Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e Informática

Vítor Rogério Gomes Pardal de Oliveira Pascoal

Definição de mecanismos para comunicação entre processadores e FPGAs


Universidade de Aveiro
2008
Departamento de Electrónica, Telecomunicações e Informática

Vítor Rogério Gomes Pardal de Oliveira Pascoal

Definição de mecanismos para comunicação entre processadores e FPGAs

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Francisco Manuel Marques Fontes, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor Victor Marques, Gestor de Divisão de Desempenho de Rede e Plataformas de Acesso em Rádio e Cobre da Portugal Telecom Inovação, S.A.

Com o Apoio da PT Inovação

Dedico esta dissertação ao meu falecido Avô Armando, pelo que significou e significará em toda a minha vida. Onde quer que estejas, estarás sempre no meu coração.

o júri

presidente

Doutor Rui Valadas
Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

arguente

Doutora Rute Sofia
Lider do Internet Architectures and Networking Group (IAN) do Instituto de Engenharia de Sistemas e Computadores (INESC) do Porto

orientador

Doutor Francisco Fontes
Professor Auxiliar convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

co-orientador

Doutor Victor Marques
Gestor de Divisão de Desempenho de Rede e Plataformas de Acesso em Rádio e Cobre do Departamento de Desenvolvimento de Sistemas de Rede da Portugal Telecom Inovação, S.A.

agradecimentos

À minha irmã Renata Rafaela e aos meus pais Vítor e Ester, que sempre me apoiaram nos momentos mais difíceis da vida dando-me sempre o incentivo necessário para perceber que a vida é bela.
Ao meu padrinho Rui e aos meus avós Amadeu, Ester e Maria que me apoiaram sempre.
Aos meus "irmãos", nomeadamente ao Vítor Paiva, ao Abílio e ao Lino que acompanharam sempre o meu percurso académico nesta Universidade.
À Patricia que me aturou muitas vezes até às 5:18 no MSN enquanto trabalhava para este documento dando-me sempre a motivação necessária.
Aos amigos do "Café Tako", nomeadamente ao Vítor Teixeira, ao Armando, ao Zé e ao Almeida, que são uns "pais aveirenses" para mim, e que me fizeram sentir confortável nesta cidade.
À Dona Ilda e à Maria João que aturaram sempre as minhas músicas a um som sempre bem elevado e que me fizeram companhia no meu "ninho aveirense".
Ao João Condeço e ao João Casimiro, que são "dois pais aveirenses" e à Ana Ferreira que são grandes amigos que tenho cá em Aveiro.
Aos "meus patrões" da Universidade de Aveiro, Engenheiro Fomes e Elsa@Novisad,Serbia, que tornaram o meu percurso académico bem mais bonito.
Aos meus amigos: Isaias Barros, Armando Lousada, Snower, Hugo Félix, Daniel Martins, Ricardo Souto, Camarada, Diogo Paiva, Chefe, Farturas, Luís Figueiredo, Claúdio, Emanuel Santana, Jorge Aido, Vasco Figueiras, Ricardo Pinto, Vera Teixeira, Eduardo, Ana Sofia, Marina Jordão, Daniela Bastos, Vânia, Ritinha, Andreia Ferreira, Ana, Xana, Tiago Carapeto, Artur "Gaivota", Fafe, Pêra, Carlão, Soldado e aos meus pedaços de terra que me deram sempre a força para concluir esta dissertação.
À minha ex-namorada Julieth Karina, pelo apoio que me deu ao longo do meu percurso académico, incluindo a escrita deste documento.
Ao Engenheiro Nuno Duarte, colaborador na orientação do meu trabalho de mestrado.
Aos meus orientadores, os Professores Francisco Fontes e Victor Marques por todo o tempo e papel desempenhado na orientação deste documento.

Por fim, deixo esta linha para todos aqueles a quem a minha memória falhou…

palavras-chave

FPGA, API, redes, Switch Ethernet, Hub, protocolos Ethernet, Gigabit Ethernet, protocolo de redes virtuais, Spanning tree, IGMP, memórias partilhadas, estruturas em linguagem de programação, Linux

resumo

O presente trabalho pretende mostrar o que foi estudado e implementado a cerca de um API proposta para configurar as funcionalidades L2/L3 numa FPGA. Essas funcionalidades devem ser controladas e configuradas a partir de um processador. A comunicação entre o processador e a FPGA deve ser feita por intermédio de uma porta via Gigabit Ethernet.

Foi então implementada em linguagem C de programação uma dummy da FPGA, para simular a comunicação entre a API e uma FPGA, bem como verificar se os mecanismos implementados na FPGA, partindo de vários pressupostos funcionam correctamente. Foram implementados os mecanismos de hub, switch e o protocolo de VLANs referente à camada 2 do Modelo OSI.

keywords

FPGA, API, LANs, Switch Ethernet, Hub, Ethernet Protocols, Gigabit Ethernet, VLANs, Spanning tree, IGMP, Shared Memories in programming language C, Structs in Programming Language C, Linux

abstract

This work shows what was studied and implemented about configuring a FPGA with the specifications of L2/L3 using an API. These specifications must be configured and controlled by a processor. The communication between the processor and the FPGA must be done over Gigabit Ethernet. Using programming language C, it was done a FPGA dummy to simulate the communication between the API and FPGA and also to test the implemented mechanisms on the FPGA dummy.
The implementation of this work includes hubbing and switching mechanisms, and also the related VLAN protocol that runs over the switching mechanism.

Conteúdo

Índice de Fíguras

Índice de Tabelas

1 Introdução

A constante evolução dos computadores e dos periféricos associados a este ao longo dos tempos fez com que houvesse necessidade que eles pudessem comunicar entre si, formando assim uma rede de partilha de dados.

A necessidade da partilha de alguns periféricos, como por exemplo gravadores de DVDs, discos rígidos de grande capacidade, impressoras e de uma conexão via ADSL, faz com que haja grande interesse no desenvolvimento de mecanismos para intercomunicação de vários computadores a esses periféricos, pelo que é necessário efectuar vários investimentos para o desenvolvimento de dispositivos que possibilitem essa comunicação.

A *Internet* tornou-se o meio mais utilizado nos dias de hoje para se efectuar diversos tipos comunicação em diferentes âm*bit*os. Ela já nos possibilita efectuar comunicação com outras pessoas, via voz ou vídeo, permite efectuar negócios e transacções de dinheiro a longa distância, bem como é fonte de uma grande quantidade de informação sobre as mais variadas matérias que se pretenda estudar. Com esta evolução, é natural que se queiram efectuar vários tipos de redes de computadores para que se possa fazer chegar *Internet* aos mais diversos locais e tentar abranger uma grande área do globo terrestre com esta tecnologia. É natural que se queria também uma comunicação cada vez rápida, pelo que é feito um grande investimento na evolução de protocolos que possibilitem a transmissão de dados cada vez mais rápida. Para acompanhar esta constante evolução na transmissão de dados é necessário também um grande investimento nos dispositivos já existentes ou por desenvolver para que estes processem cada vez mais rápido a informação acompanhando a crescente melhoria na velocidade das ligações de *Internet*.

A partilha de *Internet* e de dispositivos comuns a vários computadores é feita por intermédio de *switches* e *routers*, na grande maioria dos casos. Pelo que estes são alvo de diversos estudos e implementações para se melhorar a capacidade e a velocidade de processamento de dados dentro destes dispositivos. Estes dispositivos formam o que é mais conhecido por redes *Ethernet* e usam todas as capacidades desta tecnologia. Os *switches* e *routers* permitem também fazer uma correcta gestão de redes partilhadas por vários computadores.

Para que se possam implementar estes dispositivos é necessário estar a par das novas tecnologias desenvolvidas pela área de Electrónica para que qualquer implementação destes dispositivos seja vantajosa a nível do melhoramento da capacidade e velocidade de processamento de dados dentro destes dispositivos.

Fazendo uma análise sobre os componentes e dispositivos electrónicos mais recentes e que possam satisfazer a necessidade de implementar um *switch Ethernet* com características que acompanhem o constante melhoramento nas redes *Ethernet* sobressaem-se as Field Programmable Gate *Array*s (FPGA).

As FPGAs são um dispositivo electrónico que introduziu uma nova categoria de *Hardware* reconfigurável e que apresentam funcionalidades definidas pelo programador e não pelos fabricantes, ou seja, é utilizada uma linguagem de programação para programar o dispositivo.

A FPGA é composta por um enorme número de blocos lógicos programáveis, que através de uma linguagem de programação (VHDL) permite simular o comportamento de cada circuito. Este dispositivo electrónico apresenta inúmeras capacidades. Através dele é possível simular um processador simples (MIPS, por exemplo), e ainda é possível na mesma FPGA programar um controlador vídeo, uma interface *serial* e por aí adiante.

As FPGAs mais recentes apresentam uma pequena quantidade de memória RAM e pequenos circuitos de apoio, para que se possa ter um sistema completo usando apenas um c*hip* FPGA previamente programado, carregando um ficheiro de configuração proveniente do computador para uma EPROM (ou memória *flash*). Esta inclui também interfaces USB para que se lhe possam enviar os ficheiros de configuração, bem como incluem uma fonte de energia para que possa ser inteiramente autónoma.

Obviamente que um *chip* destes com tais capacidades torna-se bastante mais caro, mas aliciante já que se pode programá-lo de uma forma personalizada e de acordo com os requisitos específicos de determinada implementação.

É aqui que surge o principal objectivo do trabalho que conduziu a esta dissertação: aliar as potencialidades de uma FPGA às redes *Ethernet*, e criar um conjunto de funções (*Application Programming Interface*) capaz de programar uma FPGA que apresenta uma implementação de modo a funcionar como *Switch Ethernet* ao nível das camadas 2 e 3 do Modelo OSI e alguns protocolos relativos a essas camadas tais como VLANs, *Spanning Tree*, *Internet Group Management Protocol* (*layer 2*) e *routing* estático (*layer 3*). A comunicação entre o processador e a FPGA deve dar-se por intermédio de *Gigabit Ethernet*.

De referir que dever-se-á utilizar todas as capacidades de uma FPGA para se conseguir configurar esta como *switch Ethernet*. É então necessário recorrer á sua memória RAM para guardar todas as configurações relativas aos diversos protocolos, bem como as informações referentes ao mais básico sistema de *switching*, que se resume em guardar um MAC origem proveniente de uma mensagem que chegou numa porta na memória da FPGA.

1.1 Breve descrição da dissertação

Inicialmente foram estudados todos os protocolos relativos a este projecto, bem como algumas APIs criadas para configurar estes protocolos em alguns dispositivos. Pelo que o segundo capítulo resume-se a explicar algumas tecnologias *Ethernet* que foram estudas, nomeadamente a *Gigabit Ethernet*, os dispositivos de *hub* e *switch* e os protocolos de VLANs, *Spanning Tree* e IGMP. No segundo capítulo é possível o leitor tomar partido de tudo o que já está implementado a nível protocolar, bem como pode se guiar por algumas API estudadas para entender quais as funções a implementar para os protocolos de *Spanning Tree* e IGMP. Alguns valores e expressões utilizados nos capítulos seguintes são também explicados neste capítulo.

Os capítulos 3, 4 e 5 descrevem a implementação realizada e descrevem os testes efectuados ao que foi implementado.

No capítulo 3 é feita uma descrição genérica do que foi implementado no âmbito da dissertação.

O capítulo 4 inclui uma descrição detalhada do que foi implementado para solucionar o que foi proposto na dissertação para se configurar uma FPGA como *Switch Ethernet* por intermédio de uma API.

No capítulo 5 é apresentado todos os testes e análise à implementação descrita no capítulo anterior.

Foi proposto realizar-se uma simulação de uma FPGA em C para interagir com um *Software* criado que inclui uma API capaz de programar a *dummy* da FPGA. Denominou-se o Software por api_sw. Foram implementados os mecanismos de *hub*, *switch* e o protocolo de VLANs a correr sobre a *dummy* da FPGA. Após isso foram efectuados alguns testes para verificar a fiabilidade do *Software* api_sw implementado e da *dummy* da FPGA.

1.2 Notação e terminologia

1.2.1 Notação Geral

A notação ao longo desta dissertação segue a seguinte convenção apresentada:

- Texto em *itálico* – para palavras em lingua estrangeira (exemplo, Inglês), também podendo ser utilizado para dar ênfase a uma determinada expressão;
- Texto **negrito** – sobretudo para dar ênfase a um conceito, palavra ou determida váriavel , biblioteca ou código de programação utilizado;
- Texto a **negrito** com "plicas" – para atribuir nome a certos processos ou tabelas presentes geradas pelo código;
- Texto a **negrito** com espaçamento simples – determinados trechos de código presentos ao longo do código.

