Figura 20 – Tabela abnormal 38
Figura 21 – Tabela users 39
Figura 22 – Tabela hosts.....40
Figura 23 – Tabela statspool.....40
Figura 24 – Tabela notification 41
Figura 25 – Constituição DNS *update* 50
Figura 26 – Secção *header* 50
Figura 27 – Secção zone 51
Figura 28 – Fluxo do processamento das *leases* activas (uma interacção).....62
Figura 29 – Interacção do processamento das entradas ARP.....69
Figura 30 – Estrutura da página.....76
Figura 31 – Menu da página de gestão.....78
Figura 32 – Tabela com informação.....79
Figura 33 – *Layout* da página.....80
Figura 34 – Formulário 82
Figura 35 – Hiperligações para navegação.....83

Figura 36 – Tabela com *delete*.....84
Figura 37 – Diagrama da rede virtual implementada 86

Glossário e definições:

API – Application Programming Interface

ARP – Address Resolution Protocol

Broadcast – Tipo de comunicação cujo objectivo é que a mensagem enviada seja entrega a todos os computares da sub-rede

CSS – Cascading Style Sheets

DDNS – Dynamic DNS

DHCP – Dynamic Host Configuration Protocol

DNS – Domain Name System

Header file – Ficheiro que contém declarações de funções

HTML – Hyper Text Markup Language

IANA – Internet Assigned Numbers Authority

IOS – Internetwork Operating System

IP – Internet Protocol

IPv4 – Internet Protocol version 4

IPv6 – Internet Protocol version 6

Lease – Associação entre um endereço IP e MAC tendo uma determinada validade temporal

Log – Ficheiro que contém informação sobre o funcionamento da aplicação

MAC – Media Access Control

MIB – Management Information Base

NAT – Network Address Translation

OID – Object Identifier

OMAPI – Object Management Application Programming Interface

Open source – Programa cujo código fonte é disponibilizado publicamente

RPM – Red Hat Package Manager

RADIUS – Remote Authentication Dial In User Service

RFC – Request for Comments

SNMP – Simple Network Management Protocol

URL – Uniform Resource Locator

VLAN – Virtual Local Area Network

CAM – Content Addressable Memory

1. Introdução

Como todos sabemos, o acesso à internet, no tempo presente, é fácil sendo um repositório de informação de tamanho incalculável onde podemos aprender a fazer quase tudo de uma forma cómoda, simples e a área da informática não é excepção. Ao longo da história, verificamos que infelizmente, em variadíssimas situações, o uso do conhecimento não é o mais apropriado, sendo utilizado para realizar actividades que só têm o objectivo de prejudicar terceiros. Ao juntarmos estes dois factos facilmente concluímos que o uso indevido de uma rede IP deverá ser detectado e notificado para que os respectivos administradores tomem as acções que pensarem ser as adequadas.

Uma rede de computadores IPv4 tem tipicamente um ou mais servidores DHCP, tendo estes o objectivo de fornecer a configuração de rede necessárias para poder existir conectividade e, assim, ser efectuado o tráfego pretendido. Apesar de não ser imprescindível para o funcionamento de uma rede, estes servidores estão sempre presentes, pois são a forma mais pratica de obter as configurações, em alternativa, seria necessário configurar manualmente vários parâmetros e fazer a gestão de quem tem o IP x, se podemos usar o IP y, etc. Num cenário de utilização do servidor DHCP, um computador ao ser ligado a uma rede envia uma mensagem *broadcast*, *dhcpdiscovery*, cujo objectivo é pedir aos servidores DHCP existentes o envio das configurações, estes respondem com uma mensagem, *dhcpoffer*, contendo as configurações de rede a utilizar. O cliente para informar que irá utilizar as configurações enviadas pelo servidor enviará uma mensagem *dhcprequest*, desta forma o servidor DHCP saberá que a sua mensagem foi aceite. Para finalizar o DHCP confirma com a mensagem *dhcpack* . A partir deste momento o processo está concluído e o cliente pode efectuar tráfego (Figura 1).

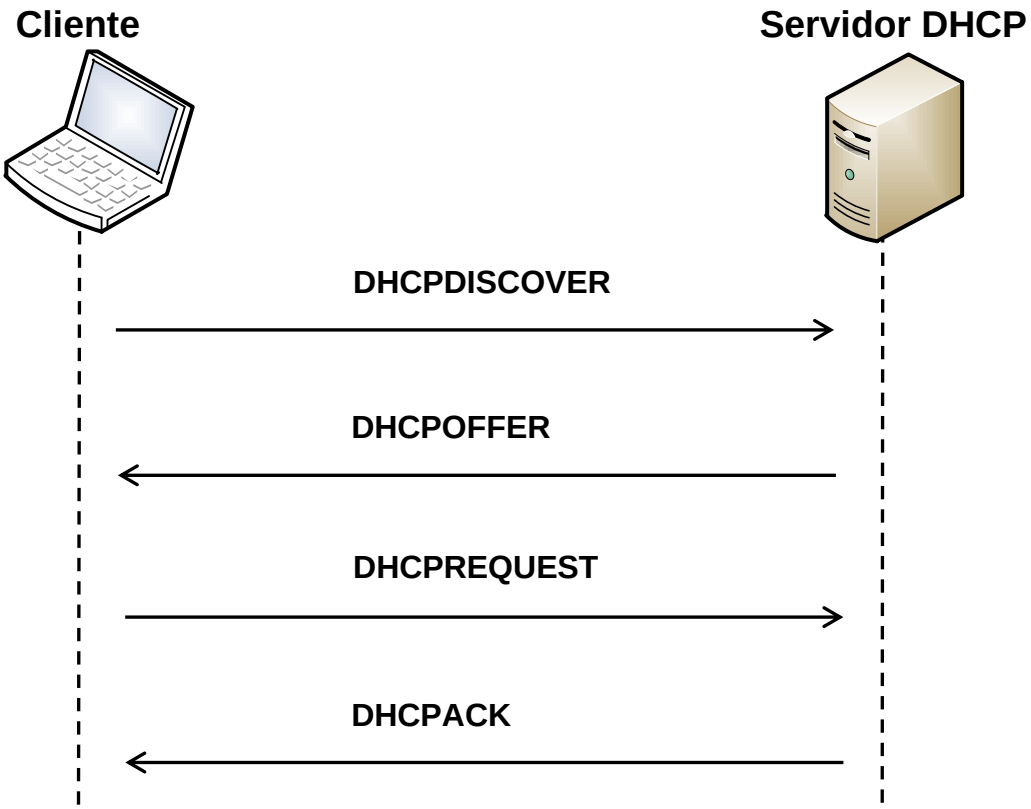


Figura 1 – Fluxo do pedido de IP

A alocação do IP ao cliente não tem uma duração infinita, mas apenas um determinado período de tempo. O cliente, para continuar a usar o IP atribuído, é obrigado a pedir a renovação deste, sendo esta feita através da mensagem *dhcprequest* ao qual o servidor responde com um *dhcpack* se a renovação pedida for aceite.

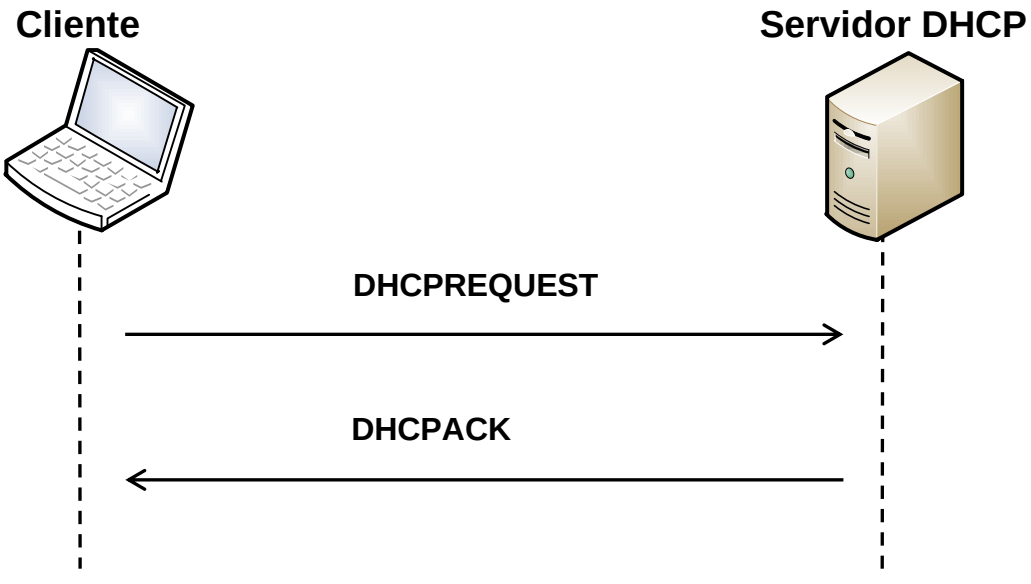


Figura 2 – Fluxo da renovação do IP

Quando o cliente já não tem necessidade do IP alocado pode libertar o mesmo enviando apenas uma mensagem *dhcprelease* ao servidor DHCP (Figura 3).

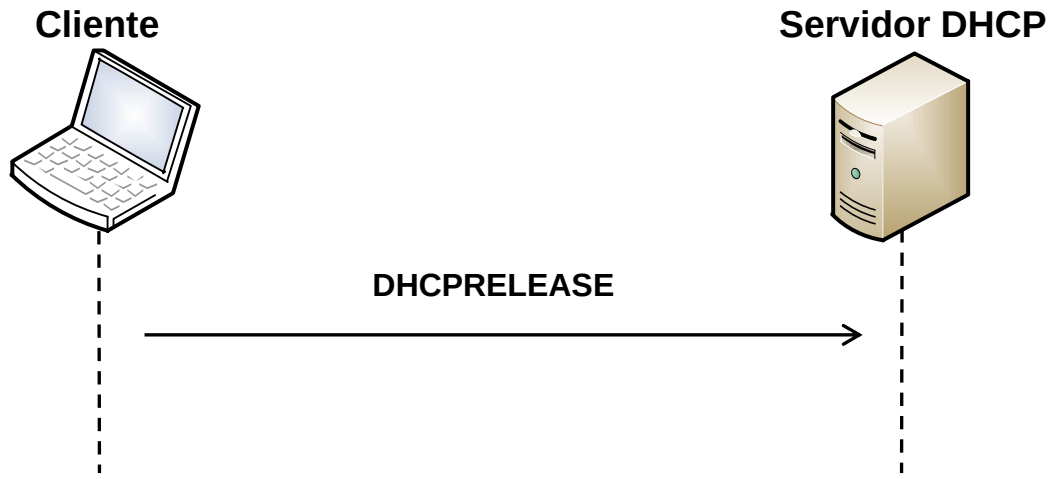


Figura 3 – Fluxo da libertação do IP

Esta introdução tem o objectivo de fornecer as bases necessárias para melhor compreender o contexto deste projecto.

1.1. Problemática central

Em segurança informática existe um ponto muito importante denominado não-repúdio, que consiste num individuo não poder negar determinada acção por ele executada. No contexto deste projecto, como já foi dito anteriormente, um utilizador ao ligar um computador a uma rede recebe as configurações necessárias para poder efectuar tráfego, ficando, assim, a conhecer quais as características da sub-rede em que se encontra. O facto de um utilizador poder mudar o IP a ele atribuído para outro IP da mesma sub-rede, continuando mesmo assim a ter a mesma conectividade, provoca que o único registo existente no servidor DHCP é que aquele endereço MAC tem atribuído o IP dado por este, não ficando a mudança de IP registada. Isto é problemático, pois, caso exista o uso indevido da rede, por exemplo, um ataque a um servidor, não será possível identificar a sua origem, pois como já dissemos anteriormente não existe nenhum registo.

Pela pesquisa realizada, para esta problemática não existe nenhum *standard*, nenhuma ferramenta *open source* que notifique ou proteja de tais situações, no entanto, é importante mencionar que existem algumas protecções como o DHCP *snooping*, *Dynamic* ARP *Inspection* ou o limitar do número de endereços MAC aprendidos por cada interface, em equipamentos de rede que permitem tratar a problemática central. O problema é que estas protecções para além de não estarem disponíveis de forma generalizada por todas as plataformas, têm diferenças de funcionamento ou limitações diferentes (por exemplo ao nível do tamanho da tabela gerada pelo DHCP *snooping*) consoante a versão do sistema. Estas protecções, no entanto, podem ser demasiado restritivas, por exemplo, o facto de limitar o número de endereços MAC de numa interface pode impedir a ligação de um hub num ponto de rede (algo usual num escritório) ou mesmo o uso de máquinas virtuais cuja configuração das suas interfaces de rede seja *bridged*. O facto de usarmos DHCP *snooping* com *Dynamic* ARP *Inspection* inviabiliza o uso de máquinas com o IP configurado manualmente, o tráfego destas nunca vai ser aceite pois não existiu nenhuma mensagem do DHCP a atribuir o endereço IP (não existirá assim na tabela construída pelo DHCP *snooping*). É necessário, ainda, ter em atenção que existem técnicas de alta disponibilidade que recorrem ao ARP gratuito no seu funcionamento despoletando as contramedidas da infra-estrutura que o podem deixar inoperativo.

1.2. Objectivos Propostos

No ponto anterior foi apresentada a problemática central deste projecto, não sendo o único propósito deste projecto o seu tratamento. O objectivo referente à problemática central deverá ser conseguido se forem reconhecidas as situações de endereços IP forçados na rede e geradas as respectivas notificações para os responsáveis pela gestão da infra-estrutura de rede. Um histórico das referidas situações é guardado para mais tarde poder ser pesquisado.

Podemos dizer que gestão é garantir o uso eficaz dos recursos disponibilizados, pois estes são finitos. Para garantirmos esta gestão é necessário termos informação, dados concretos sobre o estado dos recursos em causa, servindo isto de motivação para a aplicação suportar o fornecimento de dados estatísticos que permitam apoiar a gestão de uma rede, como as *leases* activas, histórico da atribuição de endereços IP a computadores ou o estado das *pools* de IP. Um exemplo claro é o estado do consumo das *pools* IP a atribuir pelo servidor DHCP, sendo fundamental saber se as referidas *pools* estão a ficar esgotadas para poderem ser tomadas as medidas adequadas. Estas estatísticas permitem também ajudar a verificar o impacto de alterações na configuração da rede, por exemplo, se alterarmos o tempo de uma *lease* qual o seu impacto para o estado das pools? Não foi encontrada nenhuma ferramenta que cumpra todos os pontos deste objectivo, no entanto, é importante referir algumas ferramentas interessantes existentes e suas funcionalidades:

- dhcpd-snmp: Disponibiliza informações sobre as *pools* via SNMP, tem de ser instalado no próprio servidor DHCP e cria OID com as respectivas informações consultando o ficheiro de *leases* e de configuração do DHCP;
- dhcp-pools: Disponibiliza informações sobre as *pools* e *leases* activas, tem de ser instalado no próprio servidor DHCP e resume-se a um comando que consulta o ficheiro de *leases* e de configuração;
- DHCP Lease Parser: Esta ferramenta disponibiliza a funcionalidade de guardar as *leases* numa base de dados MySQL.

Como podemos verificar as ferramentas apenas cumprem parte do pretendido e todas elas têm de ser instaladas na mesma máquina do servidor DHCP. Como iremos ver nos próximos capítulos é algo que para o sistema desenvolvido é apenas uma possibilidade e não uma obrigatoriedade sendo possível a sua instalação noutras máquinas que não a do servidor DHCP.

Numa infra-estrutura de rede nem todos computadores obtêm o IP de forma dinâmica, pois disponibilizam serviços, tendo por isso a necessidade de obter do servidor DHCP sempre o mesmo IP, ou seja, uma atribuição estática. Por exemplo, um servidor de base de dados que forneça informação a vários clientes, não pode sempre que por algum motivo é reiniciado ter um IP diferente. Isto implica que o servidor DHCP seja configurado para atribuir sempre o mesmo IP ao computador em causa, o que tipicamente exige uma reinicialização do servidor para que a nova configuração (a nova atribuição estática) fique activa. Não foi encontrada nenhuma ferramenta que permita faze-lo de uma forma transparente, que não necessite de reiniciar o servidor, algo que é feito no sistema desenvolvido, ou seja, permite configurar novas atribuições estáticas sem necessidade de reiniciar o servidor. Permite, também, consultar as atribuições existentes e eliminar as mesmas (novamente sem necessidade de reiniciar o servidor). Uma alternativa a este método é efectivamente o computador ter configurado manualmente um IP (IP forçado) não estando este IP no servidor DHCP para ser atribuído, devendo esta ser considerada uma situação normal, para suportar a referida situação é possível configurar no sistema excepções ao processamento, ou seja, situações do conhecimento dos administradores de rede que não são situações anormais.

1.3. Organização do relatório

Este relatório, para além do presente capítulo introdutório, é constituído pelos capítulos:

- Capítulo um: capítulo corrente que contém uma introdução ao tema sendo exposta a problemática central, explicado os objectivos propostos e a organização do documento;
- Capítulo dois: onde é feita a apresentação e análise referente aos dados necessários;
- Capítulo três: contém a análise realizada, tendo em conta as conclusões obtidas no capítulo anterior, sobre a melhor forma de fazer o cruzamento dos dados que estão disponíveis para atingir os objectivos propostos;
- Capítulo quatro: pretende fazer a apresentação da implementação de todo o sistema, interacções com servidor DHCP e equipamentos de rede, a solução encontrada, tecnologias usadas e os fluxos realizados pelo sistema;
- Capítulo cinco: apresenta os testes realizados, sempre necessários para confirmar o correcto funcionamento do sistema;
- Capítulo seis: contém as dificuldades mais relevantes encontradas na realização deste projecto;
- Capítulo sete: apresentação de possíveis melhoramentos;
- Capítulo oito: analise à problemática central em IPv6;
- Capitulo nove: descrição das conclusões;
- Capítulo dez: onde está presente a bibliografia consultada.

2. Análise dos dados existentes

A primeira fase deste projecto consistiu em fazer uma análise dos dados que iriam ser necessários para conseguir assegurar os objectivos pretendidos e rapidamente se chegou à conclusão que seria preciso saber o estado da rede, ou seja, que pares IP/MAC que estão activos na rede e saber quais os pares que foram atribuídos pelo servidor de DHCP.

Relativamente à informação da rede, podemos obter a mesma em equipamentos que tenham uma ARP *table* (por exemplo *routers*) que possa ser consultada. Esta tabela (Figura 4) contém, entre outras informações, o IP, o MAC e há quanto tempo é que o *router* obteve a informação (campo *age*) desse par IP/MAC. Quando um computador é ligado a uma rede, na primeira comunicação (depois de ter um IP configurado) que tem com o *router*, é adicionada por este uma nova entrada na tabela referente a esse computador.

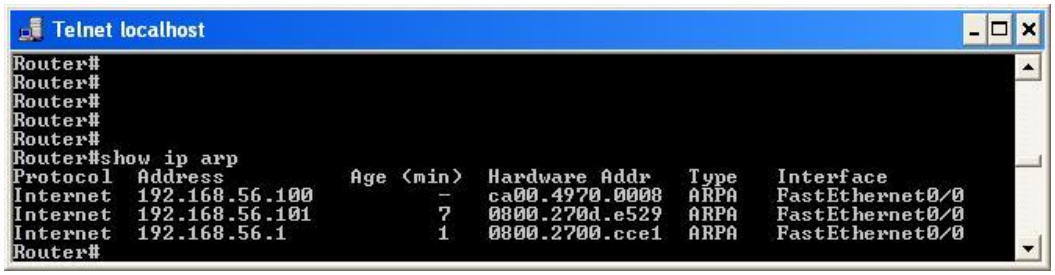


Figura 4 – ARP Table

Sempre que ocorre uma interacção ARP *Request/Reply* iniciada por uma das partes, estas actualizam o campo *age* da entrada respectiva colocando-o com o valor 0. Assim é possível saber se um determinado computador continua activo ou se provavelmente já terá abandonado a rede. Este ponto é importante pois não devemos olhar para esta tabela cegamente, uma vez que a entrada não desaparece assim que o computador é desligado da rede, pelo contrário, mantém-se por bastante tempo dependendo da configuração do tempo de *cache* no *router*. Esta situação é muito relevante pois pode provocar conclusões precipitadas, se pensarmos que as entradas são guardadas durante muito tempo mesmo após um computador abandonar a rede e que esta entrada continua a ser seleccionada para verificação de autenticidade poderá originar uma situação de falso positivo assim que a *lease* do DHCP expirar. Apesar de a tabela conter a informação atrás descrita, não significa que a mesma seja disponibilizada quando esta é consultada via SNMP. Após alguns testes é possível verificar que apenas parte da informação da tabela é disponibilizada

por SNMP no ramo *atTable* da MIB disponível no *router*, neste caso somente o IP, interface e MAC estão disponíveis (Figura 5). Devido a este facto

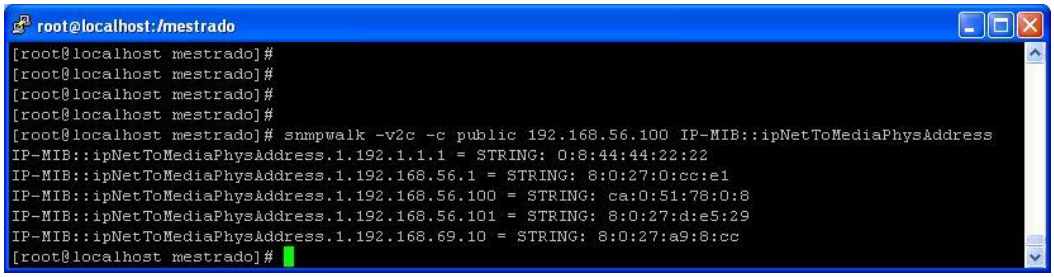


Figura 5 - ARP *table* por SNMP

perde-se a noção do estado actual de actividade das máquinas mencionadas porque não sabemos há quanto tempo estão inactivas.