1.2.2 Acrónimos

API	*Application Programming Interface*
ATM	*Asynchronous Transfer Mode*
BPDU	*Bridge Protocol Data Unit*
FPGA	*Field-Programmable Gate Array*
Gbe	*Gigabit Ethernet*
Gbps	*Gigabit per Second*
HSSDC	*Highspeed Serial Data Connection*
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engeneering Task Force*
IGMP	*Internet Group Management Protocol*
IP	*Internet Protocol*
L2	*Layer 2* do Modelo OSI
L3	*Layer 3* do Modelo OSI
LAN	*Local Area Network*

MARS	*Multicast Address Resolution Server*
Mbps	*Megabit per Second*
MIB	*Management Information Base*
MIPS	*Microprocessor without Interlocked Pipeline Stages*
MPLS	*Multi Protocol Label Switching*
MSTP	*Multiple Spanning Tree Protocol*
OSI	*Open Systems Interconnection*
PAM	*Pulse Amplitude Modulation*
QoS	*Quality of Service*
RAM	*Random-Access Memory*
RSTP	*Rapid Spanning Tree Protocol*
RSVP	*Resource Reservation Protocol*
STP	*Spanning Tree Protocol*
TCP	*Transmission Control Protocol*
TTL	*Time to Live*
USB	*Universal Serial Bus*
UTP	*Unshielded Twisted Pair*
VLAN	*Virtual Lan Area Network*

1.2.3 Símbolos gerais

η	Rendimento

2 Tecnologias de Redes *Ethernet*

2.1 Introdução

As redes locais espalhadas por todo mundo utilizam a tecnologia *Ethernet*. Os princípios de operação entre as muitas variações da tecnologia *Ethernet* são sempre os mesmos, sendo que as variações dessa tecnologia surgem sobretudo da necessidade de uma maior velocidade e melhor facilidade de uso.

Hoje em dia, uma das velocidades amplamente usada em redes locais é 10/100 Mbps, embora se deva referir que a *Ethernet* ao longo destes vários anos tem sofrido grandes alterações no seu desempenho de velocidade. Uma tecnologia já em franca expansão na sua utilização é a *Gigabit Ethernet* (1000Mbps), pelo que é de todo interesse o seu estudo e aplicação em diferentes projectos que necessitem uma rápida ligação entre dois ou mais dispositivos de rede.

De referir também que uma consequência do aumento de velocidade da tecnologia *Ethernet*, implicou uma alteração dos diferentes meios físicos usados para a transmissão de dados *Ethernet*. Foram utilizados por exemplo, cabos coaxiais grossos, cabos coaxiais finos, vários tipos de pares entrançados, bem como ultimamente, fibra óptica.

De seguida estudar-se-á a tecnologia *Gigabit Ethernet*, já que é parte integrante da forma de comunicação entre os diferentes blocos do projecto desta dissertação.

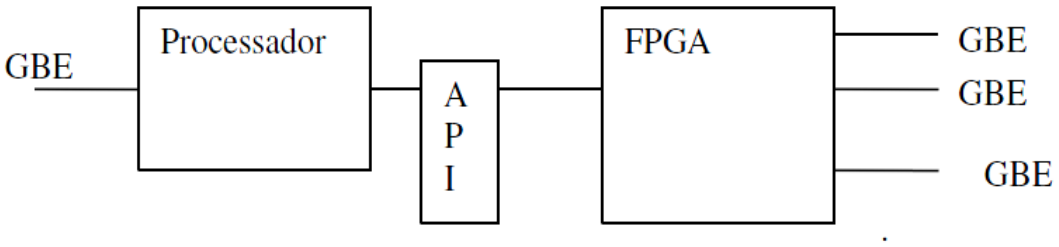

Figura 2.A: Diagrama de blocos integrantes no projecto de dissertação

2.2 *Gigabit Ethernet* (Gbe)

A *Gigabit Ethernet* é uma norma de camada física (PHY- *Physical Layer*) e de controlo de acesso ao meio (MAC- *Media Access Control*), especificado a camada de enlace ou de dados (*layer 2*) do modelo OSI. Esta norma é a base para toda a comunicação ponto-a-ponto entre os equipamentos de rede.

A necessidade de aumento da largura de banda nas "pontas" das redes (servidores e estações), bem como a necessidade de redução de custos provenientes das tecnologias levou à criação da tecnologia *Gigabit Ethernet*. O congestionamento em *backbones*, resultante da

utilização cada vez maior de aplicações multimédia (voz e vídeo) fez com que a *Gigabit Ethernet* fosse encarada como solução para este problema, na medida em que esta norma é compatível com a norma *Ethernet*, trazendo uma redução no tempo dispendido para aprendizagem da norma e uma redução de custos resultante de proteger todo o investimento já realizado em redes *Ethernet*.

Os objectivos da *Gigabit Ethernet* resumem-se à ligação entre equipamentos activos, como *Switches* e *Routers*, bem como à ligação a estações de trabalho de alto desempenho (GNIC).

A *Gigabit Ethernet* é definida pela norma IEEE 802.3z (1998/06) [5] para cabo coaxial e fibra óptica, pela norma IEEE 802.3ab (1999/06) [5] para UTP5. A *Gigabit Ethernet* define também, o modo ***Shared*** e ***Full-duplex***, sendo na prática usado apenas o modo ***Full-duplex***.

O modo ***Full-duplex*** em *Gigabit Ethernet* é um modo de comunicação em que um dispositivo envia e recebe simultaneamente sobre o mesmo link, duplicando desta forma a largura de banda, ou seja, para uma conexão *Gigabit Ethernet* (1000 Mbps) no modo *Full-duplex* esta possui uma largura de banda de 2000 Mbps (2Gbps).

Quando ao modo ***Shared***, este é um modo de comunicação em que o dispositivo é capaz de enviar ou receber, mas nunca envia e recebe simultaneamente, daí ter sido anteriormente referido que na prática apenas se usava o modo *Full-duplex*.

Para se usar a *Gigabit Ethernet* é necessário respeitar certos requisitos, um dos quais como é óbvio é esta ser compatível com redes 802.3. É necessário também manter o mesmo protocolo de nível MAC da *Fast Ethernet*, na medida em que para serem detectadas colisões é preciso que o comprimento mínimo da trama seja suficiente para encher de *bits* todo o comprimento da rede (**slot time**). Para se aumentar a velocidade de transmissão para 1000 Mbps é de referir que o comprimento ocupado por um *bit* no meio físico reduz 10 vezes em relação à *Fast Ethernet* (100 Mbps).

A *Gigabit Ethernet* para uma correcta utilização tem que seguir uma das seguintes opções:

- **Diminuição do comprimento máximo de rede para apenas 1 repetidor** entre duas máquinas, constituindo isto, numa redução do atraso. Com a generalização de cabos UTP (*Unshielded Twisted Pair*) em estrela centralizados em *hub*s, com limitações de distância na ordem dos 100 metros entre *Hub*/*Switch* e máquina resulta um comprimento de rede de 200 metros.
- **Aumento do comprimento mínimo de trama para 512 *bytes*** (4096 *bits*). É realizado sempre um ***carrier extension***, quando a rede tem que enviar uma trama inferior a 512 *bytes* a estação continua a transmitir um sinal especial até fazer o tempo dos 512 *bytes*. É de referir também, que existe um aproveitamento da margem de segurança das normas anteriores para que não haja um aumento no comprimento mínimo da trama.

Já foi referido anteriormente o ***slot time***, pelo que convém saber as regras de cálculo do ***slot time*** para o protocolo IEEE 802.3ab. Para o cálculo do slot time é necessário saber os diversos campos que condicionam o comprimento mínimo da trama:

Comprimento máximo de um segmento	$\boldsymbol{L = 100(m)}$
Comprimento máximo da rede	$2\ Segmentos + 1\ Repetidor$
Velocidade de Transmissão	$Vt = 1000\ Mbps = 10^9\ (bps)$
Tempo de bit	$bt = \frac{1}{Vt} = 10^{-9}(s) = 0.001\ (\mu s) = 1\ (ns)$
Velocidade de Propagação	$Vp = C \times \frac{2}{3} = 3 \times 10^8 \times \frac{2}{3} = 2 \times 10^8$(m/s)
Tempo de Propagação num segmento	$\frac{L}{Vp} = 0.5\ (\mu s) = 500\ (bt)$
Espaço de bit	$Lb = Tb \times Vp = 10^{-9} \times 2 \times 10^8 = 0.2\ (m)$
Atraso máximo de um repetidor	$Dr = 488\ (bt)$
Atraso máximo de um DTE	$Dd = 217\ (bt)$
Atraso máximo de um segmento	$Ds = 556\ (bt)$

Tabela 2.A: Parâmetros para *Gigabit Ethernet*

Ora se repararmos temos um atraso máximo de rede de 2 vezes os segmentos, de um repetidor e de dois DTE's (um a inicio e outro no fim), logo teremos:

$$(Dd1 + Ds1 + Dr1 + Ds2 + Dd2 = 2Dd + 2Ds + Dr) \times 2 = (2 \times 217 + 2 \times 556 + 488) \times 2 = 4068\ (bt)$$

Como o ***slot time*** terá que apresentar sempre uma margem de segurança, que será feita pela introdução de um sinal até perfazer os 512 *bytes* quando a trama a transmitir é inferior a 512 *bytes*, então poderemos calcular para uma trama mínima quantos *bits* terão que ser transmitidos para perfazer os 512 *bytes*:

$$Slot\ time = 2 \times Bits\ de\ Rede + bits\ margem\ de\ segurança = 512\ bytes = 4096\ (bt)$$

$$Bits\ margem\ de\ segurança = Slot\ time - 2 \times bits\ de\ rede = 4096 - 4068 = 28\ (bt)$$

Através deste cálculo de ***slot time***, apercebe-se que terão que ser efectuadas algumas alterações à camada MAC, uma das quais é o ***slot time*** passar para *512 bytes* (4096 *bits*) feito por recurso de *Carrier extension*. Existem mais duas alterações, uma delas é óbvia que é a alteração do formato da trama novamente pela introdução de uma extensão á trama, e outra menos óbvia que é a possibilidade de fazer *bursting*, termo que será explicado adiante.

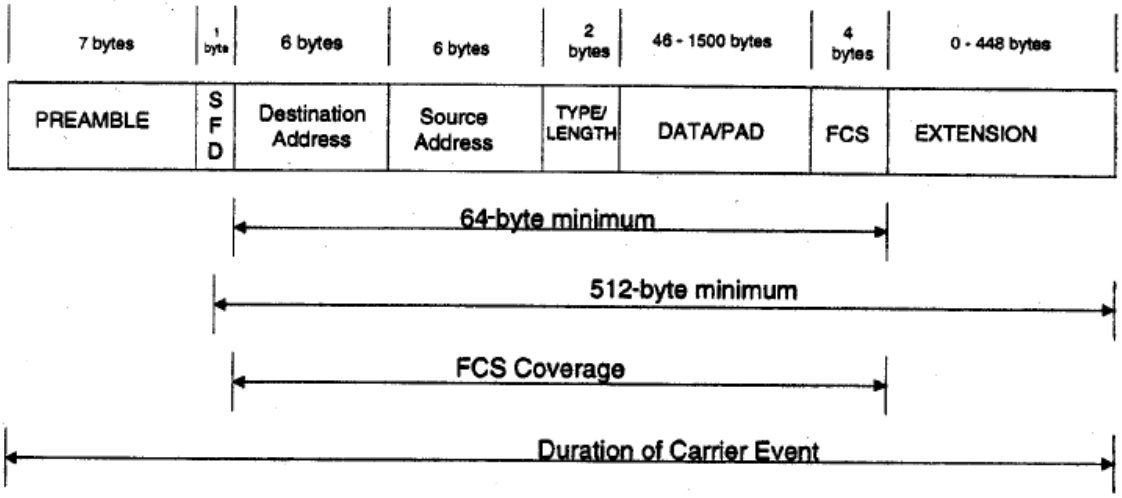

Figura 2.B: Formato da trama *Gigabit Ethernet*

Quando se calcula a **eficiência** para o pior caso em que se tem tramas de 64 *bytes* (512 *bits*), que são enviadas com *carrier extension* (4096 *bits*), com *preâmbulo* de 7 *bytes* (56 *bits*) e *inter-frame gap* (96 *bits*) verifica-se que esta é de aproximadamente 12%, pelo que o recurso a *bursting* é necessário para aumentar a eficiência no envio de tramas.

$$\eta = \frac{512\ bits}{(4096 + 96 + 64) bits} \approx 12\%$$

Ao recorrer-se ao *bursting*, a primeira trama é enviada normalmente com *carrier extension*, se for necessário, sendo que as restantes são enviadas a seguir separadas por um *inter-frame gap* essencial e sem *carrier extension* até expirar o *burst timer* (65536 *bit* time). Com este funcionamento de envio de tramas consegue-se atingir o objectivo de melhorar a baixa eficiência da *Gigabit Ethernet* para tramas menores que 512 *bytes* (aproximadamente 50%).

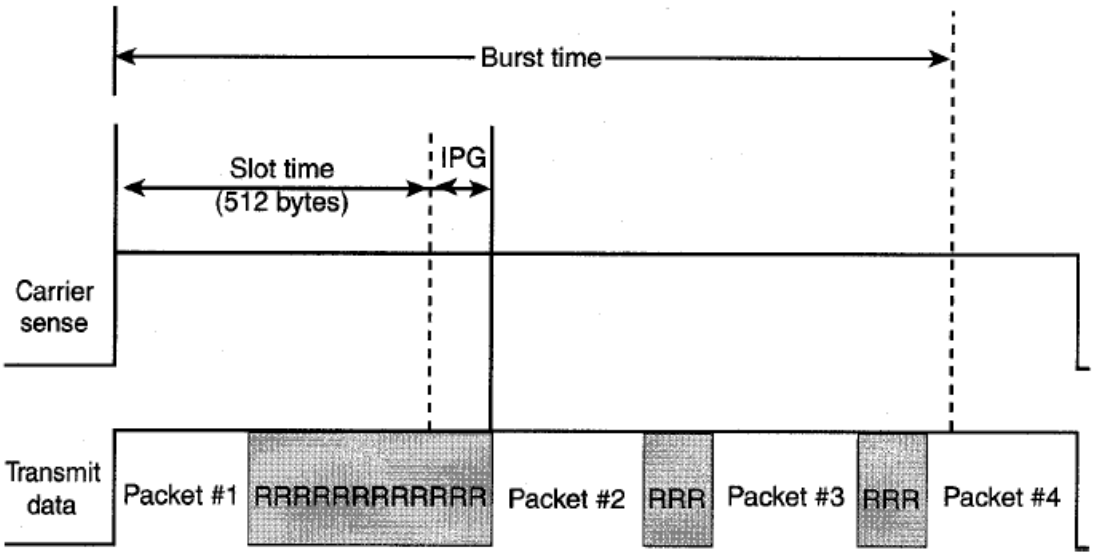

Figura 2.C: Transmissão de pacotes via *Gigabit Ethernet* recorrendo a *bursting*

Quando se recorre a *bursting* no envio de tramas verifica-se que o pior caso de 64 *bytes* (512 *bits*) que são enviadas num *burst timer* (65536 *bits*) com preâmbulo de (64 *bits*) e um *inter-frame gap* (96 *bits*) implica uma melhoria significativa na eficiência no envio de tramas.

Para o cálculo da eficiência no envio de tramas com recurso a *bursting* é necessário primeiro calcular o número de tramas que se podem transmitir num *burst timer*:

$$Tramas\ no\ burst = 1 + \frac{65536 - 4096}{512 + 96 + 64} \approx 93$$

Após este resultado, já é possível obter a eficiência em *bursting*, basta à fórmula anteriormente usada para cálculo da eficiência multiplicarmos o número de piores tramas possíveis (512 *bits*) pelo número de tramas no *burst*, bem como multiplicarmos o número de tramas no *burst* menos 1 (excluí-se a primeira do cálculo, já que esta é a única transmitida por *carrier extension*) pela soma do número de *bits* de dados, do preâmbulo e da *inter-frame gap*, sem esquecer de somar os 4096 *bits* da primeira trama enviada. Resulta daqui, uma melhoria significativa da eficiência em relação ao envio de tramas sem *bursting*.

$$\eta = \frac{93 \times 512}{4096 + 92 \times (512 + 96 + 64)} \approx 72\%$$

Ao falar-se de *Gigabit Ethernet* convém referir os diferentes tipos meios de transmissão que podem ser cabo de cobre, fibra óptica e par entrançado. Abaixo, será colocada uma tabela que refere os diferentes transcievers que são usados como norma:

Norma	Protocolo	Meio	Distância	Tipo de Ligações
1000Base-LX	IEEE 802.3Z	Fibra multimodo	Até 5 Km.	*Switch* a *switch*
1000Base-SX	IEEE 802.3Z	Fibra multimodo	Até 550 m.	*Switch* a *switch*
1000Base-CX	IEEE 802.3Z	Cabo coaxial	Até 25m.	*Clusters* de servidores e ligação entre *switches*
1000Base-T	IEEE 802.3ab	UTP5	Até 100m.	Ligações entre *switch*s a estações e servidores

Tabela 2.B: Comparação de vários factores que influenciam as ligações *Gigabit Ethernet*

A norma 1000BASE-X usa uma codificação 8b/10b, ou seja, este código codifica 8 *bits* de dados em 10 *bits* para a transmissão, pelo que apresenta uma redundância de 20%, que traz alguns beneficios, sendo os mais relevantes a ausência de componente DC, a fácil recuperação de clock, bem com a correcção de erros.

A boa densidade de transição deste código permite uma fácil recuperação de clock e uma sincronização byte/palavra, que é introduzida por uma regra do código que garante que 5 *bits* consecutivos não assumem o mesmo valor.

Este tipo de codificação garante um número par de *bits* 1 e 0 numa palavra, o que significa que o sinal é balanceado, eliminando desta forma a componente DC.

A fibra óptica é usada como meio físico de dois padrões: o 1000BASE-SX que utiliza lasers de comprimento de onda curto (0.85 μm) e o 1000BASE-LX que utiliza lasers de

comprimento de onda longo (1.30 μm). Os dois padrões podem usar fibras ópticas multimodo, embora apenas a norma 1000BASE-LX possa trabalhar com fibras ópticas monomodo.

A norma 1000BASE-CX usa cabos de cobre balanceados e blindados de 150 ohm. O tipo de conexão pode ser efectuado por intermédio de DB-9 ou por HSSDC (*High Speed Serial Data Connection*).

A norma 1000BASE-T utiliza 5 níveis de *Pulse Amplitude Modulation* (PAM), 8 estados de codificação Trellis, 4 dimensões, para além de equalização e *pulse shaping*. A sua taxa de transmissão é de 125 Mbaud por cada par UTP, operando em dual-duplex em cada par, ou seja, existe transmissão e recepção de dados simultaneamente e em ambas direcções.

A modulação PAM é baseada na amplitude do sinal para codificar dados. A sinalização binária transmite um 0 ou um 1, pela presença ou ausência de um impulso. A modulação tem 5 níveis de quantificação (-2, -1, 0, 1, 2), sendo transmitidos 2 *bits* de dados e alguma informação codificada em cada impulso. A largura de banda duplica, sendo que a transmissão ao efectuar-se a 125 milhões de símbolos por segundo (125Mbaud) traduz-se na transmissão de 250 milhões de *bits* por segundo (250Mbps), em cada par.

Para melhorar a transmissão recorre-se nesta norma, a uma codificação Trellis, que adiciona redundância ao sinal. O código Trellis utiliza o estado actual e a entrada de dados para determinar o estado seguinte. Na descodificação é calculado o próximo estado baseado nos dados recebidos e nos estados anterior. O descodificador escolhe então os dados com menor distância de Hamming, resultante da comparação entre o estado calculado e o estado actual. Este tipo de codificação serve para detectar e corrigir erros provenientes do sinal gerado.

O *pulse shaping* filtra o sinal de forma a minimizar os efeitos das frequências distorcidas no canal, fazendo com que haja uma melhoria na recepção do sinal.

Para finalizar, deve referir-se as vantagens e desvantagens que advém do uso da *Gigabit Ethernet*.

As principais vantagens são a popularidade adquirida pela tecnologia *Ethernet* e o seu reduzido custo. Apresenta vantagens ao nível da velocidade e desempenho em redes entre comutadores e servidores, bem como não introduzir nenhuma nova camada de protocolo a ser estudada.

A maior desvantagem é a norma *Gigabit Ethernet* não suportar QoS (*Quality of Service*), embora o IEEE esteja a desenvolver um protocolo que defina um esquema de prioridade muito próximo do QoS (802.1p).

O *Gigabit Ethernet* é uma tecnologia MAC e PHY, pelo que o problema da Qualidade de Serviço estende-se a protocolos de camadas superior como o 802.3x (Controlo de Fluxo), 802.1Q (redes virtuais -VLANs), 802.1p (Prioridade de Tráfego) e RSVP (Reserva de banda). Ou seja, apesar de termos um acréscimo na velocidade de dados, o *Gigabit Ethernet* pode comprometer o fornecimento de filas de prioridades para suporte de aplicações de voz e vídeo sobre IP em tempo-real.

2.3 *Hubs*, *Switches* e *Routers*

2.3.1 Introdução

Uma rede é constituída por *hosts* (estações de trabalho e servidores) interligados. A comunicação entre estes *hosts* é feita por intermédio de equipamentos de interconexão, sendo os de maior relevo os *hub*s, *switches* e *routers*. Os equipamentos anteriormente referidos estabelecem uma sub-rede de comunicação, embora todos eles apresentem diferenças importantes, implicando a sua utilização em situações particulares. As diferenças entre estes equipamentos surgem em várias dimensões:

- Na escalabilidade, ou seja, no tipo de sub-rede de comunicação que é pretendido realizar com o equipamento. Os *hub*s são usados para montar redes pequenas, os *switches* para montar redes médias e os *routers* para redes de grande dimensão.
- No alcance geográfico atingível, sendo que os *hub*s e *switches* são usados para redes de pequeno/médio alcance e os *routers* para redes de longo alcance.
- Na camada de protocolo de actuação: os *hub*s actuam sobre a camada física, os *switches* sobre a camada de enlace e os *routers* sobre a camada de rede. Tal como se pode averiguar pela figura abaixo que mostra as camadas do modelo OSI onde os diferentes equipamentos actuam.

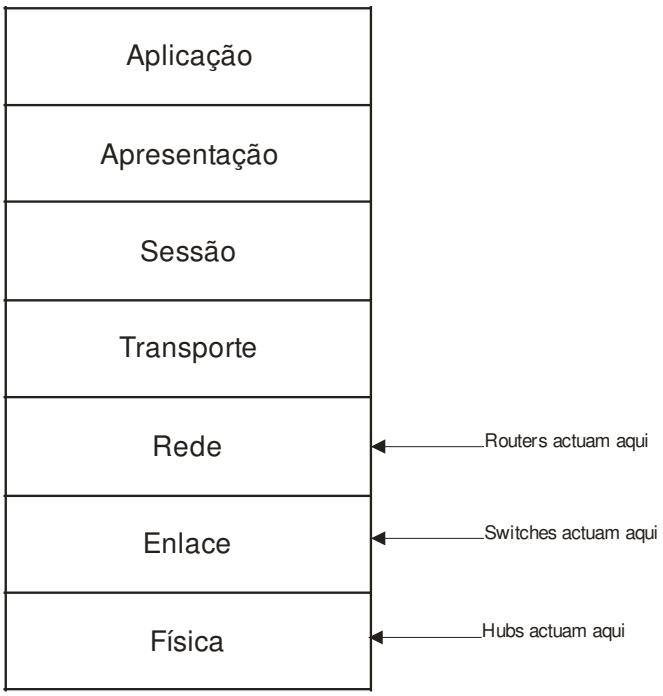

Figura 2.D: Modelo OSI mostrando onde os diferentes equipamentos actuam na sua camada

- Quanto ao preço, os *hub*s são mais baratos que o *switches* e estes por sua vez mais baratos que os *routers*.
- Na qualidade dos serviços fornecidos, já que a sofisticação de serviços vai crescendo de *hub*s para *switches* e destes para *routers*.

2.3.2 *Hubs*

Os *hub*s estão em constante desuso devido a dois factores, um dos quais o aparecimento da Fast/*Gigabit Ethernet* e do progressivo abaixamento de preços dos *switches* e *routers*. Em tempos, os *hub*s foram bastante utilizados em redes de pequeno e médio alcance, especialmente ligados directamente a estações de trabalho e funcionando a 10Mbps.

Os princípios de funcionamento são simples. Um *hub* possui várias portas para interligar equipamentos, sendo que qualquer um dos *hosts* conectados ao *hub* pode comunicar com outro pelo simples envio de uma trama para o *host* destino. O *hub* ao receber um sinal enviado por um dos *hosts*, retransmite-o em cada uma das outras portas. Pode-se considerar que o *hub* é um amplificador de sinal, já que o sinal que é recebido numa porta é amplicado e enviado para as outras portas. O mecanismo de *hubbing* é uma evolução do segmento *Ethernet*, que usava cabos coaxiais (10BASE -5 e 10BASE-2) para uma solução em que o cabo está logicamente presente dentro do *hub* e cada *host* liga-se individualmente ao segmento com o seu próprio cabo, tipicamente usando cabos de pares entrançados.

Devido às funcionalidades de um *hub* pode-se afirmar:

- O segmento *Ethernet* é partilhado entre todos os *hosts*. Se o *hub* for de 10 Mbps, então a sua largura de banda é de 10 Mbps para o tráfego total dos *hosts*.
- É permitida apenas a transmissão de uma mensagem de um *host* de cada vez, sendo esta a operação *half-duplex*.
- As colisões podem ocorrer num segmento.
- Todas as portas operam utilizando a mesma tecnologia *Ethernet*, tipicamente 10BASE-T ou 100BASE-TX.

Quando se afirmou que o *hub* é um "amplificador de sinal" é apenas uma simplificação. Para além disso o *hub* realiza as seguintes tarefas:

- Faz "imposição de colisão", certificando-se que cada vez que é detectada uma colisão sem ambiguidade para os *hosts* envolvidos.
- Regenera o sinal em cada porta.
- Faz o auto particionamento das portas, isolando o segmento das portas que estejam a causar conflito. O *hub* ao operar na camada física, qualquer falha presente numa das portas afecta todos os *hosts* conectados ao *hub*. Qualquer defeito num conector ou placa de rede, ou em qualquer dispositivo da sub-rede pode causar falhas do tipo de interferência no sinal ou tempo excessivo de colisão. O *hub* é capaz de fazer o auto particionamento de uma porta quando forem detectadas 30 falhas consecutivas. O restauro de uma porta a um segmento é efectuado sempre que não se detectar qualquer defeito numa porta.

Para finalizar, é necessário referir que um *hub* é transparente, já que os *hosts* não possuem informação de qualquer tipo de endereçamento, não sabem da existência do *hub*. Um *host* quando conectado a um *hub* não possui qualquer endereço que o identifique, ou seja, o *hub* não possui nem endereço MAC nem endereço IP.