Algo que também acontece quando ligamos um computador a uma rede, é este (se assim tiver configurado) pedir a sua configuração de rede através do protocolo DHCP incluindo o IP. O servidor DHCP reserva por um determinado tempo (configuração no servidor) o IP atribuído ao computador (este tempo é renovado pelo computador para que possa continuar a usar o IP dado), podendo este tempo ser de vários minutos, horas ou dias. Observando as práticas correntes tipicamente quando um computador é desligado da rede não é feito o *release* do IP (informar o servidor DHCP que o IP atribuído já não é necessário), este é retirado abruptamente sendo retirado o cabo, desligado indevidamente, alguns sistemas operativos nem fornecem comandos que permitam fazer o referido *release*. Esta situação provoca que existam *leases* activas de computadores que já abandonaram a rede, não sendo também possível confiar cegamente no *end time* da *lease* pois efectivamente é possível fazer o *release* da mesma.

Pela descrição verificada anteriormente poder-se-á afirmar que os dados existentes não podem ser vistos como absolutamente verdadeiros devido à natureza dos mesmos.

Por último, é importante referir que na análise destes dados foram detectadas duas situações à partida não previstas, o servidor de DHCP quando consultado via OMAPI sobre o estado de uma *lease responde* em algumas situações com o erro "*more than one object matches key*", esta situação seria à partida provocada pelo facto de não ser suportado o retorno de mais que uma *lease,* no entanto, verifica-se após investigação que o servidor, provavelmente

como forma de optimização de desempenho, não actualiza de imediato o estado das *leases* existentes em memoria, no entanto, verifica-se também que se for efectuada uma consulta apenas com o IP e não com o IP e MAC o servidor já devolve as informações sobre a *lease*. Esta situação pôde ser comprovada pelo ficheiro de *leases* do servidor (neste ficheiro é guardada toda a informação sobre o estado actual do servidor), tendo sido verificada a existência de múltiplas referências com o estado activo para o mesmo IP e MAC. Poucos minutos após ter sido verificado o ficheiro, este já não possuía as duas entradas, mas apenas uma (a documentação indica que caso haja mais que um referência à *lease* a última é a actual). O facto de serem devolvidas as informações sobre a *lease* se a pesquisa for feita apenas com o IP é um comportamento que não foi possível compreender, uma vez que não existe documentação sobre o mesmo, no entanto, deverá ser um comportamento erróneo do servidor pois se com dois parâmetros de procura (IP e MAC) o servidor devolve um erro indicando que existe mais que uma *lease* para a procura realizada, diminuir os parâmetros de procura não deveria resolver o problema como o faz.

A segunda é quando o servidor DHCP retorna o erro "*key conflict*", também neste caso se verificou a existência de mais de uma referência ao mesmo par IP MAC e ambas tinham o estado activo (foi testado o *release* do IP num computador neste estado tendo o estado da *lease* assumido o valor *free*). Novamente, não foi possível até à presente data identificar o motivo do comportamento referido, pois não foi possível encontrar documentação nem informação nas listas de apoio, apenas com mais tempo e investigando detalhadamente o código fonte do servidor.

3. Análise do cruzamento dos dados

Como exposto no ponto anterior existem algumas características a ter em atenção para fazermos o correcto cruzamento dos dados provenientes do servidor DHCP e dos equipamentos de rede. Se não tivermos em atenção os pontos anteriores iremos criar os seguintes erros:

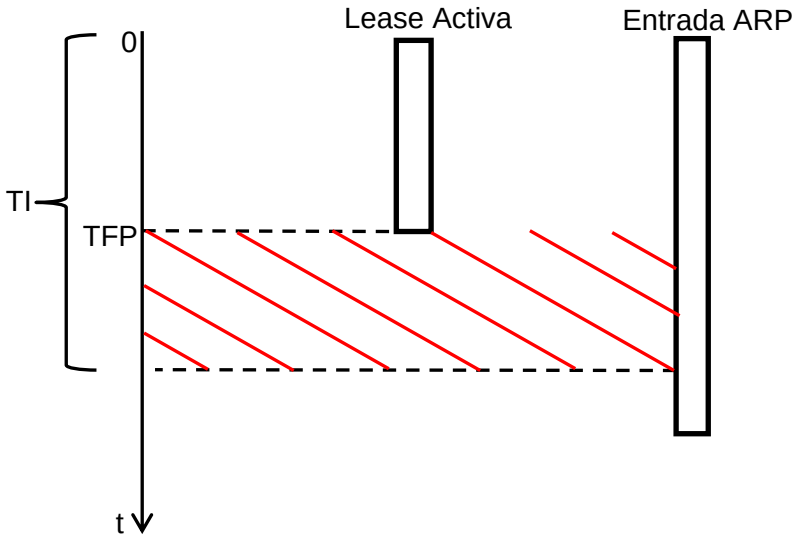
- Quando existe uma *lease* activa e o computador da respectiva *lease* abandona a rede, esta continua activa. Se alguém forçar o seu IP para o IP da *lease* anterior e o cruzamento de dados se basear apenas no IP irá acontecer um falso negativo (o mesmo acontece se assumirmos o *end time* da *lease*). Para evitar esta situação é necessário que o endereço MAC seja incluído na validação.

- A persistência das entradas da tabela ARP, como referido anteriormente, tem uma duração configurável. Falsos positivos irão acontecer se estas entradas forem verificadas e as respectivas *leases* já tiverem expirado. Para resolver este problema as entradas de ARP só irão ser verificadas uma vez (explicado ao longo deste capitulo).

- O retorno dos erros "*key conflict*" e "*more than one object match to the key*", nesta situação irá ser feita a verificação da *lease* apenas com o IP na OMAPI e posteriormente verificado o seu MAC e estado do lado da aplicação de análise.

No início da análise efectuada sobre a natureza dos dados e sobre a melhor forma de fazer o seu cruzamento, uma possibilidade seria através do campo *age* da tabela ARP, assumindo que a partir de um determinado período de inactividade (TI) não seria necessário fazer a sua verificação nos dados do servidor DHCP, desta forma não se iriam validar todas as entradas, mas apenas aquelas que seriam consideradas activas optimizando o funcionamento do programa. Neste processo o ponto-chave seria escolher o valor correcto para TI pois existe o problema dos falsos negativos no caso de o TI ser muito grande pois a verificação da entrada ARP poderia continuar a ser feita tendo a *lease* já expirado. Para facilmente compreendermos a anterior afirmação a Figura 6 pretende explicar esse facto, representando uma linha temporal do instante 0 a t com a duração da *lease* e da entrada ARP. Assumindo que no

instante 0 o respectivo computador abandona a rede e a *lease* expira antes da entrada ARP iremos ter o falso positivo a partir do ponto TFP até ao fim do intervalo TI. A forma de evitar este problema seria escolher um TI menor que o tempo que a *lease* demora a expirar se não for renovada.



Legenda:

TI – Tempo até Inactivo

TFP – Tempo Falso Positivo

Figura 6 – Verificação Falso Positivo com TI

Para ajudar a perceber a situação também indicada anteriormente (falsos negativos), a Figura 7 representa a mesma linha temporal, em que agora o tempo da entrada da tabela de ARP é maior do que o tempo que demora a *lease* a expirar. Assumindo a mesma acção para o instante 0 que na situação anterior, se alguém forçar o IP do seu computador para o IP que tem a *lease* activa (o computador que tinha esta *lease* abandonou a rede em t=0) vai originar um falso negativo desde o instante TFN até ao fim de TFNF. Se alguém causar esta situação (IP forçado) e abandonar a rede antes de TFNF, ou seja, antes de a *lease* activada pelo outro computador expire passará despercebido sem que haja qualquer registo dessa situação.

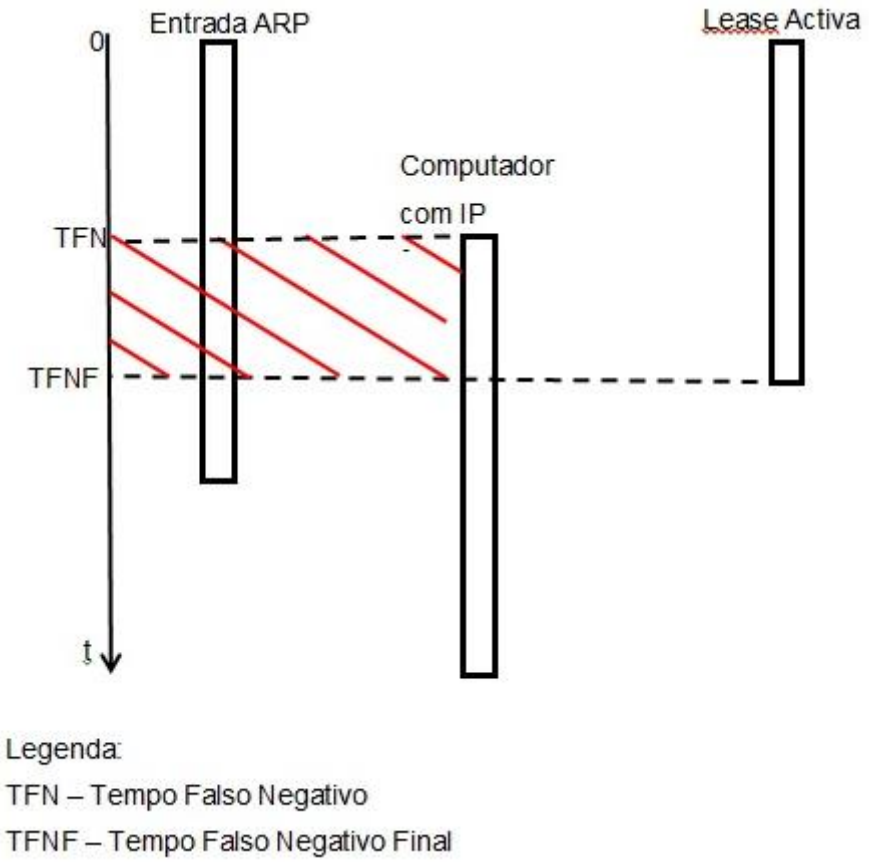
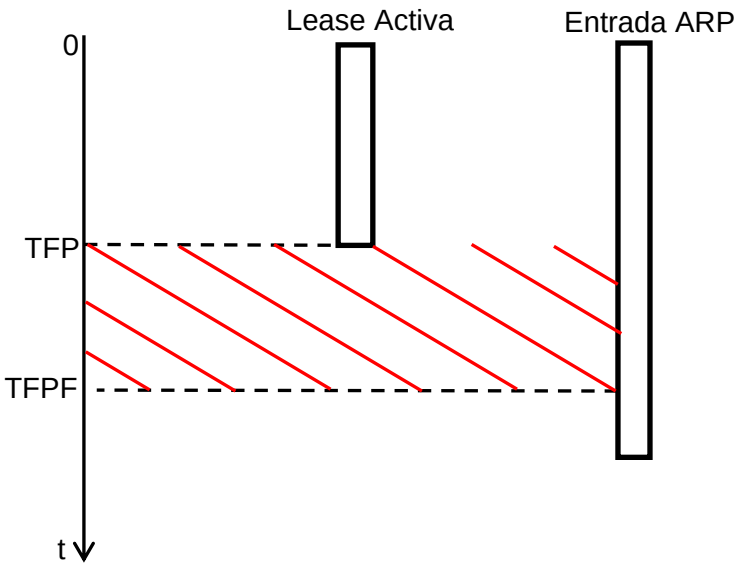


Figura 7 – Verificação com Falso Negativo com TI

Como podemos ver existe a necessidade do MAC ser usado no cruzamento dos dados e por isso será utilizado, algo que não acontecerá com o método usando o TI, este, mesmo que bem escolhido não poderá ser utilizado pois, como anteriormente vimos, o campo *age* da tabela ARP dos equipamentos não é disponibilizado por SNMP na tabela de *Address Translation Group*, assim não temos a noção de há quanto tempo o computador está inactivo para ser comparado com o valor TI.

Neste momento, é possível dizer que para a verificação de uma *lease* será usado o IP e o MAC, no entanto, falta ainda encontrar o quando, quantas vezes ou até quando estas verificações devem ser feitas. Como vimos, não é possível aplicar um método usando um valor TI, sendo necessário estudar outros métodos.

A Figura 8 (mesma representação que as figuras anteriores) mostra que sem nenhum outro mecanismo iremos obter sempre falsos positivos quando na configuração no servidor DHCP o tempo da *lease* seja inferior ao tempo de uma entrada da tabela ARP de um *router* (também configurável).



Legenda:

TI – Tempo Inactivo

TFP – Tempo Falso Positivo

TFPF – Tempo Falso Positivo Fim

Figura 8 – Verificação com Falso Positivo

Se assumirmos pelas configurações que o tempo de *lease* é maior que o da persistência da na tabela ARP, obtemos a Figura 9. Como as verificações são efectuadas sobre as entradas da tabela ARP, se a entrada expirar e deixar de existir esta não será verificada, logo não provocará falsos positivos.

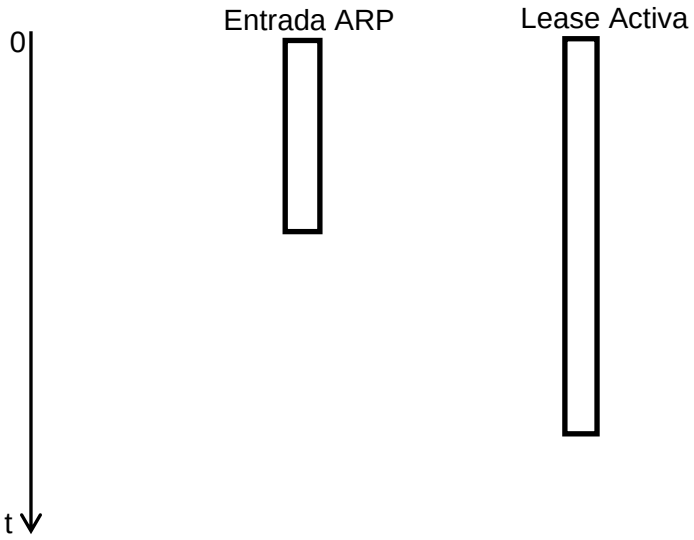
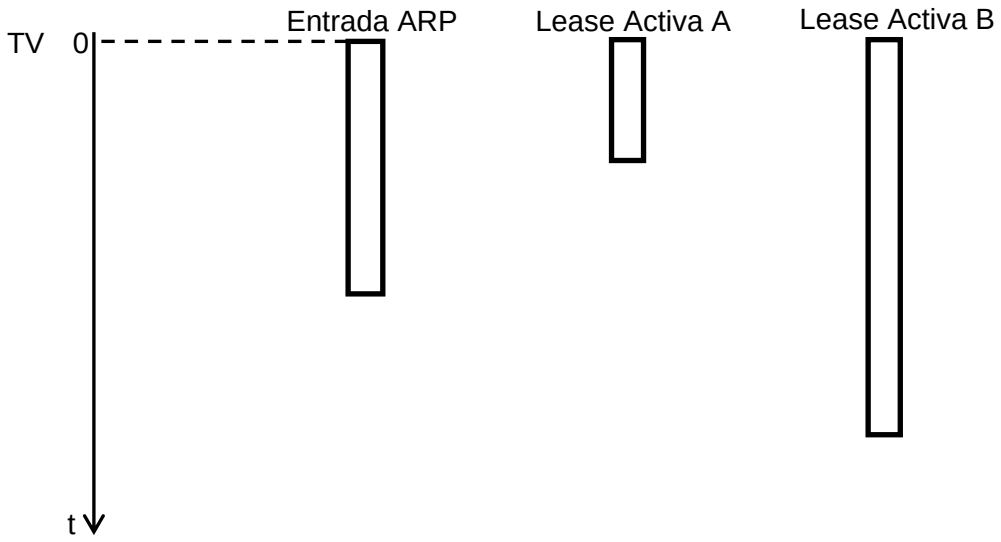


Figura 9 – Verificação sem erros

Este método, apesar de funcionar, entrega um papel muito importante às configurações do servidor DHCP e equipamentos de rede algo que não consideramos ser a melhor forma pois pode introduzir erros, existindo ainda a questão se tais configurações são possíveis de coordenar devido ao tamanho/topologia da rede, tamanho da tabela ARP, tamanho das *pools* do servidor DHCP. Outra questão a levantar seria o desempenho deste método que faria sempre a verificação de todas as entradas da tabela, algo que à partida parece ser bastante penoso.

Após algumas verificações, a conclusão encontrada foi que apenas seria necessário verificar a entrada da tabela ARP uma vez, ou seja, indo periodicamente buscar a tabela ARP aos equipamentos de rede só serão analisadas as entradas de ARP novas. Se determinado par já foi verificado numa interacção anterior já não é necessário efectuar o seu processamento novamente pois já foi tratado (dois pares são considerados iguais se e só se tanto o IP e o MAC forem iguais).



Legenda:

TV – Tempo de Verificação

Figura 10 – Verificação sem erros II

A Figura 10 contém os cenários em que o tempo da *lease* é maior e menor que o de persistência das entradas ARP, sendo o instante t=0 o momento em que um computador é ligado a uma rede (é neste momento que é criada a entrada ARP e a *lease*) e que TV representa o momento da verificação realizada. Neste momento é certo que para a entrada ARP existe a *lease* respectiva, se não existir é porque estamos perante uma entrada ARP que representa uma situação anormal, um IP forçado. Este método funciona correctamente e não depende das configurações dos equipamentos de rede e servidor DHCP, tendo os seus administradores liberdade para efectuar as configurações que acharem mais convenientes. Existe, também, uma grande melhoria relativamente ao método anterior sendo esta muito significativa, o desempenho é bastante superior pois a cada interacção só as novas entradas ARP são verificadas em vez da verificação integral de toda a tabela (método anterior).

4. Solução do sistema

Como indicado anteriormente não foi encontrado nenhum produto capaz de realizar os objectivos propostos, tendo em consideração este facto foi decidido elaborar um sistema capaz de assegurar os mesmos. Este sistema teria que interagir com outros componentes importantes, nomeadamente com os equipamentos de rede e os servidores DHCP.

.

4.1. Interacção com servidor DHCP

Uma solução possível para a interacção com o servidor DHCP, como nas aplicações referidas no ponto 1.2, é a leitura do ficheiro com a informação das *leases*. Este ficheiro é mantido pelo servidor sendo nele registado quando é atribuído um novo IP a um cliente, sendo criada uma nova *lease*. Ao expirar uma *lease* o servidor actualiza o ficheiro com essa informação, assim o ficheiro dhcpd.leases representa o estado das *leases* do servidor (este ficheiro é essencial para servidor, em caso de necessitar de ser reinicializado saber qual os IP atribuídos). Este método (obter conhecimento das *leases* via leitura do ficheiro) implica que a solução implementada seja instalada na mesma máquina que o servidor DHCP (a não ser que fosse implementado um sistema de cópia do referido ficheiro para uma outra máquina) para ter acesso ao ficheiro. Tal pode não ser possível se houver problemas de desempenho no servidor onde reside o servidor DHCP ou se, por exemplo, forem usados vários servidores DHCP dificultando a coordenação do sistema implementado. Além das razões atrás indicadas, esse método não resolve de forma elegante a necessidade de interacção com o servidor DHCP.

O servidor DHCP (ISC) disponibiliza uma API (OMAPI) que permite saber o estado e efectuar algumas configurações no servidor. Esta foi outra forma de interacção analisada para obter os dados das *leases*, mas desta forma seria sempre um sistema que iria consultar periodicamente a informação, algo que para este caso não é o mais interessante, pois o mais correcto é o sistema ter o conhecimento da *lease* assim que esta é criada. Outra razão para este método não ter sido o escolhido para que o sistema obtenha as *leases* activas é o facto de que o OMAPI não suporta o retorno de mais de um objecto, ou seja, ao interagirmos com o servidor será apenas sobre um e um só objecto não sendo possível, por exemplo, pedir que seja retornadas todas as *leases* activas.