Quando se pretende fazer redes maiores por intermédio de *hub*s é preciso relembrar que estes dispositivos têm um número limitado de portas. A solução encontrada é utilizar vários *hub*s, uns conectados aos outros usando uma das portas de cada repetidor. Em *hub*s que usam pares entrançados, o cabo que conecta um *host* a uma porta do *hub* é um cabo paralelo. Para

conectar um *hub* com outro é necessário utilizar um cabo cruzado para que a comunicação se efectue de forma correcta. Uma alternativa ao cabo cruzado é utilizar um cabo paralelo desde que a porta seja *uplink*, ou seja, uma porta especial já cruzada de um *hub*.

Os limites da interconexão de *hub*s surgem na tecnologia usada (10BASE-T e 100BASE-TX) que estabelece um limite no comprimento dos cabos, como um limite no número de mudança de *hub* que podem existir entre dois *hosts* quaisquer. A rede formada consiste de um único segmento partilhado, pelo que a sua largura de banda para todos os *hosts* resume-se a 10Mbps, podendo haver alguma saturação, constituindo mais uma limitação. A última limitação surge no domínio de colisões, sendo que qualquer *host* localizado em qualquer um dos *hub*s pode ter a sua mensagem transmitida colidindo com outro *host*, pelo que se verifica um incremento da taxa de colisão.

2.3.2.1 Características principais dos hubs

Algumas características diferenciam os diferentes *hub*s, sendo as mais relevantes as seguintes:

1. A tecnologia usada nas portas, tipicamente 10BASE-T e 100BASE-TX;
2. Expansão via módulos, possuindo apenas uma única fonte de alimentação para todos os módulos;
3. Suporte a *autosense*, estabelecendo a velocidade de 10 Mbps ou 1000 Mbps;

O sucessivo decaimento do preço dos *switches* fez com que os *hub*s entrassem em desuso, já que os *switches* apresentam um melhor desempenho derivado das suas características. Embora se deva referir que os *hub*s ainda são utilizados em redes pequenas com um baixo tráfego, bem como em redes mais antigas.

2.3.3 Switches

Os *switches* são o equipamento mais usado para montar redes de baixo/médio alcance, sendo utilizado em diversas aplicações de alta velocidade.

É verdade que os *hub*s são equipamentos de fácil implementação e que implicam um baixo custo, mas pelas razões já mencionadas há redes que não podem ser montadas apenas com *hub*s, pelo que houve a necessidade da criação de um novo equipamento de interconexão que contorne os problemas do *hub* e que permita fazer redes maiores abrangendo um maior número de *hosts*. Estes equipamentos terão que respeitar alguns princípios básicos:

- A largura de banda em cada porta deve ser dedicada a cada porta, evitando saturação de tráfego;
- O *switch* deverá ser transparente, sendo que os *hosts* não devem saber da existência do dispositivo, no sentido de não haver endereçamentos, embora se saiba que o *switch* para poder efectuar a gestão de rede, ele deva ser endereçável para as estações de gestão, continuando transparente aos demais *hosts*.

O princípio de funcionamento de um *switch* baseia-se em impedir que um sinal recebido numa porta seja imediatamente retransmitido em outra porta. Para isso, o *switch* ao receber o sinal numa porta, filtra os campos da mensagem, armazena-a internamente, e escolhe a porta de destino mediante a tabela de endereçamento interna, reencaminhando-a para a porta com o respectivo endereço. O destino recebe desta forma a mensagem em tempo distinto do da recepção da mensagem original na porta de origem, pelo que o comutador é então um equipamento do tipo *store-and-forward* (armazena-e-reenvia).

O *switch* para além da camada física do modelo OSI, actua também sobre a camada de enlace (*layer 2*), sendo nesta camada que se efectua a escolha do destino. A introdução da camada de enlace surge porque a escolha do destino necessita de examinar o pacote para descobrir o endereço de destino. Os conceitos de pacote e endereço de destino não existem na camada física sendo esta a principal diferença da camada de enlace para a camada física.

2.3.3.1 Algoritmo de um switch

Já foi dito em cima que o *switch* deverá ser transparente pelo que não poderá haver qualquer tipo de configuração prévia de endereços MAC relacionados com as portas dos *switches*.

O algoritmo de encaminhamento de um *switch* usa uma tabela de encaminhamento, que é actualizada sempre que é recebida uma mensagem numa porta com um MAC associado à porta. Inicialmente, a tabela de encaminhamento encontra-se vazia, sem nenhum MAC associado a uma porta. Ao receber um pacote destinado a um endereço MAC, o *switch* encaminha o pacote para a porta associada caso exista o endereço na tabela de encaminhamento. Se, pelo contrário, não for encontrado nenhum endereço, o pacote é encaminhado para todas as portas, menos na porta originária da mensagem. A esta técnica chama-se *flooding*.

Falta apenas explicar o modo como o *switch* efectua a aprendizagem das correspondências que ele mantém na tabela de encaminhamento. A técnica utilizada dá-se pelo nome de *backward learning* e é bastante simples: quando um pacote é recebido numa porta X com endereço de origem E_o e endereço de destino E_d, o *switch* pode não saber qual a porta que está associada ao endereço de

destino E_d, mas sabe que a origem E_o é alcançada pela porta P, pelo que o *switch* coloca o par (E_o,P) na tabela de encaminhamento. Quando mais tarde, chegar uma mensagem com um pacote endereçado com destino E_o, o *switch* encaminhará directamente para a porta P sem efectuar *flooding* a todas as portas.

De seguida é apresentado um exemplo que nos faz perceber todos os mecanismos de aprendizagem e de encaminhamento de mensagens.

As tabelas dos dois *switches* encontram-se inicialmente vazias, indicando a transparência que os *switches* devem apresentar. Se uma mensagem entrar pela porta 5 do *switch* à esquerda e com destino MAC AA verifica-se que é feito um *flooding* a todas as portas inclusive às portas do *switch* da direita já que nada se encontra nas tabelas de encaminhamente dos *switches*. A figura abaixo apresenta o mecanismo de *flooding* efectuado.

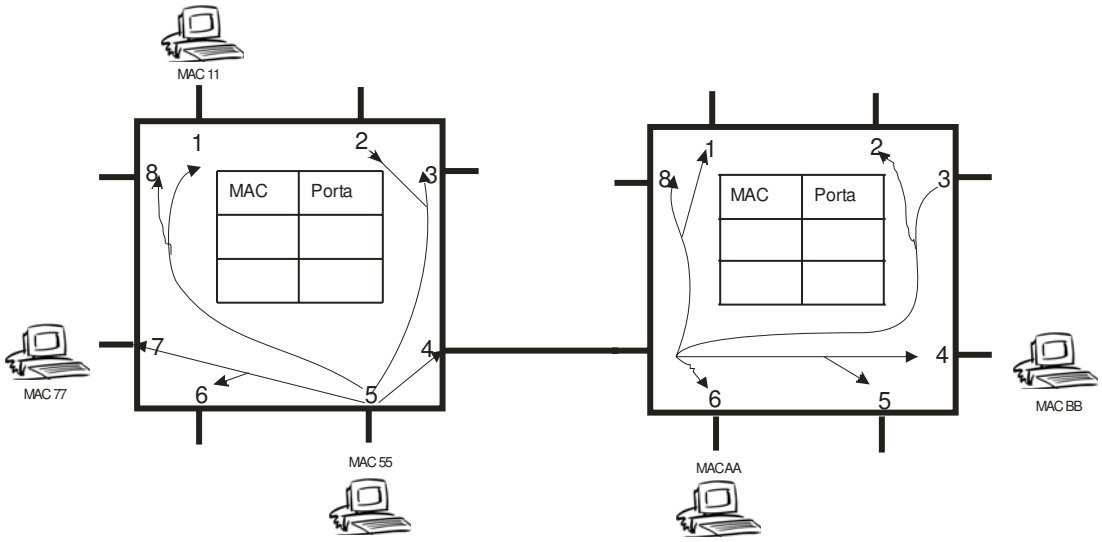

Figura 2.E: Figura exemplificando o *flooding*

Entretanto é associado às duas tabelas de encaminhamento a forma de chegar até ao MAC que originou a mensagem anterior, associando à porta 5 do *switch* da esquerda o MAC 55 e á porta 7 do *switch* da direita o MAC 55. É possível verificar isto na figura abaixo.

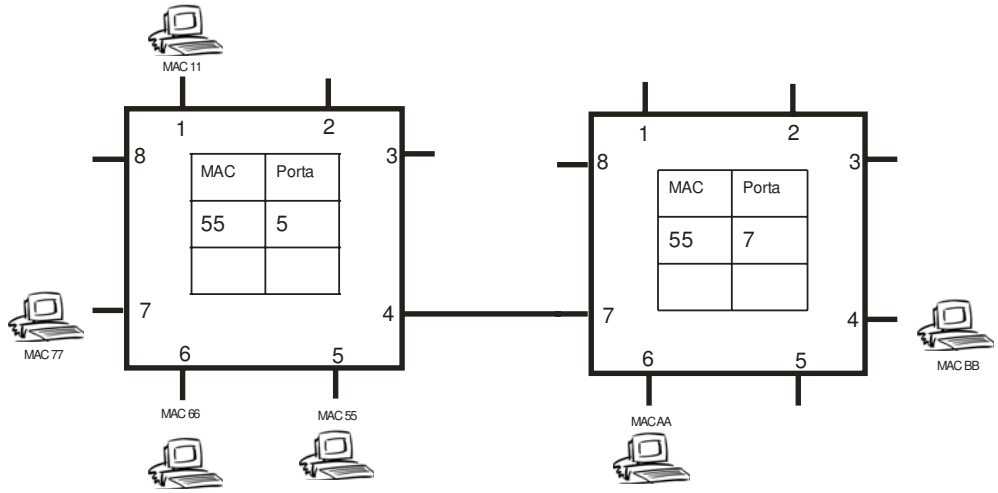

Figura 2.F: Tabelas de encaminhamento após a aprendizagem do endereço MAC 55

Se agora for enviado um pacote com destino o MAC 55 da porta 4 do *switch* da direita, já não é preciso recorrer ao mecanismo de *flooding* pois as tabelas de encaminhamento já aprenderam o endereço MAC 55, associando as portas que conduzem a esse destino. A figura abaixo mostra o caminho que o pacote percorre no exemplo que está a ser estudado.

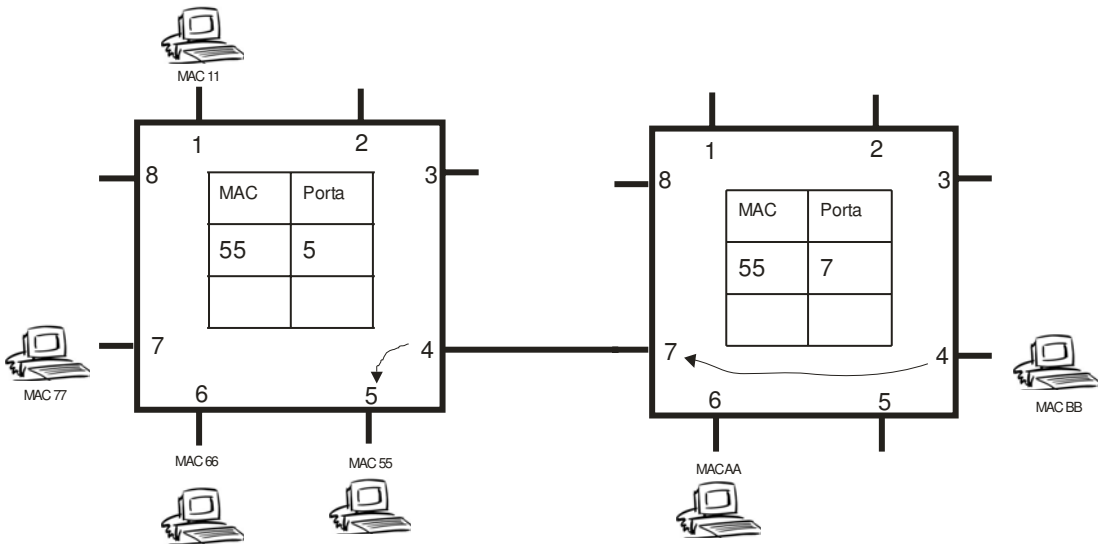

Figura 2.G: Pacote enviado pelo MAC BB com destino o MAC 55

De reparar que quando é enviado o pacote na porta 4 do *switch* da direita este tem origem no MAC BB, pelo que cabe às tabelas aprenderem esse MAC, associando a porta dos *switches* que lhes permite mais tarde fazer chegar uma mensagem destinada a esse MAC.