Após algum estudo verifica-se que o servidor DHCP permite ser configurado para que sempre que exista uma nova *lease*, este faça um DNS *update* a um servidor DNS sendo possível neste pedido incluir a informação do IP e MAC da respectiva *lease*. Esta capacidade é a necessária para obter o conhecimento das *leases* activas, desta forma o sistema obtém conhecimento da *lease* no momento da sua criação e evita a necessidade de ser instalado na mesma máquina que o servidor DHCP, ou seja, cumpre os requisitos tendo sido a forma escolhida para receber as *leases* activas. O sistema terá que ser implementado ficando à escuta dos pedidos enviados pelo servidor DHCP simulando ser um servidor DNS.

Neste momento, está encontrada a forma de como o sistema tem conhecimento de todas as novas *leases* que existem na rede, mas é necessário saber quando é que estas expiram sendo que os pedidos DNS *update* não são enviados neste caso. Para servir este propósito a OMAPI, descrita anteriormente neste ponto, é a escolha adequada permitindo manter a capacidade do sistema correr numa máquina independente do servidor DHCP. A forma como as interacções com o servidor DHCP está neste momento encontrada e é resumida pela Figura 11.

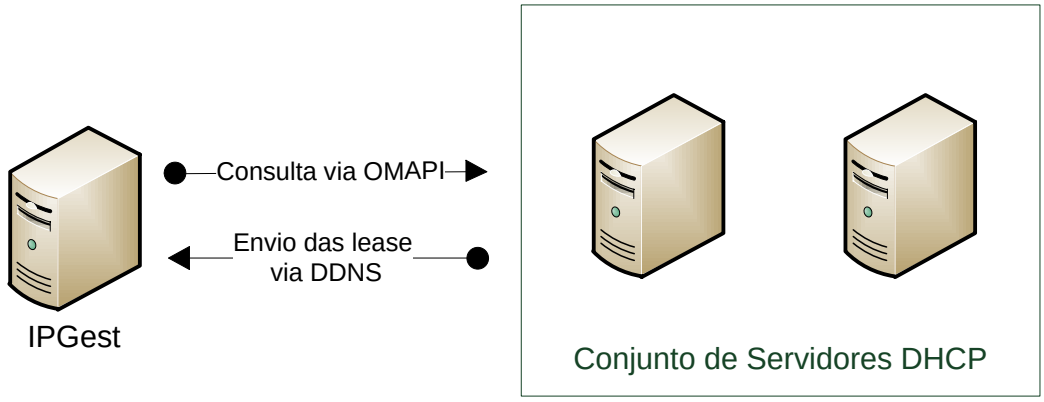


Figura 11 – Interacção com o servidor DHCP

4.2. Interacção com os equipamentos de rede

Este ponto foi bastante mais simples que o anterior dado a existência de um protocolo *standard* criado com o objectivo de gerir equipamentos numa rede IP, o SNMP. Este protocolo não tem, no entanto, o objectivo de definir qual a informação (presente em variáveis), que tem de ser disponibilizada, sendo esta da responsabilidade de cada dispositivo, um exemplo é como podemos ler no ponto 2, o facto de apenas alguma informação ser disponibilizada, é possível que outros equipamentos disponibilizem mais informação. O que este protocolo define é a estrutura (MIB) onde a informação disponibilizada está guardada, é nesta estrutura que podemos verificar quais as variáveis existentes e tal o seu identificador (OID).

Do ponto de vista deste projecto e a sua interacção com equipamentos de rede (possuidores de uma tabela ARP) é possível afirmar que este protocolo é implementado por todos os que tenham funcionalidades de nível três (rede) ou garantidamente pela esmagadora maioria dos equipamentos sendo por isso a escolha correcta a fazer (Figura 12).

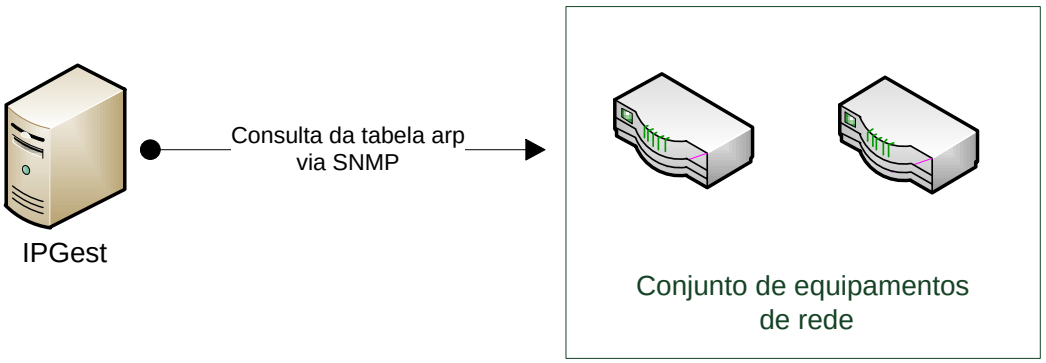


Figura 12 – Interacção com equipamentos de rede

4.3. Implementação

Este capítulo explicará pormenorizadamente os aspectos relacionados com a implementação do sistema, são abordados os seguintes aspectos:

- Ambiente de desenvolvimento
- Configuração do sistema
- Base de dados
- *Logs* produzidos
- Interacção com a base de dados
- Recepção de pedidos DNS *update*
- Consulta via OMAPI de uma *lease* activa
- Processamento das *leases* activas
- Normalização do formato MAC
- Processamento das entradas ARP
- Notificação
- Página de gestão

4.3.1. Ambiente de desenvolvimento

Este projecto, pela sua natureza, para ser desenvolvido e testado necessita dos vários componentes até aqui discutidos, equipamentos de rede, um servidor DHCP, vários computadores para fazer pedidos DHCP, algo que é incompatível com a realidade económica. Para contornar este facto recorremos à tecnologia de máquinas virtuais (Oracle Virtual Box) e a um programa com a capacidade de simular um *router* da CiscoSystems (Dynamips). Quando utilizamos o *Oracle Virtual Box* é criada uma interface de rede virtual que permite a comunicação entre a máquina física e a virtual (Figura 13).

Figura 13 – Interface Virtual

Para utilizarmos o Dynamips necessitamos de uma imagem do IOS da Cisco e de configurar um ficheiro que permite ao programa saber entre outras configurações o local da imagem, quantos equipamentos devem ser simulados e configurações das interfaces de rede.

```
        ...
[localhost]
 [[7200]]
image = c:\Program Files\Dynamips\images\c7200-jk9o3s-mz.124-
19.bin
npe = npe-400
ram = 256

[[ROUTER rt]]
console = 2001
f0/0 = NIO_gen_eth:\Device\NPF_{4544D5B1-988A-4C30-BBF5-
48ABB375CC91}
```

Para que fosse possível o Dynamips utilizar a interface virtual do Virtual Box foi necessário configurar a interface f0/0 do *router* rt (*router* criado no Dynamips) como estando ligado à virtual, tal foi realizado recorrendo à aplicação *Network device list* (incluída no Dynamips) que permite ver os identificadores das interfaces da máquina física (Figura 14).

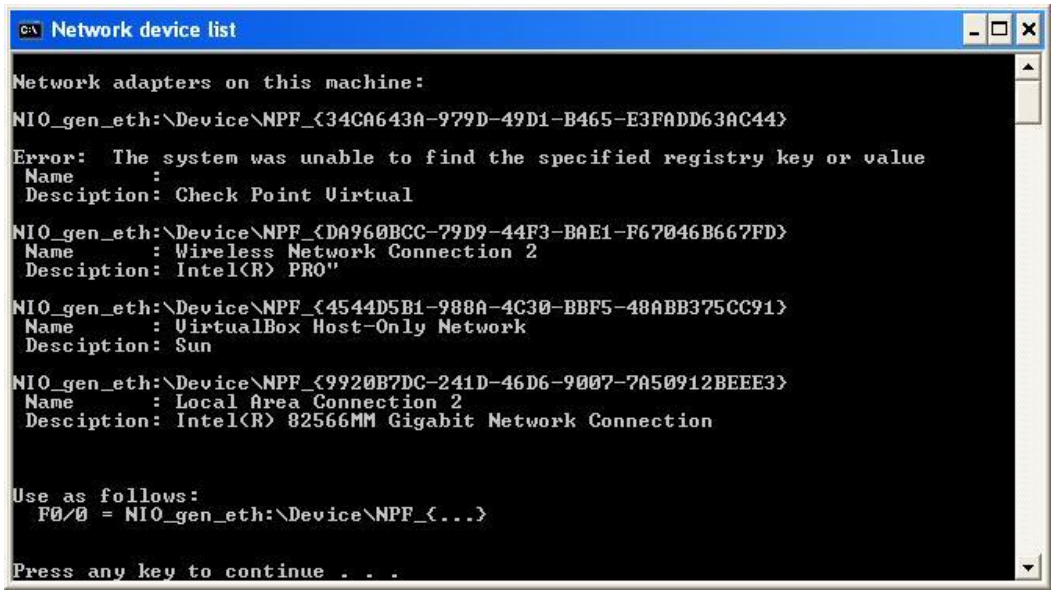


Figura 14 – Network device list

Por fim, foi necessário instalar alguns pacotes de software (formato RPM) como o mysql-dev, dhcp-devel, bind-devel, snmp-devel, php-mysql que contêm os *header files* necessários assim como as respectivas bibliotecas. O código foi escrito no vim 7.0.237 e o compilador usado foi o gcc 4.1.2.

4.3.2. Configuração do sistema

O uso de ficheiros de configuração tem a grande vantagem de evitar a necessidade de ter de alterar o código e voltar a compilar o mesmo quando se pretende fazer uma alteração, por exemplo um IP, por este facto torna-se indispensável o seu uso.

O ficheiro de configuração do sistema permite configurar um conjunto de parâmetros relacionados com os vários componentes que o constituem, no caso da base de dados são os seguintes:

- MySQL:
    - mysql_ip=127.0.0.1
    - mysql_user=root
    - mysql_pass=""
    - mysql_bd=ipgest

Assim é possível de forma simples configurar qual a máquina de destino que detém a componente da base de dados MySQL, ou seja, as credenciais para acesso à base dados e o nome desta. Em seguida, os parâmetros relacionados com o próprio sistema:

- IPgest:
    - Listen_ip=127.0.0.1
    - Listen_port=53
    - Arp_cicle_time=30
    - Lease_check_cicle=30
    - log_path=./ipgest.
    - log_level=1
    - arpfile=./arpread
    - on_dhcp_error=1
    - on_dhcp_error_l=1
    - sendmail=/usr/sbin/sendmail
    - Mail_sender=35950@alunos.isel.ipl.pt

Estas configurações têm o objectivo flexibilizar o funcionamento da aplicação. De forma semelhante ao indicado acima para o MySQL, é possível configurar qual o IP e port que o sistema vai usar para escutar os pedidos DDNS. Os parâmetros Arp_cicle_time e Lease_check_cicle representam o

tempo de pause entre ciclos do processamento da tabela ARP e de *leases* activos, ou seja, quando uma interacção é completada não é iniciada a próxima sem ter passado o valor do parâmetro em segundos. O logpath representa o local e nome onde o sistema irá depositar o seu registo de eventos. Ao nome configurado será acrescentado um sufixo com o formato yymmdd.log, representando o ano, mês e dia. O "./" significa que irá ficar na mesma directoria em que o binário da aplicação seja iniciado, ainda sobre os *logs* é possível configurar o nível de detalhe desejado tendo mais ou menos informação. Arpfile representa o ficheiro temporário que irá servir de *input* para leitura da tabela ARP (explicado posteriormente durante este ponto sobre implementação). Algo que deverá existir neste género de aplicações é uma forma de mudar o comportamento a adoptar em caso de erro, on_dhcp_error e on_dhcp_error_l são variáveis que indicam qual o comportamento a assumir caso a verificação no servidor DHCP resulte num erro, as mesmas são respectivamente usados no processamento da tabela ARP e *leases* activas. Para a notificação é usado o utilitário *sendmail*, sendo este parâmetro a sua localização e usa como remetente o endereço de correio electrónico Mail_sender. Por fim, os parâmetros relacionados com o servidor DHCP:

- DHCP
    - dhcp_ip_port=127.0.0.1 7911

É possível adicionar várias entradas semelhantes à anterior de forma a permitir o suporte para vários servidores DHCP que uma rede possa ter.

Após esta descrição do ficheiro de configuração é apresentada a função que faz a sua leitura, a *void ReadConfig()*. Esta função abre o ficheiro de configuração para leitura, se não conseguir o programa não pode prosseguir pois não tem as informações necessárias para continuar. A leitura é feita linha a linha (*fgets()*) e esta é verificada se é igual a alguma máscara de configuração (*sscanf()*), se for o valor desejado é guardada numa variável global para toda a aplicação ter acesso a esta, caso contrario é lida uma nova linha.

```
while(fgets(l,1024,fconfig)!=NULL){
                sscanf(l,"Listen_port=%d",&LST_PORT);ipdhcp
                sscanf(l,"log_level=%d",&LogLevel);
                sscanf(l,"log_path=%s",&logp);
```

Como podemos verificar acima todo o ficheiro é lido linha-a-linha e cada linha é comparada para obter por todas as configurações. A única leitura que é uma excepção são as configurações dos servidores DHCP visto a máscara de configuração ser igual e não sabermos a quantidade de servidores que irá ser configurada sendo lida à parte das outras configurações. Neste caso, uma diferença notória é o facto de a variável ser alocada dinamicamente (não é previsível o número de servidores que irão ser configurados) e é guardado o número de servidores que são lidos para mais tarde podermos saber qual o tamanho do *array*.

```
        ...
while(fgets(l,1024,fconfig)!=NULL){
                if(sscanf(l,"dhcp_ip_port=%s %d",&str2,&p)){
                        DHCP_IP[i]=malloc(12);
                        DHCP_PORT[i]=(int*) malloc(sizeof(int));
                        strcpy(DHCP_IP[i],str2);
                        DHCP_PORT[i]=(int*)p;
                        i++;
                }
        }
dhcp_n=i;
        ...
```

4.3.3. Base de dados

O sistema desenvolvido tem de guardar informação de forma não volátil pois guarda um histórico, para ser consultado mais tarde pelo utilizador, como informação necessária para o processamento corrente. Para satisfazer esta necessidade foi escolhida uma base de dados MySQL, que sendo distribuída gratuitamente é largamente usada em sistemas UNIX, existindo muita documentação disponível.

Foi criado um script bastante simples que serve o propósito de criar a base de dados pretendida e tabelas que a compõem, bastando um simples comando para criar a mesma (Figura 15).

Figura 15 – Criação da base de dados

Este comando pode variar sendo que nesta forma utilizada as opções por omissão, ou seja, conecta-se ao sistema de base de dados local (a própria máquina) e as credenciais usadas são o utilizador que executa o comando mysql, neste caso o utilizador root, não sendo utilizada palavra-chave. É possível verificar todas as opções suportadas se for dada a opção “--help”, assim como breve referência a opção –u server para indicar o utilizador, -p a palavra-chave e –h o IP ou nome da máquina que têm a base de dados.

Em seguida, são descritas as tabelas da referida base de dados indicando o seu propósito e omitindo a forma como são utilizadas, pois tal será referido ao longo dos vários pontos deste capítulo. Serão, também, mostrados os comandos necessários para a sua criação presentes no script.

Para criar a base de dados propriamente dita é feito o seguinte comando:

```
create database ipgest;
use ipgest;
```

Neste momento existe uma base de dados vazia e estando a mesma seleccionada sendo possível criar as tabelas necessárias (o segundo comando acima referido serve para indicar que os comandos posteriores são para ser executados na base de dados indicada). A primeira tabela criada é a active_lease:

```
create table active_lease(id int auto_increment primary key,
ip varchar(40),
 mac varchar(20),
 start datetime,
stop datetime,
ipdhcp varchar(40));
```

É possível confirmar a sua criação como mostrado na Figura 16:

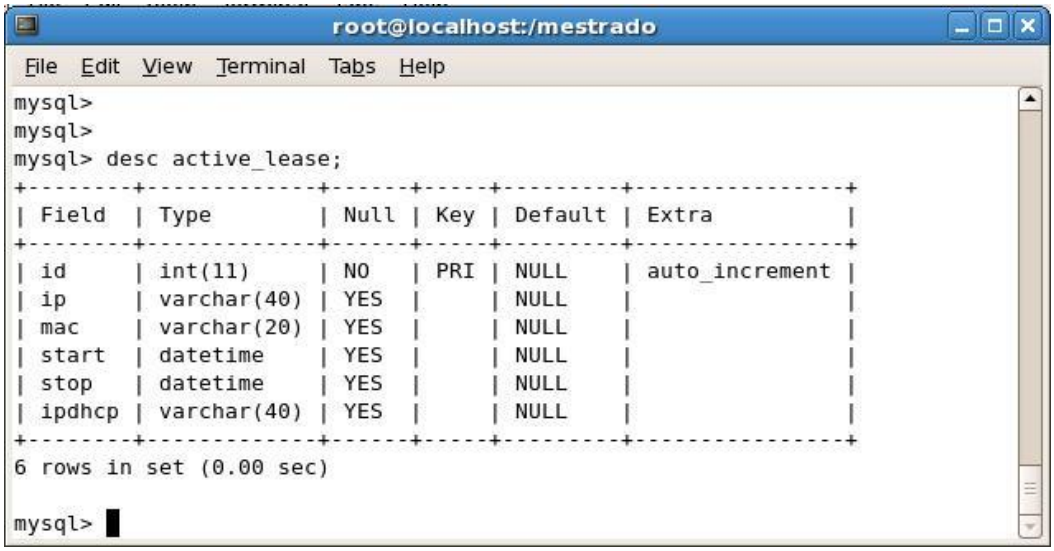


Figura 16 – Tabela active_lease

Nesta tabela são guardados os pedidos DNS *update* vindos do servidor DHCP, o nome dos campos é auto-explicativo, no entanto, o campo ip guarda o IP que o DHCP atribui ao cliente, o mac guarda o MAC do cliente, o start marca o tempo inicial da *lease* (data de criação), o stop tem o instante em que o sistema detecta que a *lease* já não está activa, por fim o ipdhcp guarda o IP do servidor DHCP.

A tabela history é criada pelo seguinte comando:

```
create table history(id int auto_increment primary key,
ip varchar(40),
 mac varchar(20),
 start datetime,
 stop datetime,
ipdhcp varchar(40));
```

Verificamos a sua criação (Figura 17):

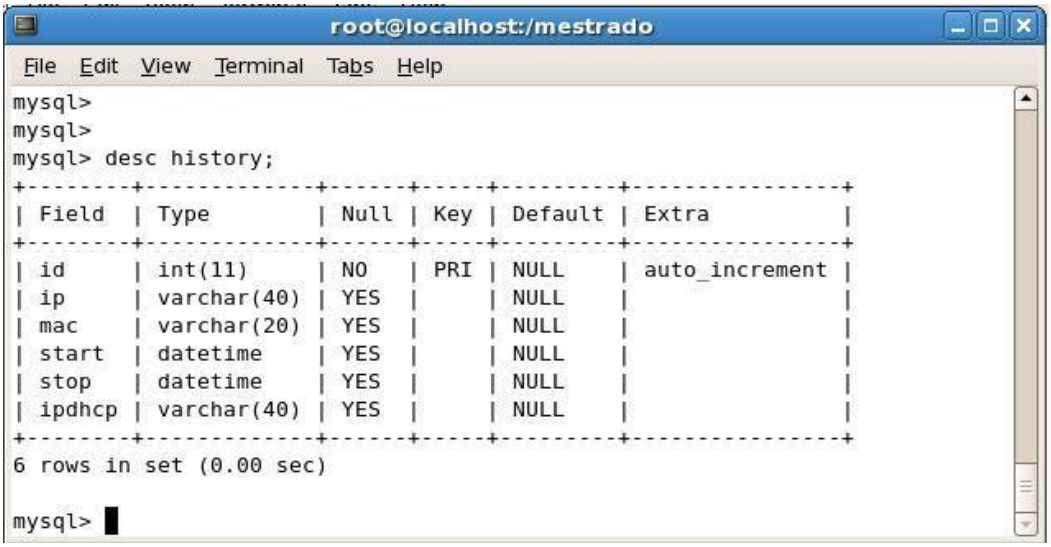


Figura 17 – Tabela History

Esta tabela é uma cópia da tabela anterior e os campos servem para albergar exactamente as mesmas informações. A diferença reside no facto da anterior servir para manter as *leases* activas, esta tem o histórico de atribuições, ou seja, o sistema quando detecta que uma *lease* expirou move-a de tabela.

A tabela arp é criada da seguinte forma:

```
create table arp(temp int,
ip varchar(40),
mac varchar(20),
iprt varchar(40));
```

Estando a sua verificação na Figura 18.

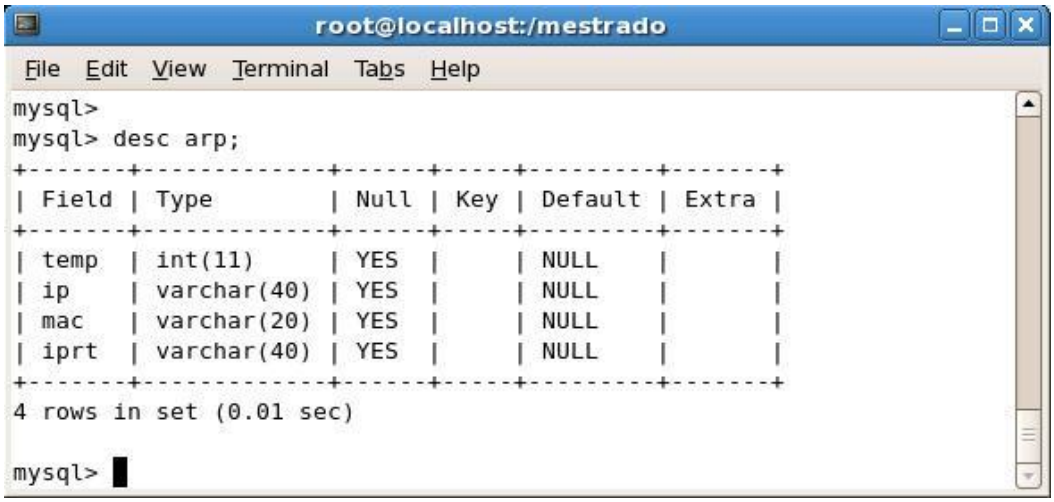


Figura 18 – Tabela arp

Esta tabela vai receber entradas ARP vindas dos equipamentos de rede em que ip e mac são o IP e MAC de um computador que está activo na rede. O campo iprt é o IP do equipamento que continha a informação. O campo temp vai ter o valor 0 ou 1 e vai servir para saber se uma entrada foi obtida na interacção corrente ou na anterior (explicado mais detalhadamente posteriormente).

A tabela exception:

```
create table exception(
ipini varchar(40),
ipend varchar(40),
mac varchar(20));
```

Sendo a sua verificação (Figura 19):

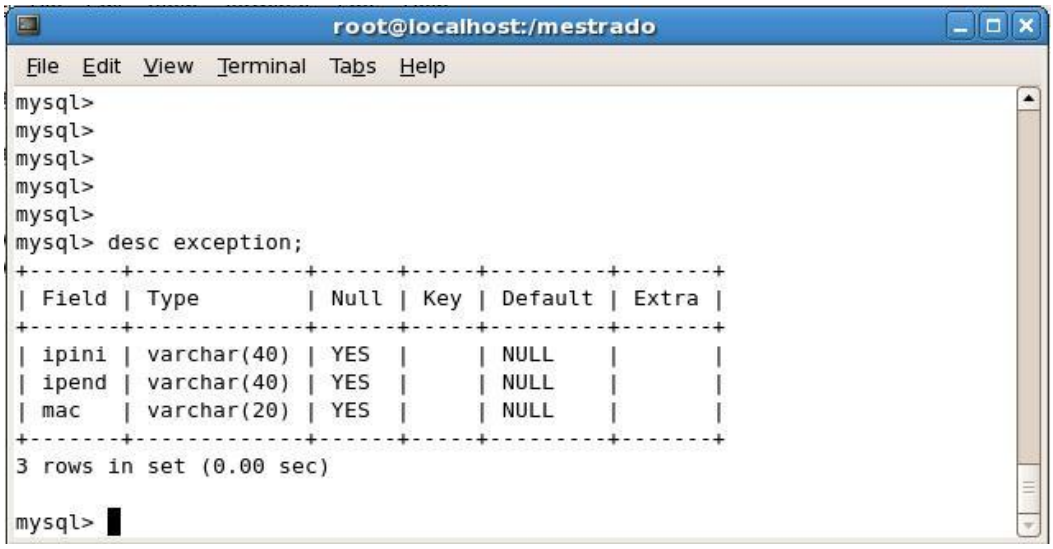


Figura 19 – Tabela exception

Esta tabela contém as excepções ao processamento da tabela ARP, isto porque como vimos anteriormente, é possível configurar manualmente o IP. O ipini guarda o primeiro endereço (em caso de ser configurada uma gama), o ipend tem o último IP se for uma gama ou um IP igual ao ipini se apenas for um IP, por ultimo o mac, este campo deverá ter o valor do MAC para a respectiva excepção (o mesmo não se aplica no caso de ser uma gama).

A tabela abnormal:

```
create table abnormal(
id int auto_increment primary key,
temp int,
ip varchar(40),
mac varchar(20),
start datetime,
stop datetime,
iprt varchar(40));
```

Verificando (Figura 20):

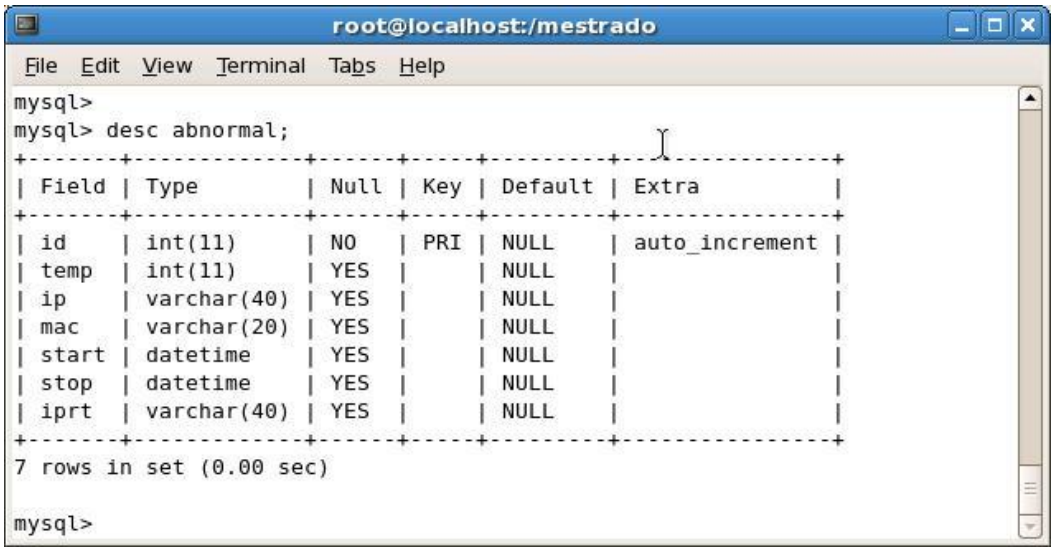


Figura 20 – Tabela abnormal

As informações sobre as situações anormais são guardadas nesta tabela, sendo que o campo temp vai ter o valor 1 ou 0 significando que aquela situação (aquela entrada especifica) está ou não fechada, ou seja, se está a acontecer neste momento. O ip e o mac são o IP e MAC da situação detectada, portanto, o IP forçado. O start e o stop têm os instantes em que foi detectado e inicio e fim da referida situação anormal.

Em seguida será apresentada a tabela users:

```
create table users(
user varchar(15),
pwd varchar(15));
```

Como podemos ver (Figura 21):

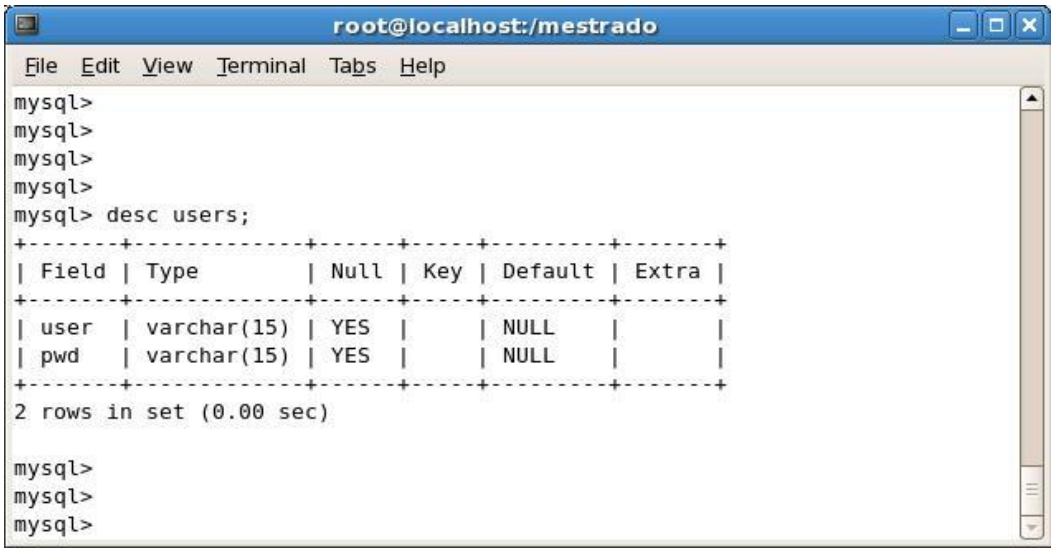


Figura 21 – Tabela users

Esta tabela pretende dar suporte à autenticação na página de gestão, por isso, de forma intuitiva o campo user guarda o nome do utilizador e o campo pwd a sua palavra-chave.

A tabela hosts:

```
create table hosts(
name varchar(100),
ip varchar(40),
mac varchar(40),
ipdhcp varchar(40),
port varchar(10)
);
```

Verificação da tabela (Figura 22):

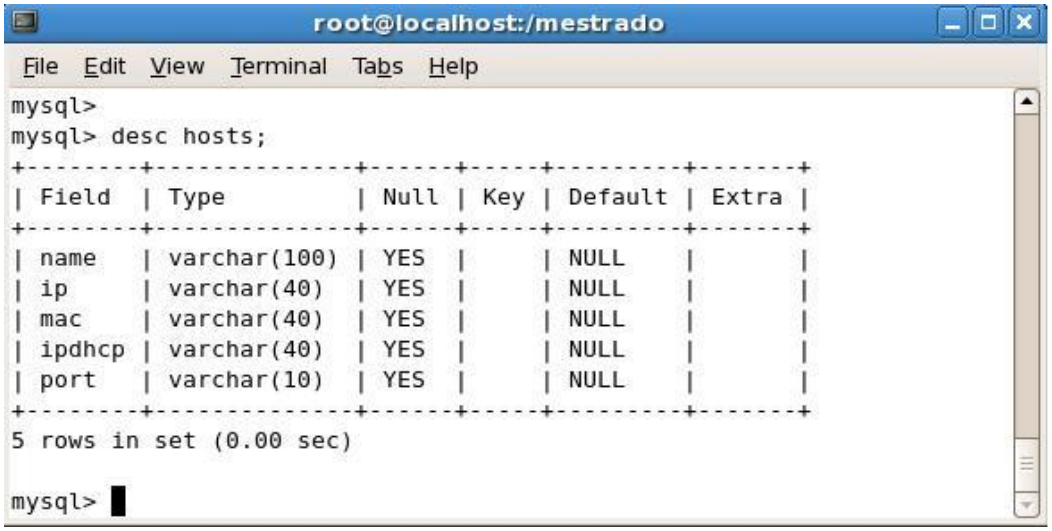


Figura 22 – Tabela hosts

As informações aqui presentes pretendem dar suporte à funcionalidade de listar as atribuições estáticas efectuadas, assim, o campo name, o ip e o mac são o nome, o IP e o MAC da atribuição. O ipdhcp e port são a identificação do servidor DHCP que recebeu a respectiva atribuição estática.

A tabela statspool:

```
create table statspool(
name varchar(40),
ipini int unsigned,
ipend int unsigned);
```

Verificação:

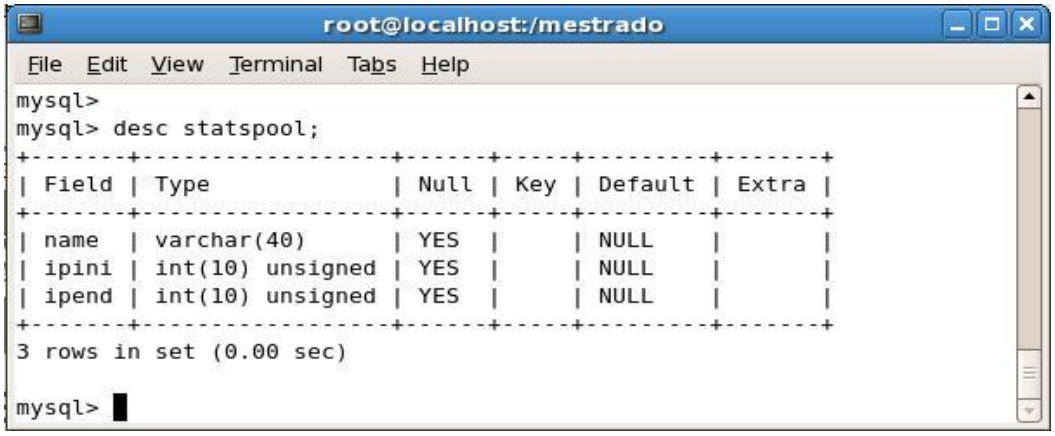


Figura 23 – Tabela statspool

Nesta tabela são guardadas as *pools* configuradas, o objectivo é replicar as *ranges* do servidor DHCP para fornecer estatísticas sobre as mesmas. É configurado um nome, um IP inicial e o final.

Por fim a tabela notification:

```
create table notification(
mail varchar(40),
ipini int unsigned,
ipend int unsigned);
```

Verificação:

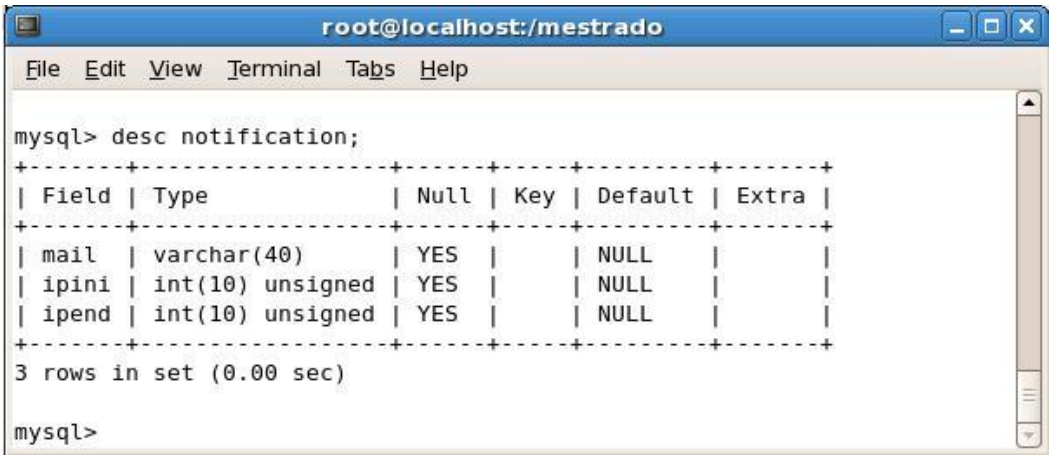


Figura 24 – Tabela notification

Nesta última tabela estão armazenados os contactos de notificação que serão usados quando forem detectadas situações de IP forçado. Para suporte o envio de *mails* para diferentes destinos é guardado o endereço, um IP inicial e final, permitindo assim o envio para diferentes destinatários consoante a range onde se encontra o IP detectado.

4.3.4. *Logs* produzidos

Em qualquer sistema é indispensável a existência de *logs* aplicacionais, por isso, este não podia ser uma excepção. Neste ponto está explicado a implementação dos *logs* com excertos de código.

Visto que qualquer parte da aplicação pode ter a necessidade de escrever informação para log foram criadas algumas funções (*WriteLog(char *, int), WriteLogSs(char *,int) SemReq()* e *SemFree())* para evitar a duplicação de código e para criar um mecanismo de protecção no acesso ao ficheiro de *log*. Começando pela *WriteLog()* e relativamente à duplicação de código seria logicamente penoso ter de abrir (*fopen()*) e fechar (*fclose()*) um ficheiro para escrita com as respectivas validações em cerca de uma centena (número de vezes que a função de escrita é invocada) de locais no código da aplicação. A opção do ficheiro poder ser aberto no início do programa e fechado no final, não é uma escolha viável pois é vulnerável a potenciais bloqueios que possam acontecer, visto que o conteúdo que enviamos (*fprintf()*) para um ficheiro não é escrito de imediato, apenas periodicamente com o enchimento das memórias temporárias ou quando fechamos o ficheiro. Devido ao facto anterior, se tal opção fosse escolhida também não seria possível analisar o ficheiro de *log* com o comando *tail –f* (de uso comum em sistemas Linux, servindo para verificar continuamente as novas entradas de um ficheiro).

```
flog=fopen(logpath,"a+");
if( flog==NULL)
{
        printf("can't open log file");
}

strtime=ctime(&timec);
strtime[strlen(strtime)-1]='\0';
sprintf(str,"%s - %s",strtime,s);
fprintf(flog,"%s\n",str);
fclose(flog);
```

Visto a aplicação ter mais que uma *thread* em execução em simultâneo é necessário proteger o acesso para escrita, bloqueando o mesmo até o recurso estar disponível. O mecanismo consiste em criar uma zona de exclusão mútua, recorrendo ao uso de funções e variáveis presentes em *semaphore.h*. Foi necessário criar uma variável global para que todas as *threads* possam aceder à mesma e executar funções que permitem reservar (*sem_wait()*) e libertar (*sem_post()*) a referida variável, não sendo possível reservar a variável duas vezes seguida, pois a segunda execução irá ficar bloqueada até que a variável seja liberta.

```
sem_wait(&ctr_log);
        ...
        //Código de exclusão mútua (verificação do log level e
        respectiva escrita)
        ...
sem_post(&ctr_log);
```

Uma característica importante é a capacidade de aumentar ou diminuir o detalhe com que uma aplicação gere um *log*, portanto foram criados três níveis de *log*, sendo o *info* (valor “3”) o menos detalhado seguindo-se o *debug* (valor “2”) e por último *o extreme debug* (valor ”1”), sendo este valor configurado na variável log_level como referido no ponto 4.3.1. Sendo a variável um inteiro é bastante simples de implementar recorrendo a um *if,* pois a função de escrita recebe por argumento o texto a escrever e a indicação a que nível pertence, logo se o nível configurado for maior ou igual é escrito em log.

```
if ( i >= LogLevel)
{
        //escrever em ficheiro ( i é um parâmetro da função)
}
```

A penúltima característica prende-se com a não separação de linhas de *log* do mesmo pedido, ou seja, quando a aplicação recebe um pedido vindo do DHCP, dependendo do nível de detalhe, podem ser escritos todos os campos do pedido, não fazendo sentido estes ficarem separados ao longo do ficheiro. Para implementar esta funcionalidade foram criadas três funções (*SemReq ()*,*WriteLogSs(char *s,int i) e SemFree()*). Neste caso, existem as funções que efectivamente controlam o semáforo (ao contrario do anterior que estão embutidas na função de escrita), assim quando existe a necessidade de fazer várias escritas é invocada a função que "fecha" o semáforo provocando que outras invocações ao mesmo fiquem bloqueadas à espera que este seja liberto. Em seguida é invocada a função de escrita em log (*WriteLogSs(char *s,int i)*) as vezes necessárias e por fim é liberto o referido semáforo para que outros acessos possam ser efectuados.

```
void SemReq (){
        sem_wait(&ctr_log);
}

void WriteLogSs(char *s,int i)
{
        //semelhante à função WriteLog,  não tendo a zona de
exclusão mútua
}

void SemFree(){
        sem_post(&ctr_log);
}
```

Por fim, o ficheiro de *log* a ser escrito não é sempre o mesmo, como verificámos no ponto 4.3.1, o nome colocado na configuração serve apenas como prefixo do nome completo, que compreende também a data, desta forma é gerado um ficheiro de log por dia.

```
sprintf(logpath,"%s%d%.2d%d.log",logp,timet->tm_year+1900,timet-
>tm_mon+1,timet->tm_mday);

flog=fopen(logpath,"a+");
```

4.3.5. Interacção com a base de dados

Ao longo da aplicação existe a necessidade de interacção com a base de dados. Para maior abstracção da aplicação em relação aos pormenores da criação da ligação, pedidos e respostas desta, foram criadas algumas funções. A *MYSQL * NewConn()* tem o objectivo de instanciar as ligações à base de dados, realizando as devidas verificações e devolvendo uma variável de contexto para a utilização nas restantes funções.

```
MYSQL * NewConn(){
        MYSQL *conn;
        char str[512];
        ...
        if(!mysql_real_connect(conn,MYSQL_IP,user,password,database,0,NULL,0))
        {
                WriteLog("Can't connect to DB",3);
                sprintf(str,"\n2-%s", mysql_error(conn));
                WriteLog(str,3);
        }
        ...
        return(conn);
}
```

A função *int DoQueryI(MYSQL *conn,char *strq)* recebe como argumento uma ligação e uma *string* que tem a *query* pretendida e devolve um inteiro com o resultado da operação, sendo que o valor zero (0) significa sucesso e um (1) insucesso.

```
int DoQueryI(MYSQL *conn,char *strq){
        MYSQL_ROW row;
        MYSQL_RES *res;
        char str[512];
        if(mysql_query(conn,strq)!=0)
        {
                sprintf(str,"cant do query - %s",mysql_error(conn));
                WriteLog(str,3);
                return(1);
        }
        else
        {
                sprintf(str,"inserted - %s",strq);
                WriteLog(str,2);
                return(0);
        }
}
```

A função *int DoQueryCt(MYSQL *conn,char *strq)* é bastante semelhante à anteriormente descrita, com a diferença no valor de retorno. Esta, retorna menos um (-1) em caso de erro ou o valor do primeiro campo da primeira linha. Esta função é útil quando é preciso saber o valor de uma contabilização, saber se uma determinada condição já existe na base de dados.

```
int DoQueryCt(MYSQL *conn,char *strq){
	MYSQL_ROW row;
	MYSQL_RES *res;
	char str[512];
	int i;
	if(mysql_query(conn,strq)!=0)
	{
		sprintf(str,"cant do query - %s",mysql_error(conn));
		WriteLog(str,3);
		return(-1);
	}
	else
	{
		res=mysql_store_result(conn);
		row=mysql_fetch_row(res);
		i=atoi(row[0]);
		sprintf(str,"Quey to check if IP/MAC exist - %s: already
in-%d",strq,i);
		WriteLog(str,1);
		mysql_free_result(res);
		return(i);
	}
}
```

Ao longo do desenvolvimento não se verificou um uso frequente destas funções, com excepção da *NewConn* pois as interacções com a base de dados revelaram ser bastante diferentes, desde inserções, consultas (com objectivos diferentes) ou apenas para eliminar, tornando-se complicado criar funções genéricas com o objectivo de serem reutilizadas.