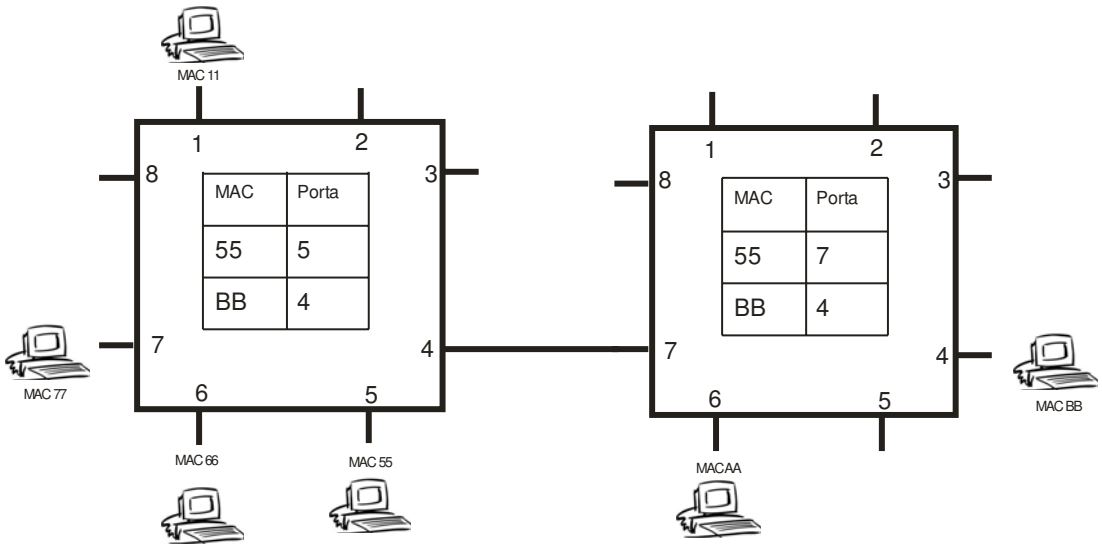

Figura 2.H: Tabelas de encaminhamentos finais dois dois *switches*

Os algoritmos descritos anteriormente foram implementados inicialmente num tipo de equipamento que se chama *brigde* (equipamento de duas portas que foi inicialmente utilizado para prolongar redes Ethernet). Pode-se encarar um *switch Ethernet* como uma *brigde* mas com múltiplas portas.

Só para finalizar, é necessário referir que mensagens de difusão (ou *broadcast*) ou de *Multicast* serão sempre encaminhadas pela técnica de *flooding*, bem como os algoritmos de *switching* são sempre implementados a nível de *Hardware*, para que seja aproveitado o máximo desempenho da velocidade nominal das portas.

Devido ao modo de operação de um *switch* ser do tipo armazenamento-e-reenvio, cada porta do *switch* é um domínio de colisão independente. Sendo que é de ter em conta que se uma porta estiver conectada a um *hub* a comunicação deve ocorrer em *half-duplex*, e colisões poderão ocorrer. Mas se a porta não estiver conectada a um *hub*, então nessa porta deverá operar o modo *full-duplex*, garantindo que não há colisões no segmento. A figura da página seguinte é elucidativa do que acima foi referido.

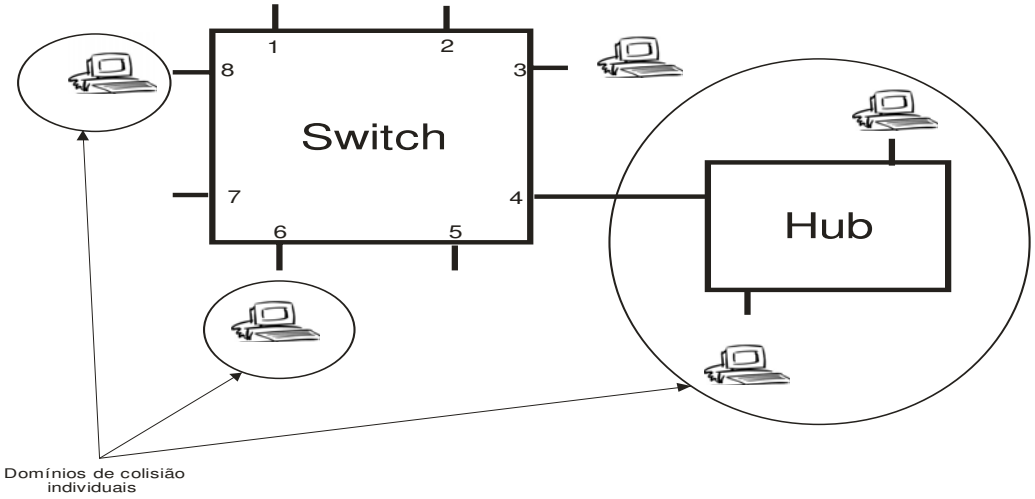

Figura 2.I: Domínios de colisão de um *switch*

A retransmissão de uma porta para outra não é imediata, pelo que desde já pode-se concluir que as portas não necessitam de operar todas à mesma velocidade. Pode-se então ter várias portas a funcionar a velocidades completamente distintas, ou seja, podemos ter uma porta a operar a 10Mbps, e outra a 1Gbps. Uma consequência que advém disto, é que podem ter portas com tecnologias diferentes, ou seja, umas portas com pares entrançados e outras portas com fibra óptica. Esta situação também contrasta com a velocidade de operação das portas de um *hub* que terá que ser a mesma para todas.

Também dever-se-á referir a política que é escolhida pelo *switch* quando são recebidas duas mensagens simultaneamente em portas distintas e com o mesmo destino ou quando portas operam em velocidades diferentes. A chegada de duas mensagens distintas implica a formação de uma fila, sendo que um dos pacotes transmitidos terá que esperar a sua vez. As consequências são óbvias destas filas de espera no *switch*, implicando um aumento de atraso na informação e uma possibilidade de perda de informação, na eventualidade do pacote ser descartado, já que os *switches* possuem uma memória interna limitada que permite guardar os pacotes em fila de espera para depois serem retransmitidos.

2.3.3.2 Características que diferenciam os switches

Algumas características que permitem diferenciar os vários modelos de *switches* são:

- O número de portas disponíveis varia de 8 a várias centenas;
- Vários tipos de tecnologia empregados nas portas, desde fibras mono/multimodo (10/100BASETX) e fibra óptica;
- Capacidade variada de matriz de *switching*, podendo alcançar centenas de Gbps;
- Suporte de gestão via protocolo SNMP, podendo fazer suporte às seguintes MIBs (*Management Information Bases*): MIB-II, *Ethernet* MIB, *Switch* MIB, etc;
- Opção de gestão tais como o suporte a *Switch*ed *Port Analyser* (SPAN);
- É facilitada a gestão de um *switch*, já que quase todos incluem um servidor *Web* embutido.

Os *switches* são então dispositivos de interconexão mais utilizados para implementar uma rede de pequeno/médio alcance, pelo que o seu baixo custo e alto desempenho fazem com que estes tenham há muito superado os *hub*s.

2.4 *Virtual Lan Area Network* (VLAN)

Uma das mais importantes características de *Switching Ethernet* é sem dúvida a capacidade de suportar redes locais virtuais, as VLANs.

O conceito de *Virtual Lan Area Network* (VLAN) inclui um conjunto de mecanismos que permite partilhar equipamentos, nomeadamente *switches*, para criar redes logicamente separadas, ou seja, redes virtuais. Desta forma, consegue-se aumentar o desempenho geral da rede. Em cada rede virtual verifica-se que todos os nós partilham a mesma rede, bem como cada uma define um domínio de *Broadcast* diferente. Verifica-se que as VLANs criam então um grupo de utilizadores que usam a mesma aplicação, embora se possam localizar em áreas físicas diferentes. Isto torna-se possível se na implementação de um dispositivo comutador (*switch*) se adicionar uma tabela adicional que agrupa os vários endereços MAC de servidores/hosts situados em locais diferentes. O número de *VLANs* varia de acordo com os padrões de tráfego, normas de gestão de rede, certas características comuns de diferentes grupos e com o tipo de aplicações.

Uma das propriedades da introdução de *VLANs* em redes *Ethernet* é o facto do tráfego entre VLANs ser restrito. Os *switches* encaminham o tráfego *Unicast*, *Multicast* e *Broadcast* para os segmentos da rede local virtual a qual o tráfego pertence. O papel do *switch* neste caso fica apenas reduzido a fazer a filtragem de *Broadcast*, bem como oferecer segurança e gerir o fluxo de tráfego de mensagens.

A criação de *VLANs* deve facilitar a reconfiguração da rede, ou seja, deverá ser simples a mudança de uma máquina de uma porta para outra no mesmo *switch* mantendo-a na mesma rede virtual.

A configuração de VLANs pode ser efectuada por intermédio de associação de portos a VLANs ou endereços MAC a VLANs, pelo que se designa a primeira opção por *VLAN layer-1* e a segunda por *VLAN layer-2*.

A definição da VLAN com base no endereço MAC apresenta a vantagem de permitir a mudança da porta de uma máquina, sem haja necessidade de reconfigurar o *switch*. Neste caso, é necessária a listagem de todos os endereços MAC, pelo que a sua gestão pode tornar-se mais complicada.

Para a configuração de um dispositivo comutador (*switch*) é necessário recorrer a *Software*, nomeadamente a uma interface de programação (API) que faça a gestão das VLANS, pelo que lhe compete criar a VLAN atribuindo um identificador, indicar quais os portos que estão associados a cada VLAN, bem como possibilitar a atribuição de diferentes prioridades a diferentes VLANS, permitindo desta forma o escalonamento de tráfego em diferentes conjuntos de portas associadas à mesma VLAN.

A implementação de VLANs num *switch* faz com que ocorram algumas acções:

- O *Switch* deverá manter uma tabela de endereços MAC para cada VLAN;
- Uma mensagem entra numa porta associada a uma determinada VLAN (identificada po um VLAN ID), e consequentemente o *switch* deverá procurar o VLAN ID na tabela de endereços;
- Se for recebida uma mensagem com MAC de origem que não conste da tabela de endereços, este deve ser agrupado à tabela.
- É preciso verificar o MAC de destino, se este constar da tabela de endereços e o VLAN ID for o mesmo. Neste caso a trama deverá ser encaminhada para a porta associada ao MAC, caso contrário, deverá ser feito um *Broadcast* a todas as portas associadas à VLAN em questão.

A possibilidade de interligação de *switches* não implica qualquer alteração nas VLANs, pelo que, em cada *switch* uma ou mais portas podem ser usadas para interligar a outros *switches*. Este tipo de implementação gera então o problema de uma porta de interligação poder receber tramas com VLANs distintas. A solução para este problema passa pela inclusão de um campo identificador da VLAN na trama de nível 2 (quando existe) ou ser transportado numa etiqueta adicional (semelhante ao *MPLS* mas sem troca em cada *hop*). Quando se fala neste assunto, também surge muitas vezes o termo *Pruning*, que evita reencaminhar tramas para *switches* onde não constem portos associados ao identificador de VLAN da trama a ser reencaminhada.

2.4.1 Protocolo 802.1Q

Após muito estudo e avaliação, o IEEE decidiu adoptar um novo cabeçalho norma *Ethernet* em 1998. Este cabeçalho contém uma tag de VLAN constituindo o protocolo 802.1Q [3].

Ao tentar compreender o protocolo 802.1Q verifica-se que os campos reservados para VLAN apenas serão usados pelos dispositivos de *switching* e *routing*, sendo na maioria dos casos desnecessário para as estações de trabalho. Daqui conclui-se também que placas de rede presentes em qualquer estação de trabalho, por muito obsoletas que sejam, nunca serão afectadas pela norma 802.1Q.

É preciso referir que o primeiro equipamento capaz de reconhecer uma rede virtual incluirá o cabeçalho 802.1Q e último dispositivo do percurso o remove, entregando ao destino a mensagem livre de cabeçalhos.

A figura a seguir mostra onde se inclui o cabeçalho do protocolo 802.1Q no actual modelo de referência, o modelo OSI.

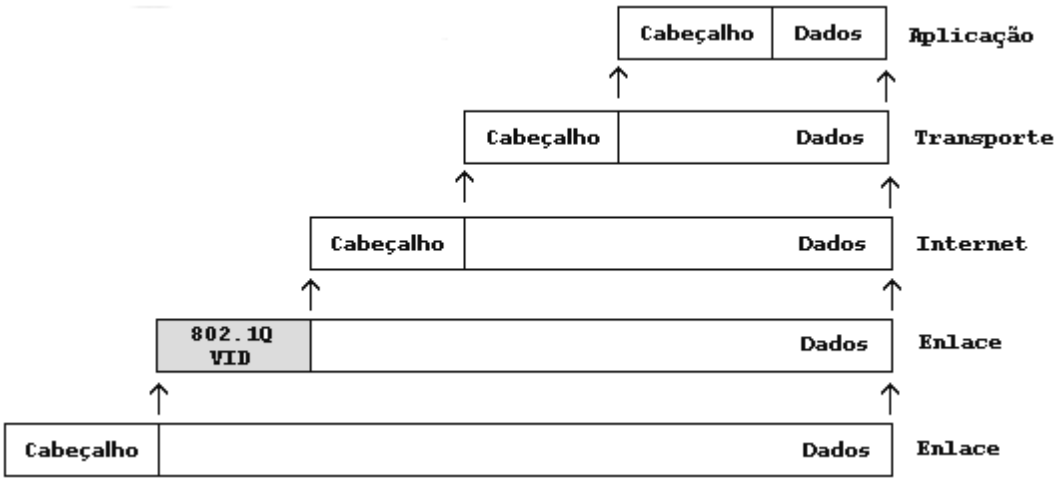

Figura 2.J: Etiquetação das tramas 802.1Q

Os campos que surgem resultantes do aparecimento deste novo cabeçalho são os seguintes:

- Tag Protocol *Identifier* (16 *bits*), para identificar uma trama etiquetada com 802.1Q usa-se o valor 0x8100;
- Priority (3 *bits*), resultante da norma IEEE 802.1p e que define a prioridade atribuída a cada rede local virtual. Os valores atribuídos para as prioridades das diferentes VLANs variam de 0 a 7. É implementado por *Hardware*;
- *Canonical Format Identifier* (1 *bit*), por norma este *bit* é sempre 0 para indicar que estamos perante uma trama CSMA/CD;
- VLAN *Identifier* (12 *bits*), este campo é usado para indicar a qual VLAN respeita a mensagem que se seguirá. É preciso mencionar que os valores estarão compreendidos entre 2-4094, visto que os valores 0, 1 e 4095 estão reservados.