4.3.6. Recepção de pedidos DNS update

Neste ponto é apresentada a forma como foi implementada esta funcionalidade. A intenção, como já explicado, é de certa forma simular um servidor DNS, pois do ponto de vista do servidor DHCP é exactamente o que esta aplicação é (esta interacção). Da pesquisa realizada, não foi possível encontrar funções que fizessem o pretendido de forma prática, por isso, a opção foi implementar de raíz. A recepção do pedido está em si implementada na *thread* principal *main()* num ciclo infinito com um *while()* pois sempre que a aplicação está ligada deve estar à escuta de pedidos.

```
void main()
{
        ...
   while( 1 )
   {
        //Código relativo a recepção do pedido
   }
        ...
}
```

Quando falamos de uma rede, nada impede que duas ou mais pessoas liguem os seus computadores ao mesmo tempo provocando dois pedidos DHCP, por sua vez este irá fazer dois DNS *update*, havendo também a possibilidade de existirem vários servidores DHCP na rede, torna-se, assim, óbvio que esta aplicação tem de suportar um processamento *multithread*, não podendo estar esta indisponível para receber um pedido por estar a fazer o tratamento de outro. Por este motivo, após ser recebido um pedido este é copiado para outra estrutura, estrutura esta que é passada por argumento no lançamento da *thread* que executa a função responsável pelo seu tratamento.

```
struct DataTh
{
int bytes_read,sin_len,sock;
char *buffer;
struct sockaddr_in  sin;
};
```

Quando é efectuado o lançamento de uma *thread* não é possível enviar o número de parâmetros superior a um directamente, pois só é aceite um. Para contornar este problema foi criada a estrutura, referida no excerto acima, que contém os dados necessários para o processamento do pedido na *thread*. O campo bytes_read tem o número de bytes lidos, ou seja, o tamanho do campo buffer que contém o pedido em si. Os restantes três campos são utilizados na resposta ao servidor DHCP. No código a seguir é demonstrado a alocação dinâmica de memória para a estrutura que será copiada para a *thread* e o lançamento desta.

```
while( 1 )
   {
            dth=malloc(sizeof(struct DataTh));
           bytes_read = recvfrom( sock, buffer, sizeof( buffer ), 0, (
struct sockaddr * ) &sin, &sin_len );
            dth->buffer = malloc(bytes_read);

            for(i=0;i<bytes_read;i++)
                  dth->buffer[i]=buffer[i];

            dth->bytes_read=bytes_read;
            dth->sin_len=sin_len;
            memcpy(&dth->sin,&sin,sizeof(struct sockaddr_in));
            dth->sock=sock;

            pthread_create(&th,NULL,&TreatRequest,dth );
            pthread_detach(th);
   }
```

Para se conseguir implementar a função que faz propriamente o processamento é necessário analisar o RFC 1035 (*Domain Names – Implementation and Specification*) e o 2136 (*DNS Update*) complementando com a verificação de um *trace* de rede feito com o comando *tcpdump* que produz um ficheiro contendo pacotes que foram transmitidos, sendo este aberto no programa Wireshark.

A *thread* tem no seu início um *buffer* com o pedido, mas é necessário saber a posição no *buffer* onde começa e acaba cada campo, portanto a ideia foi construir as estruturas necessárias (com os mesmos campos que o pedido DNS *update*, com excepção nos campos de tamanho variável) e colocar os ponteiros destes nos locais correctos do *buffer*, desta forma podemos aceder aos campos que pretendemos pois já os conseguimos referir. O pedido DNS *update* segundo o RFC 2136 tem a seguinte constituição:

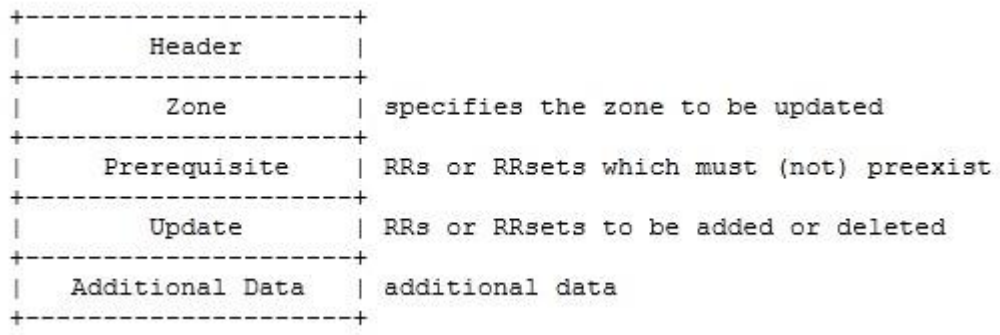


Figura 25 – Constituição DNS *update*

Como podemos verificar o pacote é constituído por secções bem limitadas, sendo necessário conhecer os campos das referidas zonas para os podermos registar em log e para extrairmos o IP e MAC nele contidos. O *header* é o seguinte:

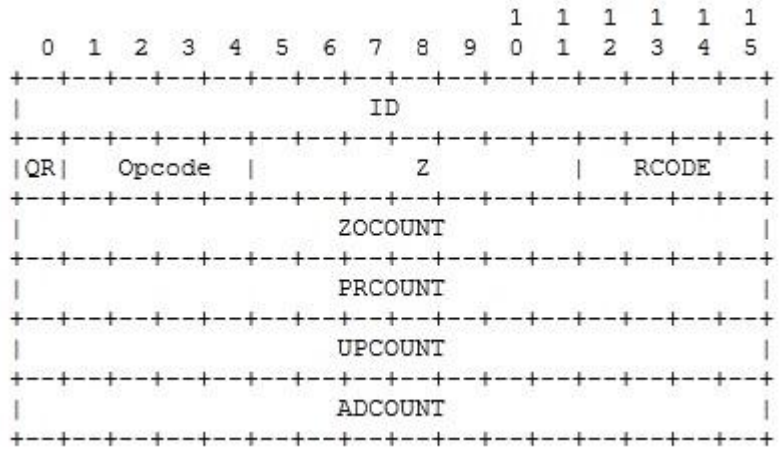


Figura 26 – Secção *header*

Sendo a classe construída respectiva a seguinte:

```
struct DNSH
{
        unsigned short id;

        unsigned char rd :1;    //recursion desired
        unsigned char tc :1;    //truncated message
        unsigned char aa :1;   // authoritative answer
        unsigned char opcode :4; //purpose of message
        unsigned char msgt :1;          //query response type

        unsigned char rcode :4;         //response code
        unsigned char cd :1;   //checking disabled
        unsigned char ad :2;   //authenticated date
        unsigned char ra :1;    //recursion available

        unsigned short qdcount;
        unsigned short ancount;
        unsigned short nscount;
        unsigned short rrcount;
};
```

Algo importante a referir, como é possível verificar, os campos não estão pela mesma ordem que o RFC, isto deve-se ao facto de a representação em memória *ser little-endian* (os bits num byte estão por ordem inversa).
Pelo mesmo RFC (2136):

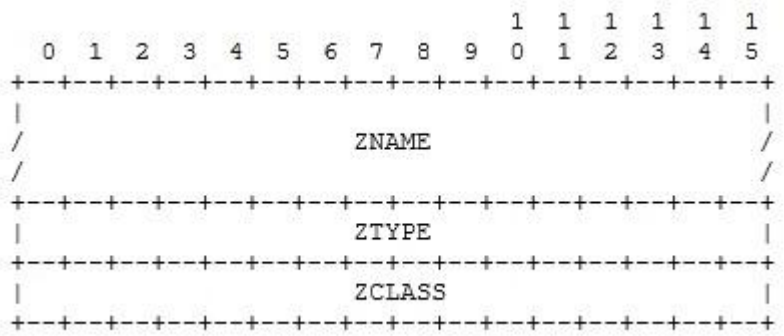


Figura 27 – Secção zone

Sendo a estrutura correspondente a seguinte:

```
struct ZONE
{
        unsigned short qtype;
        unsigned short qclass;
};
```

Como podemos verificar a estrutura apenas tem dois campos contra os três indicados no RFC, isto deve-se, como indicado anteriormente, ao facto do primeiro campo ser dinâmico e por esse facto ser tratado fora da estrutura.

Da mesma forma obtemos as seguintes estruturas para a secção de *prerequisite* e *update*:

```
struct PREREQUISITES
{
        unsigned short ptype;
        unsigned short pclass;
        unsigned int   ttl;
        unsigned short dlength;
};
```

```
struct UPDATE
{
        unsigned short utype;
        unsigned short uclass;
        unsigned int   ttl;
        unsigned short dlength;
};
```

Após a apresentação da forma como os campos são acedidos e as estruturas utilizadas será mais fácil entendermos a função *void *TreatRequest(void *argv)* (que a *thread* executa). Como indicado, tendo os ponteiros das estruturas no local correcto do *buffer* seria possível aceder aos campos sendo a primeira atribuição feita da seguinte forma:

```
        ...
struct DNSH *dns=NULL
        ...
dns = ( struct DNSH *) dth->buffer;
        ...
```

Neste momento, é possível aceder aos campos presentes no *header* do pedido fazendo dns->nome_campo. Para efeitos de *log* (revelando-se uma ajuda bastante importante no *debug* durante a realização da função) todos os campos das várias secções são acedidos e escritos no respectivo ficheiro, tendo sido criadas funções para cada secção. Será apresentada apenas a função *void PrintHeader(struct DNSH *header)*, pois as outras são em tudo semelhantes. Esta função recebe um ponteiro que representa o inicio da estrutura no *buffer*, assim basta aceder ao campo pretendido tendo em atenção o tipo do campo, se for um inteiro é necessário usar a função *ntohs(header->id)* para que o valor seja mostrado correctamente (como explicado anteriormente sobre a ordem dos bits), se for apenas um bit basta fazer um teste lógico, header->msgt ? '1': '0', desta forma se o bit for um (1) é escrito um *char* com o valor um (1) se o bit for zero (0) é escrito um *char* com valor zero (0), por fim, quando o campo é constituído por um conjunto de bit é usada a macro *#define IsBitSet(val,bit) ((val) & (1 <<(bit)))*, isto significa que onde estiver escrito *IsBitSet(val,bit)* é substituído ((val) & (1 <<(bit))). Estes operadores lógicos permitem testar qual o valor dos bits sendo um dos argumentos a posição, por exemplo:

- Sendo val =1010 e bit=2 é verificado 1010 & 100 sendo igual a 0000=false
- Sendo val =1010 e bit=1 é verificado 1010 & 10 sendo igual a 0010=true

De forma semelhante ao caso anterior é possível *fazer IsBitSet(header->opcode,3)?'1':'0'*.

```
sprintf(str,"id - %d",ntohs(header->id));
WriteLogSs(str,1);

sprintf(str,"msgt - %c",header->msgt ? '1': '0');
WriteLogSs(str,1);

sprintf(str,"opcode - %c%c%c%c", IsBitSet(header-
>opcode,3)?'1':'0',IsBitSet(header->opcode,2)?'1':'0',IsBitSet(header-
>opcode,1)?'1':'0',IsBitSet(header->opcode,0)?'1':'0');
WriteLogSs(str,1);
```

A colocação dos ponteiros seguintes é semelhante com a diferença da necessidade de somar as secções já escritas ao ponteiro inicial e verificar a quantidade de secções existentes no pedido.

```
        ...
dns = ( struct DNSH *) dth->buffer;
        ...
qname = dth->buffer+sizeof(struct DNSH)+1;
        ...
zone = (struct ZONE *) &dth->buffer[sizeof(struct
DNSH)+2+strlen(qname)];
        ...
for(j=0;j<ntohs(dns->ancount);j++){
        PrintPre(pre);
        offset=offset+ntohs(pre->dlength)+sizeof(struct
PREREQUISITES);
        pre=(struct PREREQUISITES *) &dth->buffer[sizeof(struct
DNSH)+3+strlen(qname) + sizeof(struct ZONE)+st
rlen(pname)+offset];
            }
```

Desta forma conseguimos ir construindo o pedido enviado para obter os valores pretendidos MAC e IP, no entanto é ainda necessário verificar o campo update->utype para saber se o pedido traz efectivamente um endereço IP (tipo A *Host Address)*, após isto para conseguirmos escrever o IP correctamente é necessário efectuar algumas operações. Ao termos um ponteiro para o início do IP é necessário copiar o número de bytes que está no campo length, ficando

assim com o IP numa string, mas não no formato reconhecido por nós. Foi necessário recorrer à estrutura sockaddr_in (que usa a in_addr) e à função inet_ntoa().

```
struct sockaddr_in a;
        ...
if(ntohs(update->utype)==1){
        pointer = dth->buffer+sizeof(struct DNSH)+strlen(qname) +
        sizeof(struct ZONE) + strlen(pname) + sizeof(struct
        PREREQUISITES)+ sizeof (struct UPDATE)+1;

        for (i=0; i<ntohs(update->dlength) ;i++){
                aname[i]=pointer[i];
         }
        long *p;
        p=(long *) aname;
        a.sin_addr.s_addr=(*p);
        sprintf(str,"IPv4= %s - %s",inet_ntoa(a.sin_addr),mac);
        ...
}
```

Neste momento, já é conhecido o MAC e o IP sendo necessário introduzi-los na tabela active_lease da base de dados, conforme o sucesso desta operação é elaborada uma resposta ao servidor DHCP.

A inserção na base de dados é feita recorrendo a funções mencionadas no ponto 4.3.5, a DoQueryCt(conn,str) e a DoQueryI(conn,str). A inserção não pode ser efectuada sem que antes tenha sido verificado se esta já existe na base de dados, pois o servidor DHCP também envia o mesmo pedido quando o cliente renova o tempo da *lease*. Desta forma é verificada a sua existência e apenas inserida se não existir, significando isto que é efectivamente uma nova *lease* ou que no passado a respectiva inserção falhou. A verificação é feita com a função *DoQueryCt(conn,str*) que devolve (com a *query* realizada) uma contabilização de registos da tabela active_lease que tenham o IP e MAC iguais ao pedido, se o valor for superior a zero (0) (será o valor um), não é feita a inserção pois já existe, se for igual a zero é feita a inserção. A resposta para o servidor DHCP é marcada com sucesso se já existir ou for inserida com sucesso e é feita colocando o campo rcode a zero, em caso de erro o rcode é igual a 2 (*servfail*).

```
if(DoQueryCt(conn,str)==0)
                {
                        sprintf(str,"insert into active_lease
values(null,\"%s\",\"%s\",now(),null,null);",inet_ntoa(a.sin_addr),mac);
                        if(DoQueryI(conn,str))
                        {
                                WriteLog("Response with rcode: 2
(insuccess)",3);
                        dnsr->rcode=2;
                        }
                        else
                        {
                        dnsr->rcode=0;
                        WriteLog("Response with rcode: 0 (success)",3);
                        }
                }
                else
                {
                        dnsr->rcode=0;
                }
```

Tendo em conta o RFC 2136 a resposta pode ser feita de duas formas, na implementação foi escolhida a mais simples, ou seja, é feita com base no *header* enviado, em que é necessário copiar o id, qr, opcode, preencher o rcode da forma acima indicada e os restantes campos apenas têm zeros.

```
dnsr->id=dns->id;
dnsr->msgt=1;
dnsr->opcode=dns->opcode;
WriteLog("\nResponse:",1);
sprintf(str,"id = %d\nmsgtype - %c\n opcode - %c%c%c%c\n rcode -
%d",dnsr->id,dnsr->msgt ? '1': '0',IsBitSet(dnsr-
>opcode,3)?'1':'0',IsBitSet(dnsr->opcode,2)?'1':'0',IsBitSet(dnsr-
>opcode,1)?'1':'0',IsBitSet(dnsr->opcode,0)?'1':'0',dnsr->rcode);
WriteLog(str,1);
```

Para terminar este ponto basta fazer o envio da resposta para o servidor sendo isto feito com o comando *sendto()*.

```
if( sendto(dth->sock, (char*) bufferr,sizeof(struct DNSH),0, ( struct
sockaddr * ) &dth->sin, dth->sin_len ) < 0 )
        WriteLog("Error sending response",3);
else
         WriteLog("Response sent",3);
```

4.3.7. Consulta via OMAPI de uma *lease*

Neste ponto é abordada a forma como a consulta no servidor DHCP de uma *lease* é feita. Para atingir este objectivo foram criadas duas funções, a *int LeaseCheck(char *strip, char *strmac, char *stripdhcp,int portdhcp)* que faz a implementação da consulta propriamente dita recebendo o IP e MAC a serem testados e o IP e PORT do servidor DHCP que será verificado. A *int LeaseCheckTt(char *strip, char *strmac)* é a função invocada para saber o estado de uma *lease* pois esta invoca a anterior o número de vezes necessário para que todos os servidores DHCP sejam testados, sendo implementado da seguinte forma:

```
        ...
for(i=0;i<dhcp_n;i++){
        j=LeaseCheck(strip,strmac,DHCP_IP[i],(int)DHCP_PORT[i]);
        ...
```

Para obter a informação de uma *lease* são efectuados os seguintes passos (estes passos foram realizados recorrendo à biblioteca dhcpctl):

- Ligação ao servidor
- Criar um objecto local
- Preencher no objecto local as características de procura
- Pedir o resultado
- Verificar o estado
- Fechar Ligação

Assim a ligação ao servidor é realizada da seguinte forma:

```
        ...
dhcpctl_handle conct = NULL;
isc_result_t status;
        ...
status=dhcpctl_connect(&conct,stripdhcp,portdhcp,0);
        ...
```

Esta função (*dhcpctl_connect*) cria uma ligação ficando esta representada pela variável conct, sendo esta utilizada posteriormente nas outras funções necessárias para a verificação.

Em seguida é criado um objecto local:

```
        ...
dhcpctl_handle lease = NULL;
isc_result_t status;
        ...
status=dhcpctl_new_object(&lease,conct,"lease");
        ...
```

Desta forma é criado um objecto local que serve para podermos preencher com informação, informação esta que é utilizada pelo DHCP para encontrar o objecto remoto. A informação colocada no objecto local tem de gerar uma associação unívoca pois apenas é suportado o retorno de um objecto. Este preenchimento local é realizado da seguinte forma:

```
        ...
dhcpctl_data_string ipaddrstring = NULL
        ...
memset(&ipaddrstring, 0,sizeof(ipaddrstring));
inet_pton(AF_INET,strip,&convaddr);
status=omapi_data_string_new(&ipaddrstring,4,MDL);
        ...
omapi_data_string_new(&mac,sizeof(struct ether_addr),MDL);
memcpy(mac->value,ether_aton(strmac),sizeof(struct ether_addr));
status=dhcpctl_set_value(lease,mac,"hardware-address");
        ...
```

Neste momento é possível pedir que seja feita a associação entre objecto local e remoto:

```
        ...
status=dhcpctl_open_object(lease,conct,0);
        ...
status=dhcpctl_wait_for_completion(lease,&status2);
        ...
```

A primeira função coloca numa *queue* a mensagem a ser enviada ao servidor DHCP e a última envia a mensagem ficando bloqueada até receber uma resposta.

Após obter a resposta, é preciso verificar a informação que precisamos, sendo esta o estado da *lease* (campo state), para isso é feito o seguinte:

```
        ...
int i;
        ...
status=dhcpctl_get_boolean(&i,lease,"state");
        ...
```

Tendo o valor do estado na *lease* basta fazer uma comparação com os estados pretendidos e retornar 0 se estiver activa, 1 se não estiver activa ou se houver o erro "*not found*", se forem retornados os erros "*key conflict*" ou "*more than one object match to the key*" irá ser invocada a função *LeaseCheckIP* (motivo indicado no ponto 2 e 3) em tudo semelhante à já apresentada tendo a diferença de fazer a verificação da *lease* apenas com o IP (sendo feitas posteriores validações sobre o estado da *lease* e ao valor *hardware-address*) ou -1 se houver outro erro na consulta ao DHCP.

Como é possível verificar, as funções *dhcpctl* devolvem uma variável do tipo *isc_result_t*, assim foi criada uma função que faz a verificação desta variável escrevendo em *log* o sucesso ou não da operação e devolve zero se sucesso ou um se insucesso (esta função é invocada após cada função *dhcpctl* anteriormente mostrada para testar a variável status continuando ou não o processo de verificação).

```
int CheckStatus(isc_result_t status,char *strq){

        char str[512];
        if(status!=ISC_R_SUCCESS){
                sprintf(str,"Error %s -
%s",strq,isc_result_totext(status));
                WriteLog(str,3);
                return(1);
        }
        else{
                sprintf(str,"%s with success\n",strq);
                WriteLog(str,2);
                return(0);
        }
}
```

Para terminar é necessário fechar a ligação efectuada ao servidor, tal foi apenas descoberto no segundo momento de testes (ponto 5), ou seja, quando a aplicação foi submetia a uma grande quantidade de pedidos, situação causada pela omissão na documentação consultada de tal necessidade ou existência de função. Assim qualquer ramo da função *LeaseCheck* terá o seguinte conjunto de entradas para fechar a ligação e libertar recursos criados.

```
        ...
dhcpctl_data_string_dereference(&ipaddrstring,MDL);
dhcpctl_data_string_dereference(&mac,MDL);
omapi_disconnect(conct -> outer -> outer, 1);
omapi_object_dereference(&conct, MDL);
        ...
```

4.3.8. Processamento das *leases* activas

O processamento das *leases* activas consiste em verificar todas as *leases* presentes na tabela active_leases (estas, como visto, são inseridas sempre o servidor DHCP cria uma *lease*). Este processamento é feito na função *void *LeaseCheckTh(void *argv)* e consiste em verificar entrada a entrada se o par IP e MAC ainda existe activo no servidor DHCP, se existir é feito o processamento da próxima entrada, se não existir é feito um *update* nessa entrada sendo colocado no campo stop a data e hora desse instante. Quando todas as entradas forem testadas, aquelas que estão expiradas são passadas para a tabela history (Figura 28).

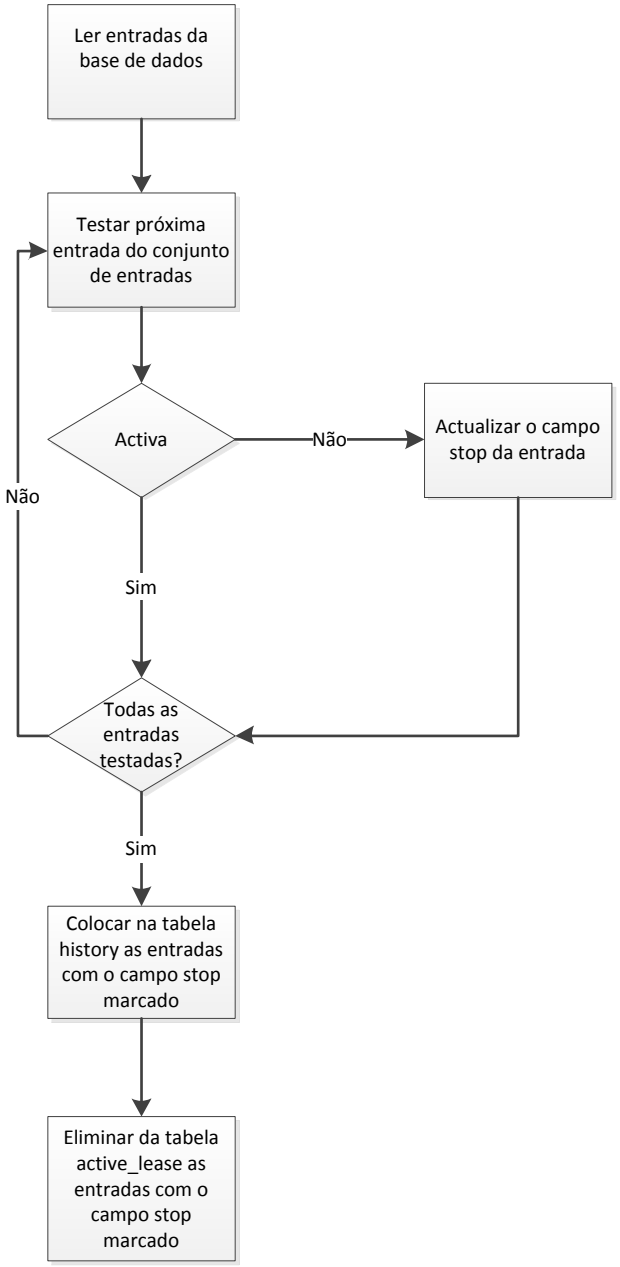


Figura 28 – Fluxo do processamento das *leases* activas (uma interacção)

Para ler os dados da base de dados são utilizadas funções em *mysql.h*. Inicialmente é criada uma nova ligação e associado a esta é realizada uma *query* que selecciona todos os IP e MAC.

```
        ...
conn=NewConn();
if(mysql_query(conn,"select ip,mac from active_lease")!=0)
        ...
res=mysql_use_result(conn);
        ...
```

Em seguida são testado todos os IP e MAC invocando a função anteriormente explicada *int LeaseCheckTt(char *strip, char *strmac)* sendo o retorno desta o argumento de um *switch*.

```
        ...
while((row=mysql_fetch_row(res)) !=NULL){
        switch(LeaseCheckTt(row[0],row[1])){
                case -1: //tratamento do erro
                        ...
                case 1: //tratamento de lease não activa
                        ...
                default: //tratamento de lease activa
                        ...
}
```

No case -1 é efectuado o tratamento do erro, sendo necessário verificar a valor da variável on_dhcp_error_l, pois esta variável decide o comportamento da aplicação em caso da verificação da *lease* originar um erro não esperado (como indicado no ponto 4.3.1). Se na configuração estiver o valor um, a aplicação assume, em caso de erro, que a *lease* não existe, sendo por isso aplicado o respectivo processamento. Por outro lado, se o valor for zero é assumido que a *lease* existe não sendo retirada da tabela active_lease.

```
switch(on_dhcp_error_l){
        case 1:
        sprintf(str,"update active_lease set stop=now() where ip=\"%s\"
and mac=\"%s\"",row[0],row[1]);
        if(DoQueryI(conn2,str))
                //escreve em log que houve a falha a colocar o stop
          else
                //escreve em log o sucesso da operação
        Break;
        case 0:
                //escreve em log o erro e que não foi retirado
        break;
        default:
                //Indicação de configuração errada
        break;
}
```

No *case* 1 encontra-se o processamento da *lease* não activa, sendo em tudo semelhante ao caso anterior se assumirmos que a variável on_dhcp_error_l tem o valor um. Por fim, o *default* apesar de significar que faz o tratamento de qualquer valor não incluído nos *cases*, apenas fará o tratamento do valor 0 que significa *lease* activa (pois os outros valores possíveis são os cases) sendo escrito apenas em *log* que a *lease* existe.

Assumindo que neste momento todas as entradas da tabela foram verificadas, aquelas cujo campo stop está preenchido têm de ser colocadas na tabela history pois são *leases* que já não estão activas.

```
        ...
sprintf(str,"insert into history select * from active_lease where stop is
not null");
if(DoQueryI(conn2,str))
        ...
```

Por fim, as entradas copiadas são eliminadas:

```
        ...
sprintf(str,"delete from active_lease where stop is not null");
if(DoQueryI(conn2,str))
        ...
```

Esta função é em si um ciclo *while(1)* que repete os passos acima descritos, sendo importante referir que entre interacções é feita uma pausa temporal, este tempo é configurado na variável lease_check_cicle.

```
        ...
sleep(lease_check_cicle);
        ...
```

4.3.9. Normalização do formato dos endereços MAC

A normalização do formato dos endereços MAC é bastante importante, pois se vamos ter de realizar comparações de endereços provenientes de múltiplas fontes é necessário garantir que têm o mesmo formato.

Constatou-se que o formado usado para representar os endereços MAC (48bit) no servidor de DHCP e na recolha da tabela de *Address Translation* (ARP) pela ferramenta de SNMP eram distintos, sendo:

- 8:0:27:0:cc:e1 (recolhido pela ferramenta de SNMP)
- 08:00:27:00:cc:e1 (na OMAPI com o servidor DHCP)

O formato escolhido foi o segundo (servidor DHCP). A diferença resume-se ao facto de no primeiro formato serem omissos os caracteres 0 de maior peso em cada byte representado em hexadecimal.

4.3.10. Processamento das entradas ARP

O processamento desta parte da aplicação não está centralizado apenas na componente desenvolvida em C, mas, também, num *script* desenvolvido em *shell script*. A opção de fazer este componente fora da aplicação teve em consideração o facto dos equipamentos de rede poderem evoluir e terem OID diferentes, terem formatos diferentes sendo este facto tratado muito mais facilmente num *script* em *shell* do que num programa em C pois iria provavelmente ter a necessidade de alterações de código e recompilação, pelo menos para efectuar o *parsing* numa possível alteração no output enviado via SNMP. Este facto é importante pois a revisão mais recente, o RFC 4293, estas entradas já são guardadas noutro OID, o *ipNetToPhysicalType*, sendo aqui guardada também a associação IP MAC em IPv6. Este Shell script tem o objectivo de colectar a tabela ARP dos equipamentos configurados no seu ficheiro de configuração. Este ficheiro é bastante simples tendo apenas a informação dos endereços IP que devem ser usados para ser feita a *query* SNMP e o OID a usar.

```
#format should be "router=value[ value ... value]"
router=192.168.56.100 192.168.55.110
oid=IP-MIB::ipNetToMediaPhysAddress
```

O *script* em si é também simples, faz a leitura do ficheiro de configuração para obter os endereços IP e o OID seguindo-se a execução de um ciclo *for* para todos os endereços IP fornecidos. Dentro deste *for* é executado o comando *snmpbulkwalk* do pacote *net-snmp* que irá buscar a sub-árvore pedida (OID). O *output* gerado pelo comando após cada interacção é concatenado num ficheiro arpwritetemp (antes do *for* este ficheiro é limpo) sendo no final de todas as interacções copiado para o ficheiro arpread servindo isto para evitar problemas de concorrência no acesso ao ficheiro. Se tal não fosse feito, facilmente haveria situações em que o ficheiro seria lido pela aplicação sem que o script tivesse terminado a leitura SNMP de todos os equipamentos. Esta situação, se não fosse analisada, iria provocar graves erros de funcionamento da aplicação, por exemplo, assumir que situações anormais teriam terminado. Tendo em consideração o referido, a aplicação principal apenas lê o ficheiro arpread, pois este está sempre completo.

Este *script* é configurado na *crontab*, sendo a periodicidade de execução da responsabilidade desta. A *crontab* é um programa existente em Linux (e não só) que faz a gestão de tarefas, ou seja, permite figurarmos de quanto em quanto tempo é que um *script* é executado. Para configurarmos é necessário executar o comando "crontab –e" e respeitar o formato exigido, sendo este o seguinte:

- `mm hh dd MM ss script (Geral)`

Cada parâmetro tem um significado, o primeiro é o minuto que o script irá ser executado, o segundo é a hora, o terceiro é o dia do mês, o quarto é o mês, o quinto é o dia da semana e por último o script a ser executado. Quando colocado apenas o carácter '*' é porque esse campo não tem interesse (é sempre executado).

- `*/5 * * * * /scripts/get_arp.sh (Exemplo concreto)`

Este exemplo significa que o *script* irá correr todos os meses, todos dias a todas as horas de cinco em cinco minutos.

Após conhecido o *shell script* é mostrada a componente em C que faz a leitura do referido ficheiro e o processamento das entradas. A responsabilidade deste processamento é da função *void *ArpProcess(void *argv)* que corre infinitamente numa *thread*. Esta função divide-se em secções muito concretas, concretizadas na sua maioria sob forma de outras funções. Inicialmente são lidas as entradas ARP de um ficheiro e inseridas na base dados, seguindo-se a retirada das entradas que constituem excepções ao normal processamento (evitando assim falsos alertas). Posteriormente é renovado o tempo das situações anormais e as novas entradas são verificadas. Por fim, é feita uma rotação das entradas da tabela ARP (Figura 29).

Figura 29 – Interacção do processamento das entradas ARP

4.3.10.1. Leitura e inserção entradas ARP

A leitura das entradas do ficheiro é feita uma a uma para fazer a devida inserção na tabela. Após a abertura do ficheiro é lida uma linha com a função *fgets(),* sendo esta (linha) colocada numa variável que servirá de *input* para a função *sscanf()*. Esta função permite, com uma máscara, extrair conteúdo directamente para variáveis, assim temos acesso ao par IP MAC e ao IP do equipamento que contém essa entrada. Em seguida é feita a normalização do formato do MAC e é inserida a informação na base dados sendo colocada a *query* numa string e invocada a função *DoQueryI()*. Um ponto importante de referir é o primeiro parâmetro do comando *insert* com o valor '1', significando que é uma entrada nova (será explicado o momento em que este campo terá o valor '0').

```
while(fgets(str,1024,fconfig)!=NULL){
        sscanf(str,"%s %s %s",&strip,&strmac,&striprt);
        MacNormalize(strmac,strmacp);
        sprintf(str,"insert into arp
values(1,\"%s\",\"%s\",\"%s\");",strip,strmacp,striprt);
        DoQueryI(conn,str);
}
```

4.3.10.2. Processamento das excepções

Este processamento sofreu alterações tendo sido alterada a sua implementação inicial. Inicialmente para fazer este processamento era invocada a função *ExProcess()* que faz a leitura de todas as novas entradas e invoca a função *int ExCheck(char *strip, char *strmac)* sendo passado por parâmetro um par IP e MAC (esta função seria chamada um vez por cada entrada).

```
if(mysql_query(conn,"select * from arp where temp=1")!=0)
        ...
while((row=mysql_fetch_row(res)) !=NULL){
        if(!ExCheck(row[1],row[2])){
                //Faz o delete e respectivas validações
        }
}
```

As excepções podem ser de dois tipos, um range estando configurado um IP inicial e final (diferentes) ou um IP tendo um MAC associado. A verificação feita tem de ter em atenção este aspecto, portanto a função *ExCheck* que recebe um par IP MAC verifica se este está em alguma excepção. Esta comparação era feita decompondo a *string* IP em quatro inteiros e estes eram comparados sequencialmente.

```
sscanf(row[0],"%d.%d.%d.%d",&ipi1,&ipi2,&ipi3,&ipi4);
sscanf(row[1],"%d.%d.%d.%d",&ipf1,&ipf2,&ipf3,&ipf4);
        ...
if(ip1>=ipi1 && ip1<=ipf1)
        if(ip2>=ipi2 && ip2<=ipf2)
        ...
```

Ao continuar o desenvolvimento do trabalho foi necessário fazer verificações semelhantes em outras secções, altura em que foram descobertas as funções *MySQL inet_aton()* e *inet_ntoa().* Estas funções proporcionam uma melhoria bastante considerável ao método anterior sendo um método mais elegante, necessitando menos linhas de código e proporcionando um melhor desempenho convertendo uma *string* num inteiro e vice-versa, desta forma ao precisar verificar se um IP está entre uma gama podemos verificar os inteiros originados pela função *inet_aton().*

```
        …
 if(mysql_query(conn,"delete from arp where exists (select * from
exception ex where arp.temp=1 and ((arp.ip=ex.ipini and
arp.mac=ex.mac) or (inet_aton(ex.ipini)<=inet_aton(arp.ip) and
inet_aton(ex.ipend)>= inet_aton(arp.ip) and ex.mac is null)))")!=0)
        …
```

A configuração de excepções é feita na página de gestão.

4.3.10.3. Renovação das situações anormais

A tabela abnormal tem a informação das situações anormais, sendo o tempo em que se deixou de verificar a situação anormal uma delas. Este tempo está preenchido desde que a situação é inserida sendo o seu valor igual ao valor de início. Nas interacções seguintes é verificado se nas entradas ARP continua a existir o par IP MAC da situação anormal, se continuar o valor do campo stop é actualizado, caso já não exista a situação anormal é fechada sendo colocado o campo temp com o valor ‘0’.

```
        …
if(mysql_query(conn,"update abnormal ab set ab.temp=0 where
ab.temp=1 and not exists (select * from arp where arp.ip=ab.ip and
arp.mac=ab.mac and arp.temp=1)")!=0)
        …
        if(mysql_query(conn,"update abnormal ab set ab.temp=0
where ab.temp=1 and not exists (select * from arp where arp.ip=ab.ip
and arp.mac=ab.mac and arp.temp=1)")!=0)
        …
```

4.3.10.4. Verificação das novas entradas via OMAPI

A função responsável por esta verificação é a *void ArpTruth()*, esta função é bastante semelhante à *LeaseCheckTt* explicada no ponto 4.3.8, tendo poucas diferenças. A primeira diferença é, obviamente, a tabela onde são lidos os pares IP MAC que vão ser testados, esta verifica as novas entradas ARP, ou seja, as entradas ARP que foram lidas na interacção corrente e que não estão na interacção anterior, para isto é feita a seguinte *query*:

```
...
if(mysql_query(conn,"select a1.ip,a1.mac,a1.iprt from arp a1
where a1.temp=1 and not exists (select 1 from arp a2 where
a2.temp=0 and a2.ip=a1.ip and a2.mac=a1.mac)")!=0)
...
```

A segunda diferença está na variável que define qual o comportamento a seguir em caso de erro na verificação do servidor DHCP sendo agora utilizada outra variável, a on_dhcp_error, desta maneira é possível configurar comportamentos distintos para as duas funções.

Por fim, a terceira diferença reside no facto de quando é detectada que a *lease* não existe é inserida uma entrada na tabela abnormal (em vez da history).

```
...
sprintf(str,"insert into abnormal
values(null,\"1\",\"%s\",\"%s\",now(),now(),\"%s\");",row[0],row[1],row[2]);
if(DoQueryI(conn2,str))
...
```

4.3.10.5. Rotação das entradas ARP

Esta rotação é o último ponto da interacção e consiste em eliminar as entradas cujo campo temp seja igual a ‘0’, significando que são entradas da interacção anterior e actualizar as restantes (têm temp igual a ‘1’, da interacção corrente) para temp igual a ‘0’.

```
        ...
if(mysql_query(conn,"delete from arp where temp=0")!=0)
        ...
if(mysql_query(conn,"update arp set temp=0")!=0)
        ...
```

4.3.11. Notificação

Quando uma situação de IP forçado é detectada na rede é feita uma notificação sobre esta, ou seja, é invocada a função *void notify(char *strip, char *strmac, char *strrt)*. Esta função envia um mail para os endereços configurados, em que estes podem ser de dois tipos:

- Endereço geral
- Endereço especifico da range

O endereço geral é utilizado como contacto em qualquer situação detectada e o endereço específico de cada range, apenas usado quando o IP da situação anómala está compreendido numa determinada range, sendo usado como contacto somente nesse caso (estes contactos configurados pela página de gestão).

No ponto 4.3.1 é indicada a existência da variável sendmp que contém o caminho para o binário do *sendmail* no sistema operativo. Antes de ser feito o processamento necessário para o envio do *mail*, como verificar quais os contactos que serão utilizados é verificada a existência do binário e só posteriormente, em caso afirmativo, é que se prossegue com o envio da notificação.

Como demonstrado no ponto 4.3.10.2, também nesta função foi utilizada a funções *inet_aton()* para verificar se um determinado IP está numa determinada range.

```
        ...
sprintf(str,"select mail from notification where ipini is NULL or
(ipini<=inet_aton(\"%s\") and ipend>=inet_aton(\"%s\"))",strip,strip);
if(mysql_query(conn,str)!=0)
        ...
```

Desta forma são seleccionados todos os endereços electrónicos que devem ser utilizados nesta notificação, restando a construção do *mail* propriamente dito. Para isso recorreu-se ao utilitário *sendmail* distribuído gratuitamente e largamente utilizado nas plataformas Linux e ao comando *echo*. Através da função *system(cmd)* podemos facilmente invocar comandos do sistema operativo, tendo sido utilizada para construir o mail (comando *echo*) num ficheiro e fazer o envio deste (comando *sendmail*). O comando *echo* envia por omissão um texto para o *standard output*, ou seja, o monitor, mas é possível fazer um redireccionamento para ficheiro através do carácter “>” (escreve num ficheiro criando-o se não existir ou apagando o seu conteúdo se existir) ou “>>” (escreve num ficheiro criando-o se não existir ou caso existir adiciona no final do conteúdo existente) sendo possível construir desta forma o *mail*. Para fazer o seu envio basta invocar o comando *sendmail* com o contacto de destino e um parâmetro, o ficheiro de *mail* a enviar. O remetente utilizado é aquele que estiver configurado na variável Mails.

```
        ...
sprintf(str,"echo \"From: %s \"> ./tmpmail",mails);
system(str);
system("echo \"Subject: Forced IP detected\">>./tmpmail");
 (str,"echo \"It was detected abnormal situation with IP-%s, MAC-%s
on network device-%s\" >> tmpmail",strip,strmac,strrt);
system(str);
SemReq();
WriteLogSs(str,3);
res=mysql_use_result(conn);
        while((row=mysql_fetch_row(res)) !=NULL){
                sprintf(str,"%s %s < ./tmpmail",sendmp,row[0]);
                system(str);
                sprintf(str,"Notification sent to  %s",row[0]);
                WriteLogSs(str,2);
        }
        ...
```

4.3.12. Página de gestão

Para cumprir alguns dos objectivos propostos foi criada uma página de gestão, esta é uma página tradicional com um cabeçalho e rodapé de menu à esquerda e conteúdo à direita (respectivamente A,C,B e D da Figura 30), tendo o seu desenvolvimento sido feito no programa Dreamweaver versão 11.