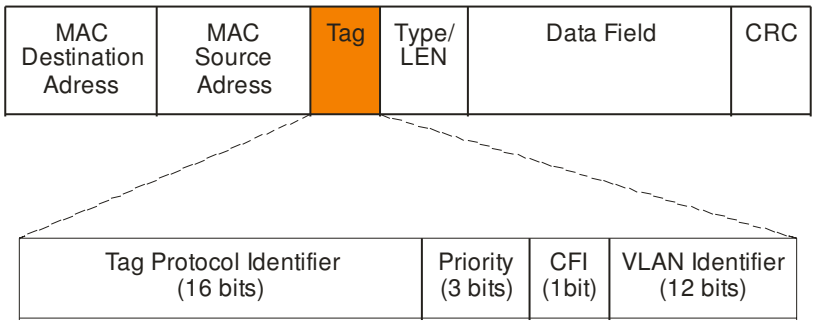

Figura 2.K: Etiquetação das tramas 802.1Q

Na figura acima verifica-se a forma de encapsulamento da trama, já com os campos 802.1Q incluídos. Também verifica-se que o novo cabeçalho é o cabeçalho da camada de nível 2, ou seja, da camada de enlace.

Apesar de haver alterações significativas com a introdução deste novo protocolo conclui-se que estas não afectam ao nível de *Hardware*, apenas afectam no encapsulamento e desencapsulamento que é feito pelo *Software* do equipamento.

2.5 *Spanning Tree Protocol*

2.5.1 Introdução

O *Spanning Tree Protocol* (STP) é um protocolo projectado para correr em *switches* e brigdes, sendo um protocolo de nível 2. Foi desenvolvido com o intuito de evitar congestionamento de rede resultante do *Broadcast*, e outros problemas relacionados com *looping* de topologia. Foi criado pela *Digital Equipment Corporation*, sendo mais tarde padronizado pelo IEEE pela norma 802.1d. De referir também que este protocolo de nível 2 funciona de forma muito semelhante aos algoritmos de *Kruskal* e *Prim*, que são aplicáveis nos grafos e calculam a distância mínima entre um vértice inical e qualquer vértice da rede de grafos.

A ideia do STP é detectar ou desabilitar o *looping* de rede, bem como possibilitar ligações de *backup* entre *Switches* e *Brigde*s. Para isso baseia-se nos seguintes príncipios:

- É atribuído a cada *brigde* um grupo de identificadores, um para o *switch* e outro para cada porta do *switch*.
- O identificador do *switch* designa-se por *Brigde* ID e é composto por 8 *bytes* (2 *bytes* para a sua prioridade e 6 *bytes* para o MAC do *switch*).
- Cada porta do *switch* possui um identificador que é composto por 16 *bits*, sendo que 6 dos quais constituem a prioridade e os restantes 10 servem para identificar o número da porta.

Para cada porta é atribuído um custo (*path cost*). Esse custo é atribuído dividindo 1000Mbps pela velocidade da porta. Tomando como exemplo uma porta com velocidade de 100 Mbps, o custo que deverá ser atribuído a essa porta é de aproximadamente 10. Abaixo é apresentada uma tabela de custo recomendada pelo IEEE 802.1d.

Velocidade de ligação	Custo Recomendado	Variação de custos recomendados
4 Mbps	250	100-1000
10 Mbps	100	50-600
16 Mbps	62	40-400
100 Mbps	19	10-60
1 Gbps	4	3-10
10 Gbps	2	1-5

Tabela 2.C: Custos recomendados e valores aceitáveis para atribuição de custos para o 802.1d

A tabela acima mostra a importância que a velocidade de ligação de uma porta toma para atribuição de custos, ou seja, quanto maior for a velocidade de ligação de uma porta, menor deverá ser o custo, visto que, se o objectivo desse protocolo é diminuir o congestionamento de rede, o tráfego suportado no mesmo espaço de tempo por uma porta é maior numa ligação de maior velocidade, já que esta entregará um maior número de pacotes no mesmo espaço de tempo, daí ser recomendado atribuir um menor custo a portas com velocidades maiores.

Retomando o objectivo principal do protocolo *Spanning Tree*, que evita o *looping* de rede, convém descrever o que é este problema que afecta as redes baseadas em *switches* e *brigde*s. Nas

duas figuras seguintes estão representadas duas situações observáveis em redes com switch Ethernet e que possuem loops físicos.

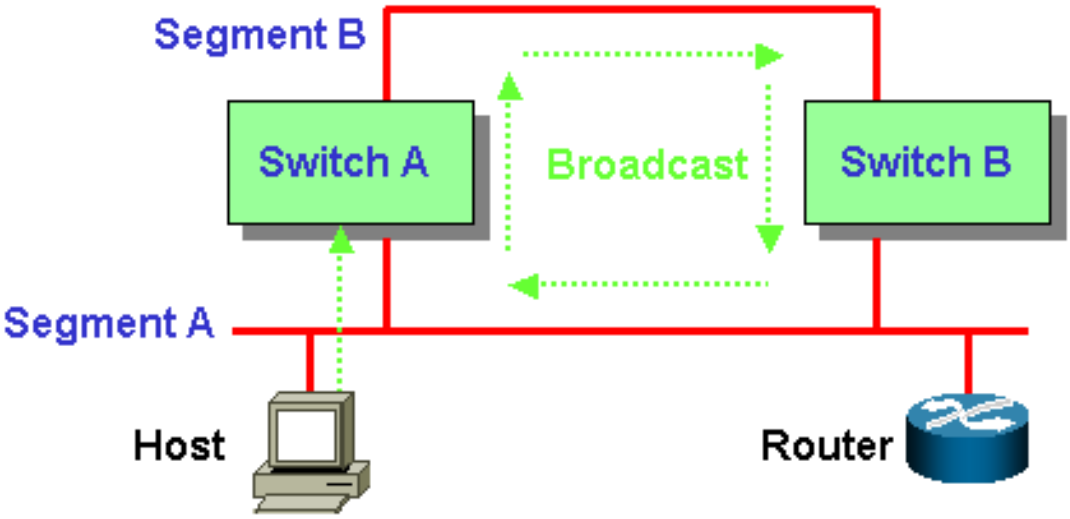

Figura 2.L: Dois segmentos de rede ligados por dois *switches* sem o STP activo e com *looping* de rede, quando é enviada uma mensagem com destino *Broadcast*

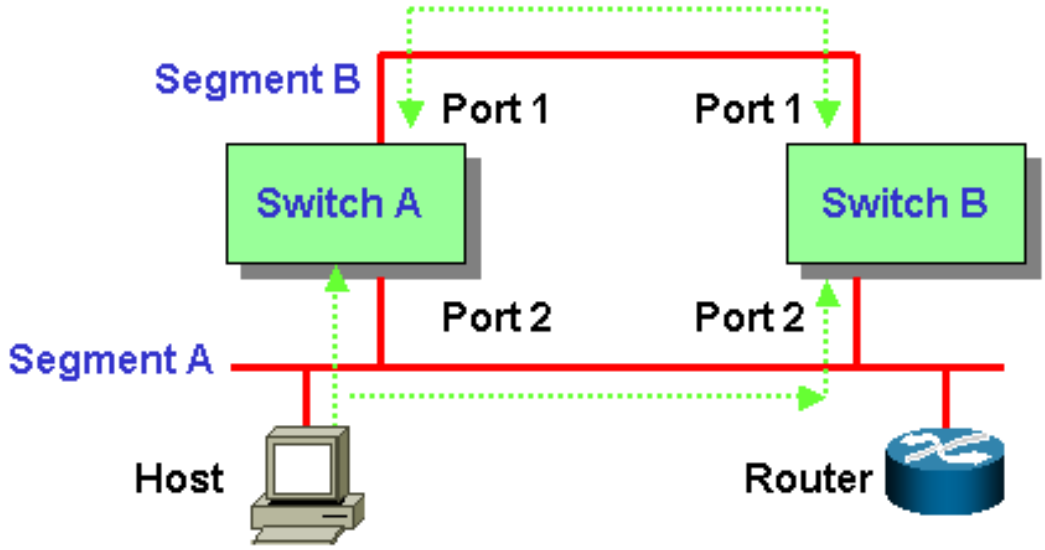

Figura 2.M: Dois segmentos de rede ligados por dois *switches* sem o STP activo e com *looping* de rede, quando é enviada uma mensagem com destino *Unicast* para o *router*

O problema recorrente da figura 1 surge quando é enviado um pacote com destino *Broadcast* (0xFFFFFFFFFFFF) pelo *Host*. Quando esta mensagem é recebida pelo *Switch* A, este identifica o destino como sendo de *Broadcast* e envia para o segmento de rede B. Este pacote chega ao *Switch* B por intermédio do segmento B e recebendo a mensagem *Broadcast*, reenvia-o para o segmento de rede A, formando-se assim um *looping* de rede.

O problema recorrente da figura 2 é um pouco diferente. Neste caso é enviado um pacote *Unicast* com destino para o *Router* presente na figura (origem e destino). O *switch* A e B recebem o

pacote e aprendem que o endereço MAC do *Host* está associado à porta 2. Entretanto, como o *switch* não aprendeu o endereço do *Router*, reenvia o pacote para o segmento B e obviamente é recebido pelo *Switch* B. Quando chega ao *Switch* B, a entrada que relaciona o endereço MAC do *Host* à porta 2 e é associada à porta 1, criando-se aqui instabilidade resultante da atribuição do endereço MAC e Porta em cada um dos *switches*, resultando em novo *looping* de rede.

A solução para este tipo de problemas foi encontrada na implementação do protocolo *Spanning Tree*, que ao descobrir um *loop*, bloqueia uma ou mais portas que causam redundância na rede. Para evitar falhas na topologia de rede, este protocolo também se encarrega de reconfigurar automaticamente as portas das *Brigde*s e *Switches* que fazem parte da rede, desbloqueando/bloqueando desta forma algumas portas. O STP para conseguir o seu objectivo, ou seja, uma rede livre de *loop*s, emite periodicamente pacotes *Multicast*, que se designam por BPDUs (*Brigde Protocol Data Unit*), e que constroem uma árvore de cobertura, e que inclui todas as bridge/switch de uma rede.

A criação da árvore de rede é feita por vários passos que convém serem descritos, para se perceber por inteiro a forma de implementação deste protocolo. A criação da árvore é feita mediante dois passos, um de identificação e outro de selecção de portas em cada *switch*.

O passo de identificação baseia-se nas seguintes regras:

- Inicialmente todos os *switches* assumem que são *root brigde*, sendo escolhida a *root brigde* numa "eleição" entre *switches* em que o que possuir menor *Brigde Identifier* (BID) ganha a eleição.
- A *root brigde* é considerada o topo da árvore e é a única na topologia, ou seja, só há uma *root brigde* na rede. Todas as portas da *root brigde* são portas designadas, ou seja, portas que estão a *forwarding* (portas redireccionando o tráfego de utilizador).
- Uma porta no modo *forwarding* é aquela que pode enviar e receber tráfego.
- Após determinar a *root brigde*, cada *switch* inicia um procedimento para determinar quais são as portas que devem estar a *forwarding* para alcançar determinado segmento de rede; são trocados entre *switches* pacotes BPDU para o efeito.

O passo de selecção de portas baseia-se nas seguintes regras:

- A localização da *root brigde* é determinante para que os outros *switches* determinem qual a sua porta de menor custo para alcançar a *root brigde*. Estas portas designam-se por *root port*, e, em cada instante, cada *switch* possui uma porta nesse estado.
- As portas que alcançam um determinado segmento de rede são designadas por *designated* ports e são escolhidas mediante o custo para alcançar a *root brigde* a partir de um segmento de rede. O *switch* com menor BID será escolhido, se o custo for o mesmo de um segmento de rede para a *root brigde*.
- As *non-designated* ports são portas que estão desabilitadas para um segmento mas que a qualquer momento podem ser activadas caso haja algum erro em algumas das portas designadas.
- Cada *switch* possui a topologia de rede, mantendo actualizada as tabelas de BPDUs.
- A porta de raiz normalmente está a *forwarding*.

Na figura seguinte representa-se um cenário hipotético de rede com três switch e onde existe redundância de ligações físicas (*loop*).