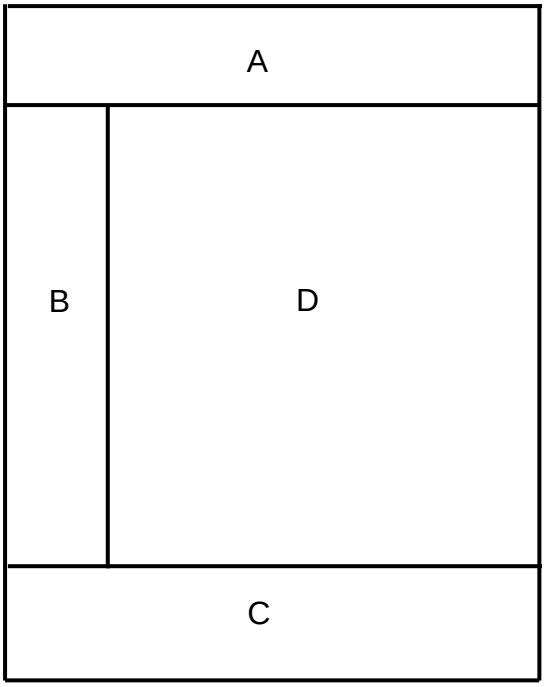


Figura 30 – Estrutura da página

Para conseguir esta estrutura foi criado um ficheiro template.php que contém (entre outras) duas funções, a *pageheader($title,$pagename)* responsável pelo desenho da parte A e B e a *pagefooter()* pela parte C, sendo a parte D da responsabilidade de todos os outros ficheiros php que contêm o seu próprio conteúdo.

```
function pageheader($title, $pagename)
{
        $html = "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0
Transitional//EN' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd'>\n";
                ...
        $html .= "<td  style='vertical-align: top; padding: 30px'>";
        return $html;
}
```

Para que todas as páginas tenham a mesma aparência o ficheiro template.php é incluído em todos os outros ficheiros php sendo nestes invocadas as funções *pageheader* e *pagefooter*.

```
<?php
        Include('template.php');
        echo pageheader('IPGEST', 'Sistema de Gestao e controlo de
endereçamento IP');
        ...
?>
        //código próprio da página
<?php
        echo pagefooter();
?>
```

Desta forma, evitamos a repetição de código em todas as páginas, para este mesmo objectivo foi criado também o db.php contendo o código necessário para a ligação à base de dados, tendo ainda a vantagem da facilidade de configuração sendo apenas necessário editar este ficheiro. Este é incluído no template.php e desta forma usado por todos as páginas (visto todas elas usarem o ficheiro template.php).

```
session_start();
//criar uma ligacao à BD
$con = mysql_connect("127.0.0.1:3306","root","*****");
if(!$con)
{
        die('Could not connect: ' . mysql_error());
}
//Seleccionar a BD
mysql_select_db("ipgest", $con);
//caminho para o binario
$binpath='/mestrado/';
```

Relativamente ao aspecto estético da página recorreu-se ao uso da linguagem de estilo CSS, sendo o objectivo que todo o conjunto de páginas tenham a mesma aparência foi criado um ficheiro (*External Style Sheet*) onde se encontra guardada as definições estéticas que algumas *tags* HTML iram assumir (irá ser apresentado apenas os elementos mais relevantes da página).

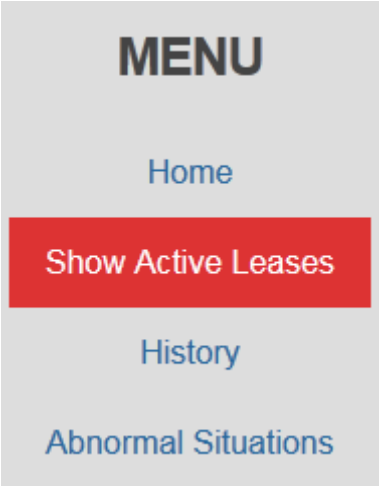


Figura 31 – Menu da página de gestão

Para se conseguir a aparência avermelhada quando o rato está sobre uma opção do menu, Figura 31, foi utilizado o seguinte CSS:

```
ul#menu a {
        ...
        display: block;
        ...
}
ul#menu a:hover {
      background-color: #d33;
      color: #fff;
      ...
}
```

Do primeiro bloco a opção *display: block* provoca que sejam accionadas as acções ao passar sobre toda a célula e não apenas sobre a hiperligação, no segundo bloco é definida a cor de fundo e texto quando o rato está sobre a célula.

IP	MAC	START TIME	STOP TIME
194.210.194.17	40:a6:d9:ea:05:a3	2011-09-20 15:27:00	2011-09-20 15:57:11
10.5.35.210	7c:2f:80:17:10:a4	2011-09-20 15:27:01	2011-09-20 15:57:17
10.5.35.190	00:04:f2:1e:af:73	2011-09-20 15:27:02	2011-09-20 15:57:23
193.137.128.72	7c:c5:37:ac:c1:c2	2011-09-20 15:27:04	2011-09-20 15:57:29

Figura 32 – Tabela com informação

Na Figura 32 está representada uma tabela com informação (neste caso histórico de atribuições de endereços IP) tendo como respectivo CSS o seguinte:

```
        ...
th {
        background-color: #555;
        color: #fff;
        font-size: 16px;
        font-weight: bold;
        padding: 10px;
        text-transform: uppercase;
}
td {
        margin: 0;
        padding: 0;
        border: 1px solid #aaa;
        padding: 5px;
        text-align: center;
}
        ...
```

Como é possível verificar estão definidas algumas propriedades como a cor, tamanho da fonte, cor de fundo entre outras. Quando é necessário que uma *tag* HTML definida no ficheiro CSS tenha outras definições basta defini-las na própria página ou *tag* pois têm prioridade sobre aquelas definidas no ficheiro.
A página inicial é mostrada pela Figura 33.

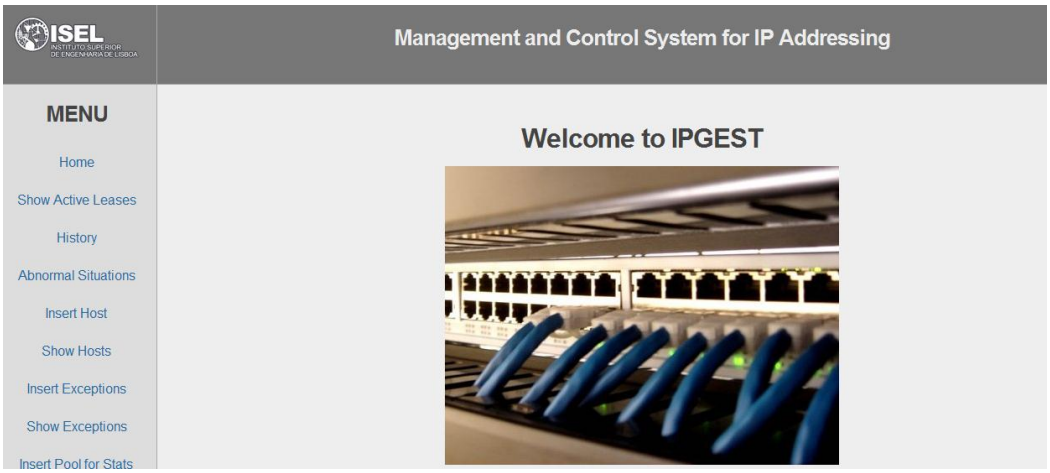


Figura 33 – *Layout* da página

No capítulo da segurança um utilizador para conseguir aceder a qualquer link do menu tem de ter as credencias correctas (nome de utilizador e palavra-chave) estando estas inseridas na base de dados. Qualquer acesso indevido (não validado correctamente) efectuado será redireccionado para a página de *login*. Este aspecto foi conseguido pela recorrendo a uma variável de sessão, sendo esta criada apenas se o utilizador inserir as devidas credenciais. Dada a existência da variável mencionada basta testar no início de cada página se esta existe, se existir é dado acesso, caso contrário é executado um redireccionamento para a página de *login*.

```
Ficheiro login.php:
if($row[0])
{
        $_SESSION['username'] = $user;
        header ('location: index.php');
}

Ficheiro template.php:
function checklogin()
{
        if(!isset($_SESSION['username'])){ Header("Location:
login.php"); }
}

Em qualquer outra página:
include('template.php');
checklogin();
```

Dada a semelhança notória entre algumas das páginas existentes estas não serão todas apresentas, mas sim apenas um subconjunto representando cada tipo de página existente.

A página *History* representa, por serem semelhantes no seu código, as páginas *Show Active Leases, Abnormal Situations e Show Stats* (dando respectivamente informação sobre o histórico de atribuições IP, as *leases* activas no momento e situações de IP forçado). Após serem efectuados os *includes* indicados é invocada a função *echo pageheader('IPGEST', 'Management and Control System for IP Addressing');* que irá desenhar parte da página, como indicado anteriormente neste ponto. Visto estas páginas serem responsáveis por mostrar informação ao utilizador deve ser possível que este tenha acesso a campos de pesquisa para colmatar a eventual necessidade de filtrar informação, assim existe um formulário que é possível preencher sendo este implementado através da *tag input* do tipo *text* e *submit* estando dentro de uma tabela (*tag table*) por fins estéticos:

```
<h2>Search</h2>
<form name='pesqle' method='post'>
<table>
<tr><th>IP</th><th>MAC</th><th>Start time</th><th></th></tr>
<tr>
<td><input type='text' name='IPle' id='IPle'  /></td>
<td><input type='text' name='MACle' id='MACle' /></td>
<td><input type='text' name='Startle' id='Startle' value="YYYY-MM-
DD" onfocus="if(this.value=='YYYY-MM-DD') this.value=""
onblur="if(this.value==") this.value='YYYY-MM-DD'" /> : <input
type='text' name='Startlem' id='Startlem' value="HH:MM"
onfocus="if(this.value=='HH:MM') this.value=""
onblur="if(this.value==") this.value='HH:MM'" /></td>
<td><input type='submit' value='Search' name='doPesqle' /></td>
</tr>
</table>
</form>
```

Para facilitar a introdução de dados os campos cujos dados são datas têm visível o formato que deve ser usado (este desaparece assim que é seleccionado para se introduzir dados). O código acima (juntamente com o CSS apresentado o formulário visível na Figura 34.

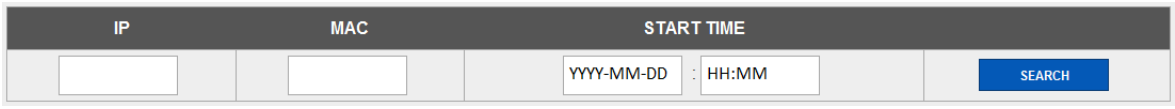


Figura 34 – Formulário

Ao abrir uma página é necessário testar se esta abertura foi originada pelo botão *Search*, por o utilizador que está a navegar nas páginas dos resultados ou se é o primeiro acesso. Caso seja o primeiro acesso apenas é mostrado formulário, em caso de alguma das restantes opções são lidas e verificadas as variáveis existentes no formulário que serão utilizadas na pesquisa a fazer (é testado se a origem é o botão ou navegação dos resultados pois o método pelo qual são transferidas as variáveis é diferente *POST* e *GET*).

```
if(isset($_POST['doPesqle']) or isset($_GET["page"]))
{
        // Ler as variveis do form
        $IP = htmlspecialchars($_POST["IPle"]);
        …
        if(!isset($_POST['doPesqle'])){
                $IP = htmlspecialchars($_GET["IPle"]);
        …
        if ($IP!=""){
                $querytmp .=" where ip like '%$IP%'";
                $const=1;
                $pageurl.="&IPle=".$IP;
        }
        if ($MAC!=""){
                if($const==1) $querytmp .=" and mac like '%$MAC%'";
                else{
                        $querytmp .=" where mac like '%$MAC%'";
                        $const=1;
                }
                $pageurl.="&MACle=".$MAC;
        …
```

A variável *$pageurl* é importante para a navegação do resultado obtido pois contém parte do URL que será utilizado, esta parte para se saber posteriormente qual os campos de pesquisa a utilizar. Para executar a pesquisa na base de dados é usado a função *mysql_query($query, $con)* sendo testado se esta foi executada com sucesso e se retornou algum resultado.

Se esta tiver sido executada sem sucesso é mostrado o respectivo erro, caso contrário, ou é dada uma mensagem sobre a inexistência de dados ou, caso estes existam, estes serão mostrados ao utilizador. Em seguida é executada uma consulta para contabilização dos registos devolvidos sendo este dividido por vinte (número de registos por página) dando origem ao número total de páginas que serão disponibilizadas (assim o utilizador poderá ver a informação em intervalos de 20 registos) sendo estas disponibilizadas abaixo dos resultados (Figura 35).

```
        …
if(!mysql_num_rows($result)) $ReturnText .= "<h2>No abnormal
situations</h2>";
        else
        …
while($row = mysql_fetch_array($result))
                {
                        $ReturnText .= "<tr>";
                        $ReturnText .= "<td>" . $row['ip'] . "</td>";
                        …
                        $ReturnText .= "<td>" . $row['iprt'] . "</td>";
                        $ReturnText .= "</tr>";
                }
        …
for ($i=1; $i<=$total_pages; $i++) {
        $ReturnText .= "<a
href='abnormal.php?page=".$i.$pageurl."'>".$i."</a> ";
};
        …
```

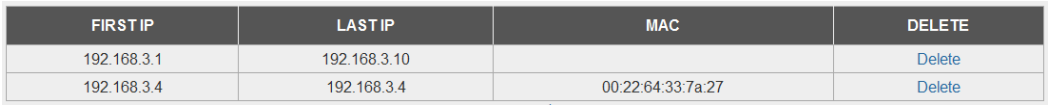
pages: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

Figura 35 – Hiperligações para navegação

Entre estas páginas apenas de diferenciar a pesquisa realizada em *Show Stats* pela necessidade de cruzar informação entre as *pools* e as *leases* activas.

```
select sp.name as name,((sp.ipend-sp.ipini)+1) as total, count(al.ip) as
used FROM statspool sp left join active_lease al on sp.ipini <=
inet_aton(al.ip) and sp.ipend>=inet_aton(al.ip)
```

O próximo conjunto a ser apresentando é constituído pela *Show Exceptions, Show Pools of Stats, Show Contacts* que contêm respectivamente informações sobre as excepções existentes, as *pools* configuradas e os contactos para a notificação. Estas páginas são em parte semelhantes ao grupo anterior na medida que têm, também, um formulário e o seu processamento é feito de forma semelhante, no entanto, existe a diferença da informação contida neste grupo poder ser eliminada, existindo na tabela de resultados uma hiperligação que elimina a respectiva linha (Figura 36).



FIRST IP	LAST IP	MAC	DELETE
192.168.3.1	192.168.3.10		Delete
192.168.3.4	192.168.3.4	00:22:64:33:7a:27	Delete

Figura 36 – Tabela com *delete*

Para fazer o delete é criada, de forma semelhante à navegação nos resultados, uma hiperligação para uma página que efectua o delete da respectiva linha recebendo, por isso, os dados contidos na linha (poder eliminar na base de dados) e os dados utlizados na pesquisa (ao voltar para esta página após a eliminação do registo o utilizador irá ver a informação respectiva aos campos usados na pesquisa anterior).

```
while($row = mysql_fetch_array($result))
{
        $ReturnText .= "<tr>";
        $ReturnText .= "<td>" . $row['name'] . "</td>";
        $ReturnText .= "<td>" . $row['ipini'] . "</td>";
        $ReturnText .= "<td>" . $row['ipend'] . "</td>";
        $ReturnText .= "<td> <a
href='delete_row_pool.php?ipini=".$row['ipini']."&ipend=".$row['ipend'].
$deleteurl."&page=".$page."'>Delete</a></td>";
        ...
```

A página que elimina o registo propriamente dito executa um *query* simples tendo a que faz a gestão dos *hosts* a diferença de não só eliminar na base de dados como também do servidor DHCP chamando um binário através da função *exec()*. Neste caso só faz sentido a eliminação na base de dados caso a do servidor DHCP tenha tido sucesso sendo mostrado ao utilizador o sucesso ou não da operação e uma hiperligação para voltar à página anterior.

```
        ...
$resultexec=exec($binpath."delete_host $NAMEh $IPh $MACh $IPd
$PORTd",$out);
        ...
if($resultexec!=""){
        if($resultexec==0){
                $query = "delete from hosts where
name=\"".$NAMEh."\" and ip=\"".$IPh."\" and mac=\"".$MACh."\" and
ipdhcp=\"".$IPd."\" and port=\"".$PORTd."\";";
                        $result = mysql_query($query, $con) or
die(mysql_error());
                $error = "Deleted with success.";
                        }
        else{
                $error = "Deleted without success.";
        }
}
        ...
echo "<a
href='show_hosts.php?".$deleteurl."&page=".$page."'>Back</a>";
        ...
```

Por fim, o último grupo constituído pela *Insert Host*, *Insert Exceptions*, *Insert Pool for Stats* e *Insert Contacts* fazem respectivamente a inserção de um *host* (na base de dados e servidor DHCP), de excepções (usado no processamento da tabela, de uma *pool* (usada para construir estatísticas) e de contactos (usados para notificar situações anormais). O código existente nestas páginas é semelhante ao já apresentado nos grupos anteriores.

Para terminar este ponto é importante saber que os binários executados (ao inserir e eliminar um *host*) foram desenvolvidos em C de forma semelhante ao componente apresentado anteriormente.

5. Testes realizados

Ocorreram duas fases distintas de testes. Uma primeira ao longo de todo o desenvolvimento, pois é impossível pensar proceder de outra forma, não é viável desenvolver uma aplicação sem que sejam efectuados testes funcionais após cada módulo ou mesmo função (dependendo obviamente da experiência de cada programador), pois irá, certamente, ser penalizador, o tempo que teremos que investir para investigar a razão de determinado comportamento não esperado. Foram realizados testes após a conclusão de cada função invocando-as com os devidos parâmetros e verificado que o resultado final era o adequado, assegurando, assim, o correcto funcionamento da mesma e que esta poderia ser incluída como parte de uma função maior. Este facto permitiu alguma agilidade na mudança de requisitos como, por exemplo, a passagem do suporte de um servidor DHCP para vários. Algo que foi também essencial, foi o facto de terem sido efectuado testes posteriormente a qualquer alteração de código, mesmo pensando que esta não iria ter um impacto funcional (*Code refactoring tests*). Dentro das capacidades do ambiente de desenvolvimento foram realizados testes com o objectivo de verificar todos os fluxos possíveis. Antes de explicar os testes feitos é importante perceber a rede e componentes criados virtualmente, tendo sido criada a máquina de desenvolvimento com duas interfaces de rede, uma que a liga à maquina física e outra que liga à rede interna das máquinas virtuais. A esta rede interna está também ligada uma segunda máquina virtual (com duas interfaces) cujo propósito é indicado posteriormente neste ponto. O componente Dynamips (simulador de um *router*) está ligado ao interface virtual que liga a máquina de desenvolvimento e máquina física (Figura 37).