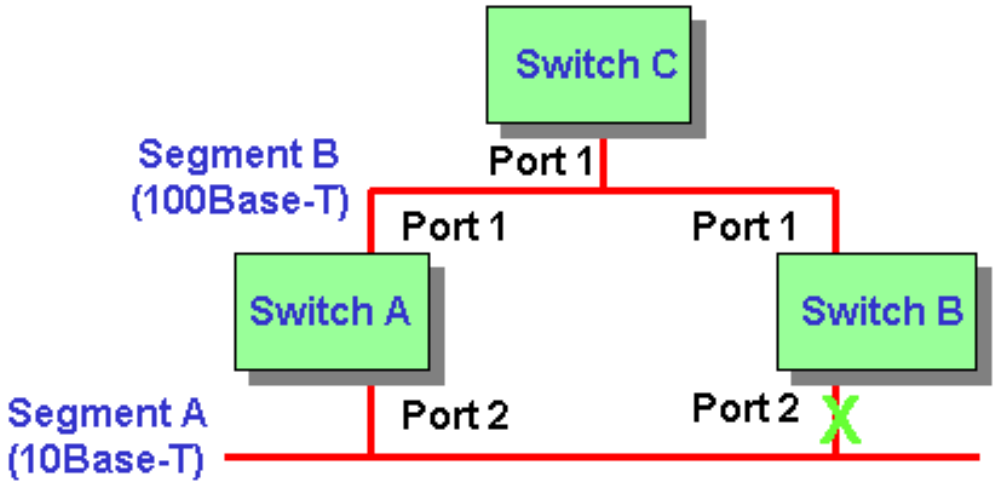

Figura 2.N: Escolha da porta designada

Verifica-se que sendo o *switch* C a *root brigde*, as portas 1 do *Switch* A e B estão activas e as portas 2 do *Switch* A e B são portas intermediárias para transporte de pacotes do segmento de rede A para o segmento de rede B. Ora pela figura pode-se concluir que a porta 2 do *Switch* B está bloqueada, pelo que deverá apresentar maior custo do que a porta 2 do *switch* A, ou então, se as portas 2 dos *Switch* A e B apresentarem custos semelhantes verifica-se que o *Switch* A terá que ter uma ID menor. Ora então ficamos com a porta 2 do *Switch* A a *forwarding* e *designated* port do segmento de rede A, e a porta 2 do *Switch* B a *blocking*.

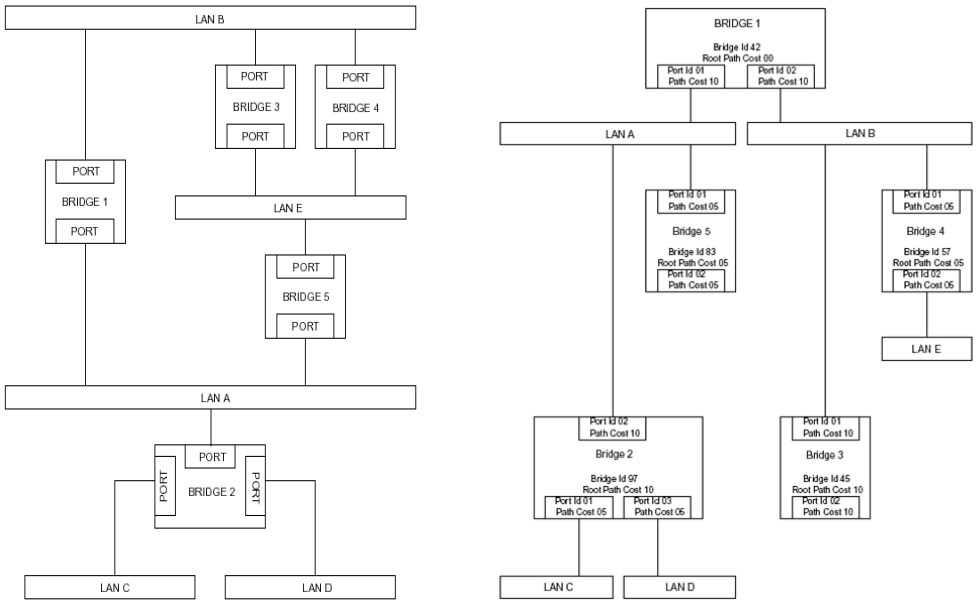

Figura 2.O: Árvore sem e após aplicação do *Spanning Tree Protocol*

Nas figuras acima é possível observar-se que na primeira temos um conjunto de *brigde*s sem o *Spanning Tree Protocol* activo, enquanto que na segunda verifica-se as mesmas *brigde*s já com o protocolo *Spanning Tree* activo, com respectivo custo de portas já aplicado.

Ao activar-se o *Spanning Tree* em todos os *switches*, aplicaram-se os respectivos custos nas suas portas correspondentes:

***Brigde* ID**	**Custo Porta 1**	**Custo Porta 2**	**Custo Porta 3**
1	10	10	-
2	5	10	6
3	10	10	-
4	5	5	-
5	5	5	.

Tabela 2.D: Custos aplicados às diferentes portas do exemplo da Figura 2.N e 2.O

A *Brigde* 1, como possui menor ID que todas as outras, fica como *Root Brigde*. Logo assume-se que a *Brigde* 1 é designada para as LAN A e B. Ao reparar-se na *brigde* 2, a *root* port é a porta 2 que liga ao segmento de rede LAN A. A porta 2 desta *brigde* apresentando custo de 10 o que faz com que o *Root path cost* desta *brigde* seja de 10. A *Brigde* 2 é também a *brigde* designada para os segmentos de rede LAN C e LAN D, já que é a única *brigde* que estabelece ligação entre estes segmentos e a raiz. Olhando para as *brigde*s 3, 4 e 5, a porta 1 de cada *brigde* é a *root* port já que é esta que as liga para a raiz. Embora a porta 2 de cada *brigde* ligue ao segmento E, é de referir que a porta 2 da *brigde* 3 fica logo bloqueada já que apresenta um custo de 10 enquanto as restantes *brigde*s apresentam custo de 5. Ora tem-se, então, duas *brigde*s (4 e 5) com *Root path cost* de 5 e a *Brigde* 3 com *Root path cost* de 10, surgindo mais um problema que o protocolo *Spanning Tree* terá que resolver, ou seja, qual das *brigde*s com *Root path cost* 5 ficará designada para o segmento de rede E? O critério de desempate neste protocolo, quando duas *brigde*s apresentam o mesmo custo para a raiz, é escolher a *brigde* que apresenta menor ID, neste caso será a *Brigde* 4, pelo que a porta 2 da *Brigde* 5 é bloqueada.

2.5.2 *Brigde Protocol Data Unit*

Falou-se bastante dos BPDUs anteriormente, mas convém referir que tipo de pacotes são estes, qual ou quais os seus objectivos, bem como quais os campos que os constituem.

Antes de mais é de notar que existem dois tipos de BPDUs:

- *Configuration BPDUs* (BPDUs de Configuração);
- *Topology Change Notification* (BPDUs de mudança de topologia).

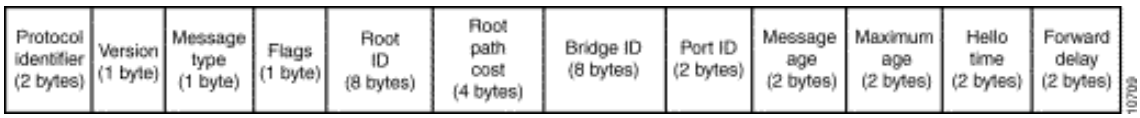

Figura 2.P: Estrutura de um BPDU

A figura acima mostra uma estrutura de um BPDU. Podem-se considerar alguns campos triviais, na medida em que o seu nome por si só já dita para que eles servem. Os campos que revelam

interesse para serem estudados são os que definem valores de tempo, já que são estes os responsáveis pela gestão da topologia de rede.

Focando em primeiro lugar os campos relativos a 3 temporizadores que estão presentes em cada *switch*, temos o *maximum age*, *hello time* e o *forward delay*.

Cada *switch* é configurado com um valor máximo de idade de mensagem. No campo *Maximum age* é configurado o tempo para a idade máxima que cada mensagem está num *switch*, sendo que se a idade for superior àquela que foi configurada pelo *switch*, o BPDU de configuração é descartado.

O campo *hello time* indica de quanto em quanto tempo o *switch* transmite um BPDU de configuração, se este for a raiz.

O campo *forward delay* indica o tempo que um *switch* demora para bloquear uma porta. A não existência de um campo deste género levaria a criação de *loopings*.Assim salvaguarda-se que uma porta não passa do estado *blocking* para o estado de *forwarding* directamente. Inicialmente a porta fica num estado de *listening* (escuta) durante o tempo que é estabelecido pelo campo *forward delay*. Só depois de esperar esse tempo é que, se a porta não foi bloqueada, entra num estado activo.

De referir também que ao inicializar-se um *switch*, todas as portas devem estar bloqueadas. Só depois, com a troca dos diversos BPDUs de configuração, é que ocorre o estado de escuta e posteriormente, se for caso disso, é atribuída a porta o estado de *forwarding*, após o tempo de *forward delay*. A figura abaixo mostra a sequência de estados que uma porta toma.

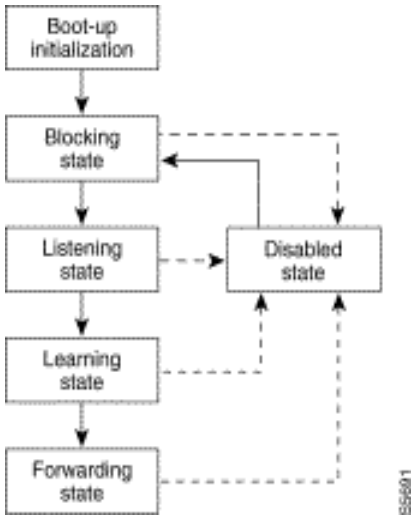

Figura 2.Q: Estados das portas

Por fim, é de referir que o campo message age mostra o tempo decorrido desde o envio da mensagem pelas portas designadas no *Root Brigde* que serviu de base para essa mensagem.

As mensagens de configuração descem na árvore de cobertura, no sentido da *Root Brigde* para baixo. A propagação na árvore de cobertura de uma mensagem proveniente da raiz é feita de uma forma simples: a *Root Brigde* emite um BPDU de configuração às *brigde*s vizinhas, e estas por sua vez transmitem a mensagem às suas vizinhas, espalhando assim a mensagem por toda a rede.

Um *switch* recebe BPDUs de configuração de *switches* que estão mais próximos da raiz que ele, ou seja, não existe BPDUs de configuração enviados no sentido da raiz. Para que seja possível

enviar BPDUs no sentido da raiz foi introduzido um tipo de BPDUs de notificação de mudança de topologia (NMT). Quando um *switch* detecta mudança de topologia são enviados BPDUs de NMTs na sua *root* port. Ao receber um NMT o *switch* responde com um BPDU de configuração com o valor de campo de flag alterado para indicar que é um BPDU de reconhecimento. Este processo repete-se até chegar á raiz. Quando a raiz é notificada de mudança de topologia, esta encarrega-se de transmitir BPDUs de configuração com o campo flag indicando que houve mudança para toda a rede durante um certo tempo. Os *switch* abaixo da raiz realizam a mesma tarefa, notificando toda a rede da mudança de topologia.

2.5.3 Evoluções do protocolo *Spanning Tree*

2.5.3.1 Rapid Spanning Tree Protocol (802.1w)

O *Spanning Tree Protocol* (802.1d), apesar de ser utilizado ainda nas redes actuais, apresenta sobretudo uma deficiência ao nível da convergência da formulação e reformulação da árvore que inclui todas as bridges, pois esta é demasiado lenta para certas aplicações. Para isso, foi criado pelo IEEE, o *Rapid Spanning Tree Protocol* (RSPT), cuja norma é dada por 802.1w. Os padrões 802.1d e 802.1w [4] são completamente compatíveis.

Na tabela abaixo indica-se a equivalência de estados do STP para o RSTP.

Estado operacional	Estado da porta no STP	Estado da porta no RSTP
Activa	*Blocking*	*Discarding*
Activa	*Listening*	*Discarding*
Activa	*Learning*	*Learning*
Activa	*Forwarding*	*Forwarding*
Desactivada	*Disabled*	*Discarding*

Tabela 2.E: Tabela de equivalência de estados do STP para o RSTP

As diferenças entre o *Spanning Tree Protocol* (STP) e o *Rapid Spanning Tree Protocol* (RSTP) resumem-se ao seguintes pontos:

- **Três estados de porta**: O RSTP apresenta 3 estados de porta, enquanto o STP apresenta 4 estados de porta enabled e um de disabled, ou seja, os estados de "*Blocking*, *Listening* e Disabled" do STP são apenas um único estado de "*Discarding*" no 802.1w.
- **Fast Aging**: Na implementação do STP só a *Root Brigde* pode enviar BPDUs de configuração para mudança de topologia, sendo que os demais *switches* apenas efectuam alteração nos campos necessários, e fazem "*relay*" desta BPDU para os *switches* mais próximos através das suas *designated* ports.

No RSTP (802.1w) todos os *switches* são capazes de notificar eventuais mudanças de topologia nos seus BPDUs. Define-se um intervalo de 2 segundos (campo do BPDU: *hello time*) em que os *switches* geram os seus próprios BPDUs e enviam-nos por intermédio das suas *designated* ports, e se um *switch* passar um intervalo equivalente a 3 BPDUs, ou seja 6 segundos, em que não recebe nenhuma mensagem do seu vizinho, este assume que o *switch* não faz parte da topologia RSTP e faz o bloqueio da porta conectada ao vizinho permitindo desta forma uma mudança rápida

muito mais rápida na topologia do que a idade máxima (*max age*) do STP, já que a convergência é *LINK* a *LINK*.