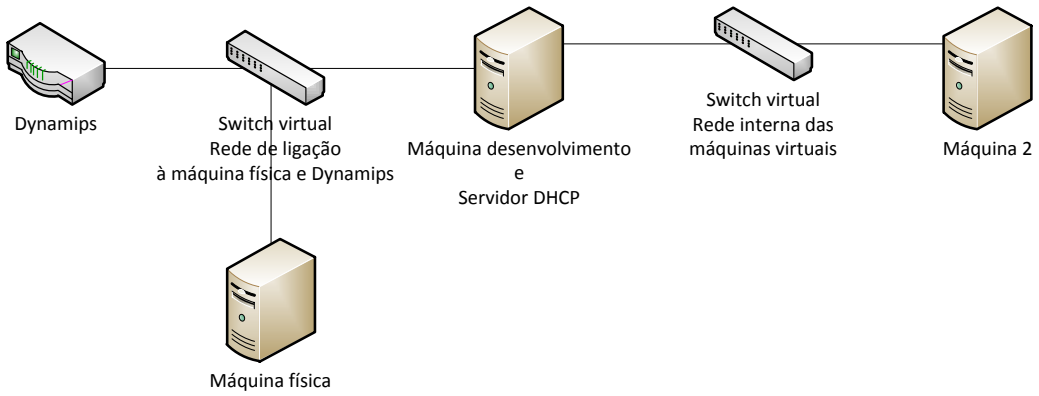


Figura 37 – Diagrama da rede virtual implementada

Para testar o fluxo do envio de pedidos DNS *update* do servidor DHCP, a segunda máquina virtual tem as suas interfaces configuradas para fazer um pedido de configuração ao servidor DHCP, desta forma o DHCP criar as respectivas *leases* e envia os pedidos DNS *update* para o componente desenvolvido. Quando configurado na opção de detalhe máximo o *log* da aplicação desenvolvida permite ver todos os campos do pedido DNS *update* sendo possível confirmar a recepção e correcta descodificação do pedido, assim como o correcto funcionamento da escrita em *log* (apesar de como dito anteriormente todas as funções terem sido testadas quando desenvolvidas). Foi verificado, ainda, que o pedido estaria na tabela active_lease da base de dados.

Estando a tabela active_lease a ser preenchida correctamente o teste seguinte consistiu em verificar que era detectado a extinção da *lease* e assim ser colocada a respectiva entrada na tabela history. Para realizar este teste utilizou-se novamente a segunda máquina virtual para originar novas *leases* tendo, em seguida, sido desligada para simular o abandono de um computador da rede. Algum tempo depois (este tempo está dependente das configurações no servidor DHCP) as *leases* expiraram tendo sido detectada correctamente. Para testar o funcionamento correcto da aplicação em caso de erro na verificação ao servidor DHCP, este foi desligado tendo a aplicação assumido a *lease* expirada ou activa consoante a variável de controlo respectiva. Este teste foi repetido posteriormente quando foi implementado o suporte da consulta a vários servidores DHCP, apesar de só ter um servidor foi possível verificar o correcto comportamento configurando na aplicação um segundo DHCP num IP inexistente e posteriormente com o mesmo IP do servidor DHCP existente, sendo testada a verificação a uma *lease* activa e inactiva e verificando no log a consulta efectuada a ambos os servidores.

O último conjunto de fluxos testados foi o processamento das tabelas ARP. Devido à dimensão foram testados pequenos fluxos correspondentes às funções invocadas para atingir um fluxo macro. Para testar o sistema de rotação da tabela ARP (função *arp_rotate*) foram realizadas inserções, seguindo-se a invocação da função e verificada a tabela tendo os registos desta sofrido alterações em conformidade (registo com o campo temp igual a ‘0’ foram eliminados e os com temp=’1’ actualizados para temp=’0’). Para simular a execução do processamento de excepções foram inseridos alguns registos na tabela exception de dois tipos possíveis (range IP e IP com associação MAC), tendo em conta as excepções, foram inseridos registos que deveriam ser

retirados (excepções) e registos que não deveriam (não seriam excepções). Foi confirmado que apenas foram eliminados os registos que eram excepções.

A verificação da autenticidade das entradas ARP é feita inicialmente pelo componente desenvolvido em shell script tendo sido este executado e verificado que o ficheiro produzido continha as entradas devidas. Na componente em c foi verificada a correcta leitura do ficheiro e posterior inserção na base de dados. Confirmou-se que foram seleccionados para verificação apenas as novas entradas ARP (entradas que não estavam presentes na interacção anterior) e que estas foram verificadas no servidor DHCP tendo o log produzido auxiliado nesta verificação. Para terminar este fluxo foi confirmada a correcta inserção do registo na tabela abnormal de situações em que a entrada ARP não tinha nenhuma correspondência no servidor DHCP (as que tinham correspondência não eram registadas).

Para testar o prolongamento da duração de uma situação anormal (função *Renew_abnormal*) foi realizada a inserção de um registo na tabela ARP igual a um registo de uma situação anormal, verificando-se que o tempo do campo stop tinha sido actualizado (estendido para o tempo actual). Caso não exista a referida entrada ARP é actualizado o campo temp da situação anormal com o valor ‘0’ (significando o fecho da ocorrência). Por fim, ao ser detectada uma situação anormal é invocada uma função de notificação tendo sido confirmada a recepção do *mail*.

Para os testes na página de gestão foram aproveitados sempre que possível os momentos em que as tabelas tinham dados gerados de forma real e noutros casos inseridos manualmente e também de forma controlada, tendo sido testados os fluxos possíveis.

O segundo momento foi o conjunto de testes realizados na infra-estrutura do IPL/ISEL, sendo estes os testes mais indicados para descobrir falhas que não são revelados em testes controlados em máquinas virtuais, pois apresentam um volume de carga real, situações reais algo que não é possível reproduzir no conjunto anterior. Os testes revelaram algumas fragilidades da aplicação, nomeadamente no processamento de um pedido DNS *update* (não estava a fazer o *parsing* correcto em algumas situações especificas), na gestão dos *file descriptors* devido a uma omissão na documentação do dhcpctl, estes estavam a ser todos consumidos (1024), tendo sido esta situação investigada e

descoberta com recurso ao utilitário *strace* e ao *gdb*. Após a correcção dos referidos erros a aplicação apresenta o comportamento esperado.

6. Dificuldades encontradas

Ao longo do desenvolvimento deste projecto foram encontradas dificuldades que são normais em qualquer projecto (por isso apenas são apresentadas as mais relevantes) e que são inerentes à curva de aprendizagem da tecnologia em causa, este projecto teve a agravante de implicar vários componentes, várias tecnologias envolvidas sendo necessário um esforço suplementar. Começando com a linguagem de desenvolvimento escolhida, o C, do ponto de vista da facilidade de implementação não ter sido a escolha mais correcta, não por ser uma linguagem complicada pois é relativamente simples, mas, pelo facto de ao longo de desenvolvimento, nas pesquisas realizadas se terem verificado situações em que linguagens de mais alto nível como o Java ou PERL terem disponíveis bibliotecas de suporte à interacção com grande parte dos protocolos usados. Estes protocolos e funcionalidades para serem realizados em C necessitam de mais linhas de código, de mais esforço, algumas vezes provocado, também, pela lacuna de documentação.

O *dhcpctl*, que implementa conjunto de funções OMAPI, padece deste facto sendo a sua documentação (uma *man page* Linux) insuficiente pois apenas descreve as funções (algumas de forma deficiente ou até omissa) e tendo apenas um exemplo de código para obter o tempo de fim de uma *lease*, sendo este o exemplo quase exclusivo na informação encontrada. Este facto provocou situações de incompreensão nalguns comportamentos em funções que interagiam com o servidor DHCP pois não funcionavam correctamente, como a inserção ou remoção de um *host*, algo que infelizmente foi resolvido um pouco por tentativa e erro. Assim, facilmente nos interrogamos sobre se estaremos a proceder erradamente, uma vez que verificamos comportamentos desconhecidos e aparentemente inexplicáveis. Após alguns testes (utilização de omshell, implementação OMAPI para linha de comandos) e alguma pesquisa incluindo o uso de listas de distribuição existentes sobre o DHCP (ISC) que reúnem uma comunidade de utilizadores e programadores verifica-se efectivamente a existência de *bugs* no comportamento da API. Este *bug* encontra-se num cenário concreto de verificação do estado de uma *lease*, sendo este quando o MAC é desconhecido e o IP conhecido no servidor, devolvendo este um erro dependendo da versão, na versão 3.0.5 da aplicação, o erro *key conflict*, na 4.1 *unkown error: 393223*. A versão 4.2.1 tinha *bug* derivado de uma nova dependência do código do BIND (conhecimento obtido numa das lista) em que o código não era sequer compilado. Para tentar

ultrapassar esta situação verificou-se o código fonte do servidor DHCP, algo que se mostrou infrutífero, devido à sua complexidade. A resposta obtida também nas listas seria que algumas questões relacionadas estariam corrigidas na última versão (4.2.1, ou seja, a que mais tarde se verificou não compilar devido à nova dependência). Com o tempo a escassear decidiu-se avançar resolvendo esta situação com as variáveis de controlo de comportamento (on_dhcp_error e on_dhcp_error_l) pois num futuro próximo este erro seria certamente corrigido.

Por fim, a última dificuldade está relacionada com a máquina virtual onde foi efectuado todo o desenvolvimento. Quando o projecto estava já numa fase avançada de desenvolvimento o *kernel* da máquina virtual ficou corrompido ao instalar-se extensões da Oracle Virtual Box, isto provocou que a máquina virtual não arrancasse, não sendo possível copiar a informação nela contida. Para recuperar toda a informação foi necessário arrancar a máquina virtual com um *live-cd* para permitir montar as partições do disco e assim possibilitar a cópia do projecto para a máquina física. Como não foi possível recuperar a máquina virtual foi instalada outra, copiando de seguida para esta o projecto.

7. Melhoramentos

Quando realizamos um projecto, o produto final deve ser alvo de uma crítica, devemos ter consciência das falhas cometidas do que pode ser melhorado que novas características devem ser introduzidas.

A base de dados é seguramente um ponto a rever apesar de ter um modelo de dados perfeitamente funcional, o mesmo poderá não ser o mais optimizado sendo necessária uma análise mais aprofundada sobre possíveis melhoramentos ou regras de boa conduta a seguir.

O código deverá ser revisto tendo o objectivo de possíveis optimizações, melhores formas de implementar determinadas funcionalidades, rever o tamanho das variáveis usadas pois é bem possível que algumas estejam sobredimensionadas (dado o nível actual de desempenho das máquinas, quantidade de memória hoje em dia disponível, não foi dada particular atenção a este ponto pois não terá um impacto significativo). Este ponto foi efectuado como indicado no ponto 4.3.10.2 sobre a função de processamento das excepções. Além da optimização a revisão deverá ter em atenção a verificação de validações adicionais nos fluxos, de forma a evitar que a aplicação termine devido a um erro, por exemplo, no tratamento de um pedido de DNS *update* criar validações extra que protejam o acesso a zonas de memória indevidas (originando o sinal *segmentation fault*). Tais validações são extremamente importantes e o facto pelo qual não foram implementadas na sua totalidade reside apenas nas restrições temporais, pois o foco foi a funcionalidade.

O suporte de IPv6 é uma funcionalidade importante de implementar, algo que não foi realizado devido à escassez de tempo. Visto os mesmos fluxos se aplicarem também ao IPv6 seria apenas necessário verificar as funções existentes e generalizar as mesmas para que as duas versões do protocolo fossem suportadas.

Seria importante criar uma funcionalidade que permitisse uma forma de configurar um determinado endereço MAC com o objectivo de ser enviada uma notificação caso este fosse detectado na rede.

À semelhança das protecções no código para componente desenvolvida em C é importante que sejam criadas na página de gestão validações na introdução de dados pelos formulários. Estas deverão verificar se o valor introduzido é o adequado, por exemplo verificar o formato, tamanho, se são letras ou números.

Neste momento o número de registos por página é fixo sendo este de 20 registos, no entanto, uma melhoria seria este valor ser configurável.

Uma notificação é uma comunicação de uma situação anómala encontrada e é enviada assim que é verificado que determinado IP/MAC não é válido, no entanto, apenas devia ser enviada no fim de cada ciclo de verificação com todas as situações anómalas encontradas.

Por fim, a aplicação devia contemplar a limpeza do histórico da atribuição de endereços IP, devendo esta ser configurável podendo ser escolhido o tempo desejado de histórico.

8. Problemática IPv6

A IANA no dia 3 de Fevereiro de 2011 alocou os últimos blocos de endereços IPv4 disponíveis sendo de seguida feita uma a distribuição automática dos restantes cinco blocos de endereçamento (/8) que se encontravam cativos pelos *regional registries*, de forma equitativa.

Actualmente a Internet (IPv4) só poderá crescer baseando-se nas reservas existentes nos *regional registies,* que são relativamente escassas. Esta situação evidencia que o IPv6 será certamente o futuro, a única questão existente será durante quantos anos o IPv4 nos irá ainda acompanhar no dia-a-dia. Neste cenário seria inconsciente o IPv6 ser por completo esquecido no contexto deste projecto.

Os endereços IPv6 são a 128bits contra os 32 do seu antecessor sendo desta forma possível ter uma conectividade global, isto é, não existe a necessidade (do ponto de vista da limitação de endereçamento) de efectuar conversões de endereços (NAT), mas o IPv6 é muito mais do que um simples aumento do universo de endereçamento, tendo entre outras características a capacidade de autoconfiguração de uma interface de rede, isto é, esta consegue configurar-se automaticamente recorrendo exclusivamente ao IPv6.

Ao ser iniciada uma interface, esta para ter conectividade pode, configurar automaticamente os seus endereços recorrendo a DHCPv6 de forma idêntica à do IPv4, gerar um sufixo de 64bit baseado no endereço MAC da interface (EUI-64) e concatenando-o a um prefixo anunciado pelos *routers* ou até usando como 64bit de menor peso um número aleatório prefixado pelo endereço de rede anunciado pelos *routers* para o segmento (endereços temporários).

Se a obtenção de endereçamento for baseada em DHCP, toda a abordagem seguida neste projecto pode ser facilmente adaptada para de forma idêntica lidar com o IPv6, no entanto, para tal será necessário o suporte pleno das funcionalidades usadas quer pelo servidor de DHCP quer pelos *routers* na disponibilização via SNMP da informação de associação entre endereços físicos (MAC) e IPv6, aquilo que no protocolo é designado de *neighbour table.*
Devido à falta de suporte da maioria destas funcionalidades nos componentes utilizados bem como ao uso pouco frequente desta forma de configuração nas redes IPv6 actuais, não foi possível qualquer investimento do suporte de IPv6

pelo projecto, na sua vertente fundamental de gestão e controlo do espaço de endereçamento.

Nas situações mais comuns do suporte de configuração de IPv6 nas redes actuais, pelo uso de autoconfiguração baseada no endereço MAC ou pior, em endereços com componentes geradas aleatoriamente e alterando-se periodicamente, é completamente inviável qualquer tentativa de controlo central de endereços legítimos ou fora do controlo da rede, na prática, neste cenário, todos os endereços se encontram fora do controlo da gestão da rede.

9. Conclusão

Este trabalho pretende fornecer uma possível abordagem à problemática central, ou seja, a questão da segurança relacionada com a possibilidade de forçar um IP, quando na rede é praticada a configuração deste via DHCP. Para atingir este objectivo foram apresentadas as análises efectuadas sobre os dados fornecendo o conhecimento necessário para o entendimento de como estes existem, a forma como são tratados e como estes foram cruzados, assim este trabalho pode ser usado também como base ou consulta para uma nova aproximação ao tema.

Nesta abordagem, pela análise efectuada as entradas ARP presentes nos equipamentos de rede consultados são apenas verificadas uma vez, sendo esta realizada no momento em que são descobertas pela primeira vez. O resultado desta verificação é importante pois será usado para o possível envio de uma notificação para os contactos configurados (pela página de gestão), caso este indique que tal entrada ARP não é válida (ou em caso de erro e assim esteja configurado).

Não focando apenas a problemática central, este projecto fornece uma página de gestão que permite aos administradores da rede visualizar o estado do consumo das *pools* existentes no servidor DHCP (configuradas também na página de gestão) assim como a inserção e remoção via OMAPI de atribuições estáticas (não sendo deste modo necessário a reinicialização do servidor para assumir a nova atribuição).

Na sua generalidade o sistema desenvolvido desempenha as funções pretendidas, foram confirmadas diversas situações de anomalias identificadas e que se tratavam de equipamentos com necessidade de excepções que após colocadas na aplicação deixaram de despoletar os alarmes.

Pela natureza das verificações realizadas é possível detectar outros tipos de ataques como o IP *spoofing*/ARP *spoofing*/ARP *poisoning* cujo objectivo é colocar informação falsa nas tabelas ARP (é detectado pois esta informação é verificada no servidor DHCP).

Foram identificadas diversas ocorrências de falsos positivos e falsos negativos, situações que geraram alertas erróneos ou que mereciam alerta,

mas não o tiveram. A maioria destas situações foi originada por respostas aparentemente inconsistentes, não documentadas e até diferentes entre versões do servidor DHCP.

As anomalias de comportamento foram resolvidas com a análise do cenário em que ocorriam e alguma dedução do porquê dos estados reportados pelo DHCP, tendo em conta o conhecimento do protocolo e da escassa documentação da implementação do mesmo.

Ao longo do projecto foram usadas várias versões (algumas que saíram durante a realização do projecto), mesmo com a última versão (4.2.2) utilizada, na tentativa de eliminar os comportamentos aparentemente errados, tal não foi possível, como os erros descritos no ponto 2 ("*key conflict*" e "*more than one object matches key*"). Após a realização deste projecto e inerente experiência obtida, é possível afirmar-se, tendo em atenção todos os erros obtidos, alguns deles sem explicação plausível, pelas lacunas graves de documentação que a interface OMAPI não está ainda numa fase madura revelando dificuldades de utilização.

10. Bibliografia

RFC 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION - http://www.ietf.org/rfc/rfc1035.txt

RFC 2136 - Dynamic Updates in the Domain Name System (DNS UPDATE) - http://www.ietf.org/rfc/rfc2136.txt

RFC 1541 - Dynamic Host Configuration Protocol - http://tools.ietf.org/html/rfc1541

RFC 2462 - IPv6 Stateless Address Autoconfiguration - http://tools.ietf.org/html/rfc2462

RFC 4291 - IP Version 6 Addressing Architecture - http://tools.ietf.org/html/rfc4291

RFC 3879 - Deprecating Site Local Addresses -http://www.ietf.org/rfc/rfc3879.txt

RFC 4293 - Management Information Base for the Internet Protocol (IP) – http://tools.ietf.org/html/rfc4293

RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II – http://www.ietf.org/rfc/rfc1213.txt

CSS Tutorial - W3Schools - http://www.w3schools.com/css/default.asp

Memory leak in dhcpd and dhcpctl - client http://www.archivum.info/dhcp-server@isc.org/2005-05/00117/Re-Memoryleak-in-dhcpd-and-dhcpctl-client.html

DNS Query Code in C with winsock and linux sockets http://www.binarytides.com/blog/dns-query-code-in-c-with-winsock-and-linux-sockets

DNS Message Header and Question Section Format - http://www.tcpipguide.com/free/t_DNSMessageHeaderandQuestionSectionFormat.htm

Sockets Tutorial - http://www.linuxhowtos.org/C_C++/socket.htm

HowTo: "Error: 209-unable to start instance" - http://7200emu.hacki.at/viewtopic.php?t=764

Tutorial: How to set up a basic Dynamips Lab using Dynagen, Part 1 - http://ardenpackeer.com/featured-articles/how-to-set-up-a-basic-dynamips-lab-using-dynagen-part-1/

Synchronizing Threads with POSIX Semaphores - http://www.csc.villanova.edu/~mdamian/threads/posixsem.html

Write to a file in C - http://hackerslife.blogspot.com/2005/01/write-to-file-in-c.html

The recvfrom() API call
- http://www.scottklement.com/rpg/socktut/recvfromapi.html

Howto: Connect MySQL server using C program API under Linux or UNIX - http://www.cyberciti.biz/tips/linux-unix-connect-mysql-c-api-program.html

Dynamips / Dynagen Tutorial - http://dynagen.org/tutorial.htm#_Toc193247997

Documentação de desenvolvimento SGBD MySQL - http://dev.mysql.com/

PHP / MySQL select data and split on pages - http://www.phpjabbers.com/php--mysql-select-data-and-split-on-pages-php25.html

How do I print one bit? - http://stackoverflow.com/questions/349931/how-do-i-print-one-bit

Net-snmp - www.net-snmp.org

Acetatos da unidade curricular TAR - http://www.deetc.isel.ipl.pt/redesdecomunic/disciplinas/TAR/acetatos/04%20-%20IPv6.pdf

Free Pool of IPv4 Address Space Depleted - http://www.nro.net/news/ipv4-free-pool-depleted

http://www.isc.org/community/blog/201104/isc-dhcp-and-ipv6-dhcpv6-story - DHCPv6 - vantagens do uso DHCPv6

SNMP Object Navigator - http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?objectInput=ipNetToMediaPhysAddress&translate=Translate&submitValue=SUBMIT&submitClicked=true

Configuring IPv6 Neighbor Discovery - http://www.cisco.com/en/US/docs/security/asa/asa82/configuration/guide/route_ipv6_neighbor.html#wp1078917

Create A Secure Network With Allied Telesis Managed Layer 3 Switches - http://www.alliedtelesis.com/media/datasheets/howto/secure_network_l3switches.pdf

Catalyst Integrated Security Features (CISF) - http://www.cisco.com/web/strategy/docs/gov/turniton_cisf.pdf

Cristóvão, Fernando, Método Sugestões para a elaboração de um ensaio ou tese, Edições Colibri, Lisboa 2001