- **Introdução de portas alternativas e portas de *backup***: Na situação em que há duas ou mais portas presentes no mesmo segmento de rede, apenas uma dela desempenha a função de *designated* port, sendo que as outras tomam a função no RSTP de *alternative* ports. No caso de haver três ou mais portas, então estas são encaradas como *backup ports*.

Uma *alternative* port é uma porta que propõe um caminho alternativo para a *Root Brigde*d da topologia em *switches* não designados. Em condições normais, a *alternative port* assume o estado de *discarding*, só caso a *designated port* do segmento falhe, é que a *alternative port* assume a função de *designated port*.

Uma *backup port* é uma porta de um *switch* não designado que não recebe BPDUs.

- ***Edge* e *Non-Edge* ports**: O RSTP define dois tipos e portas, as *Edge* ports e as *Non-Edge* ports.

Uma *Edge port* são portas que não têm outro *switch* ligado a elas, surgindo da evolução do *port-fast* do STP, só que em vez de bloquear a porta ao receber BPDUs, a porta transforma-se em *Non-Edge ports*.

Uma *Non-Edge* port é uma porta que opera em *FULL-DUPLEX* e que conectam um *switch* na outra ponta ou a um *hub* respectivamente. São portas ponto-a-ponto ou *shared*.

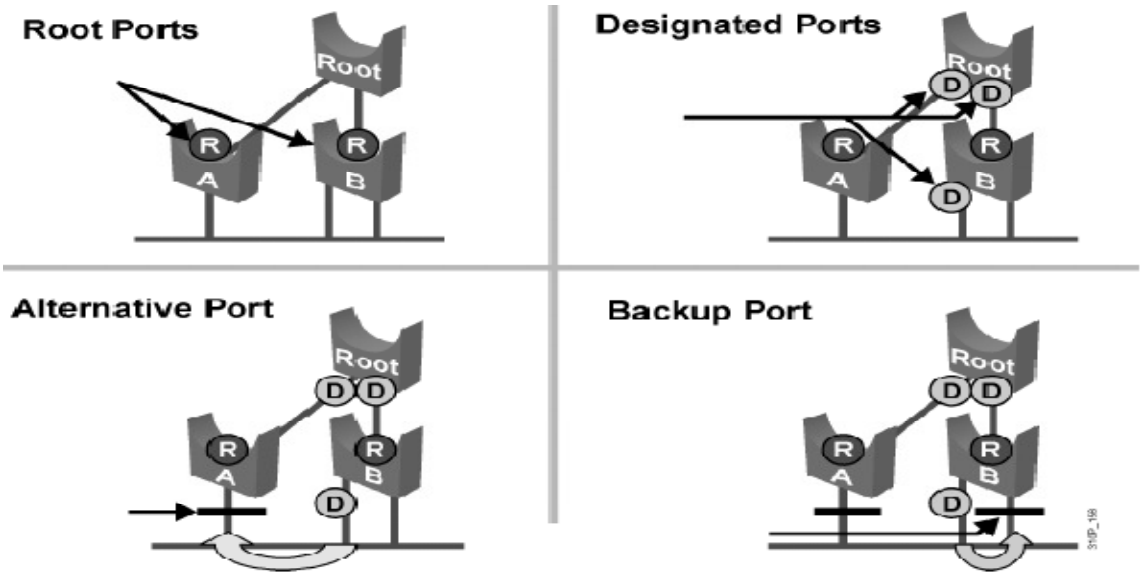

Figura 2.R: Diferentes tipos de porta no RSTP

2.5.3.2 Multiple Spanning Tree Protocol (802.1s)

A última evolução do protocolo STP, dá por nome de *Multiple Spanning Tree Protocol* (MSTP) que surgiu no âm*bit*o da deficiência do STP e do RSTP em ambientes com múltiplas VLANs. Nesse âm*bit*o o IEEE lançou o protocolo 802.1s [3], que mapeia um conjunto de VLANs que partilham a mesma topologia RSTP.

O MSTP define múltiplos STP/RSTP cada um associado a um conjunto de VLANs.

Ao agrupar múltiplas VLANs numa só instância, faz com que o tráfego de BPDUs seja reduzido, bem como, o tempo de convergência seja mais rápido.

Cada instanciação MSTP tem uma topologia RSTP independente das outras instanciações MSTP. O MSTP faz o balanceamento das instanciações para que o tráfego das VLANs de determinada instância possa usar caminhos alternativos de outras instâncias.

A instanciação MSTP implica um grupo de VLANs que partilha a mesma topologia RSTP, formando uma *REGION*. Todas as VLANs que fazem parte de um processo MSTP estão ligadas à instância 0 (Ist0) por default. Dentro da instanciação 0, as diferentes *REGION*s comunicam entre si, trocando BPDUs. Instâncias MSTPs não enviam BPDUs fora da sua *REGION*, somente dentro instância Ist0. Dentro da *REGION* os *switches* trocam BPDUs inerentes às diferentes instâncias que podem existir, cada uma delas contendo um "id" da instância de origem além de outras informações importantes ao processo.

O *Common Spanning Tree* (CST) é responsável pela interconexão de Ist0s em diferentes *REGION*s, permitindo desta forma uma comunicação entre diferentes *REGION*s. As *REGION*s podem então ser vistas como uma "*brigde* virtual" ao correr o CST.

Cada *switch* deverá ser configurado da mesma forma no que diz respeito ao mapeamento das VLANS para as suas respectivas instâncias para que estejam na mesma *REGION*.

O conjunto de Ists em cada *REGION* e de CST que interconectam as Ists dão-se pelo nome de CIST (*Common and Internal Spanning Tree*).

2.5.4 Configurando uma API para o STP

A *Application Programming Interface* para o protocolo de *Spanning Tree* terá que incluir o seguinte conjunto de funções:

- Selecção do Modo de *Spanning Tree*- função que deverá seleccionar o protocolo a correr em cada *switch* entre MSTP, RSTP e STP.
- Criação de grupos de VLANs para as diferentes instâncias – Esta função só deverá estar disponível se o modo dos *switches* tiver activo em MSTP, e deverá seleccionar o valor do grupo de VLANs e o seu espectro (por exemplo, pode-se ter um grupo de VLANs de 0 a 31 e um espectro de VLANs de 1 a 4094).
- Configuração dos parâmetros de STP, RSTP ou MSTP:

  - Configuração do temporizador *forward delay* – configura o tempo que o *switch* espera para configurar uma porta com o modo bloqueado.- Configuração do temporizador *hello time* – configura de quanto em quanto tempo o *switch* demora a enviar um BPDU de configuração.

  - Configuração do temporizador *max age* – configura o tempo que o *switch* espera por um BPDU, descartando este se passar o tempo configurado.

  - Configuração da prioridade de uma *brigde* – configura uma prioridade do *switch* alterando o seu MAC para que se possa definir qual a *root brigde*.

  - Alteração automática de *root* – configura um *switch* como *root*, alterando as prioridades dos restantes *switches* para que a prioridade máxima seja para o *root* em questão.

  - Agrupar um grupo de VLANs a uma determinada instância.
- Configuração de uma porta de acesso – configura uma porta como *Edge* port.
- Configuração de uma *REGION* e seu respectivo *revision number* – configura o nome da *REGION* e um *revision number* de 0-65535.

2.6 *Internet Group Management Protocol*

O *Internet Group Management Protocol* (IGMP) é um protocolo que possibilita a troca de informações entre um dispositivo e o *router Multicast* mais próximo, determinando se um pacote *Multicast* é enviado para um grupo específico. Cada grupo apresenta um endereço único que o identifica, sendo que os membros adicionam-se aos grupos e trocam mensagens entre si. O IGMP é também considerado uma extensão ICMP (*Internet Control Message Protocol*), sendo que as suas mensagens são encapsuladas nos datagramas IP e a sua versão 2 está descrita no RFC 2236 [2], embora já exista uma terceira versão desenvolvida do IGMP, que está descrita no RFC 3376 [17], sendo esta última menos utilizada que a versão dois.

Uma mensagem endereçada a um grupo é transmitida a todos os membros do grupo, e se o grupo for aberto, um utilizador não precisa de ser membro para enviar uma mensagem para este.

Quando o grupo é fechado, apenas os membros do grupo podem enviar mensagem para este.

De seguida apresenta-se as três versões do protocolo IGMP, sendo dado mais ênfase à versão 2.

2.6.1 IGMP v1

As especificações desta versão do IGMP encontram-se descritas no RFC 1112 [1].

Esta versão de IGMP apresenta dois tipos de mensagem: *Membership Query* (Pedido de participação) e *Membership Report* (Relatório de participação).

Uma aplicação ao iniciar um *socket Multicast*, a pilha de protocolos TCP/IP do dispositivo, envia automaticamente uma mensagem do tipo *Membership Report*. Esta mensagem é referente a um determinado grupo *Multicast*, indicando que quer participar deste grupo e o dispositivo determina também o endereço MAC do grupo.

São enviadas mensagens a todos os dispositivos de uma rede a cada 60 segundos, mensagens do tipo *Membership Report* de forma a verificar se existe pelo menos um utilizador dentro da sub-rede interessado em receber tráfego desse grupo. O *router* envia até 3 mensagens do tipo *Membership Query* espaçadas de 60 segundos sem que haja recepção de nenhuma resposta. Se não for recebida qualquer resposta a essas 3 mensagens, cabe ao *router* determinar o fim do envio de tráfego daquele grupo para aquela sub-rede. O *router* envia mensagens do tipo *Membership Report* destinadas a todos os dispositivos de rede, através do IP 224.0.0.1 e com valor TTL igual a 1.

De referir também que apenas um *router* por LAN está encarregado de enviar queries, sendo que o intervalo de queries pode ser configurado entre 60 a 120 segundos.

Um pacote IGMP v1 é constituído por 64 *bits*, com os campos Versão, Tipo, *Checksum* e o endereço de grupo.

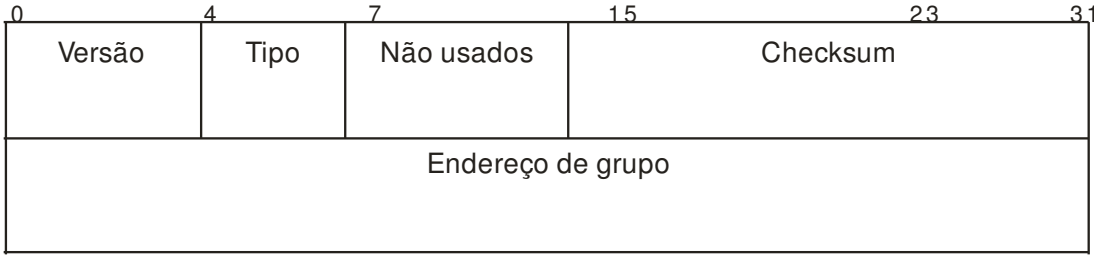

Figura 2.S: Pacote IGMP v1

Acima é apresentado um formato de mensagem IGMPv1, que é constituída pelos seguintes campos:

- O campo Versão, neste caso, será preenchido com a versão 1.
- O campo Tipo pode assumir o valor 1 se a mensagem for um *Query*, e o valor 2 sefor um Report.
- O campo *Checksum* permite verificar se a mensagem apresenta algum tipo de erro.
- O campo Endereço de Grupo apresenta o endereço do grupo *Multicast*.

2.6.2 IGMP v2

A segunda versão do IGMP veio substituir a versão 1 e é considerada actualmente a versão padrão.

Nesta versão existem 4 tipos de mensagens:

- *Membership Query* – Pedido de participação;
- *Membership Report* v1 – Relatório de participação para a versão 1;
- *Membership Report* v2 – Relatório de participação para a versão 2;
- *Leave Group* – Saída do grupo.

O funcionamento é em todo semelhante à versão anterior, sendo a principal diferença, a introdução de um novo tipo de mensagem, o *Leave Group*. Este tipo de mensagem é utilizado sempre que um dispositivo pretender comunicar ao *router Multicast* local a sua intenção de abandonar um grupo. O *router*, por sua vez envia uma mensagem do tipo *Membership Report* para esse mesmo grupo, tentando determinar se existe mais algum dispositivo que continue interessado em receber tráfego desse grupo. Se o *router* não receber qualquer resposta durante um tempo configurável (tipicamente 3 segundos), o *router* para de encaminhar tráfego para aquela interface.

A inclusão nesta versão de IGMP de mensagens do tipo *Leave Group* (Saída do grupo) veio reduzir a latência de saída de um grupo, quando se compara com a primeira versão de IGMP. O tráfego desnecessário e sem utilidade é assim cessado muito antes nesta versão padrão de IGMP do que na primeira versão.

As técnicas utilizadas para evitar o tráfego de mensagens do tipo *Membership Report*, resumem-se:

- Um dispositivo de rede quando recebe uma mensagem do tipo *Membership Query*, antes de enviar um *Membership Report* inicializa um contado para cada participação em grupo. Este contador é configurado com uma escolha aleatória de 0 a D segundos. Ao expirarem os D segundos é então enviada um *Membership Report* para o grupo, pelo que os *Membership Report* são propagados dentro de um intervalo de D segundos.

- Quando é enviado um *Membership Report*, este possui um endereço de destino IP referente ao grupo com o campo TTL com valor igual a 1, sendo que todos os participantes da rede verificam que foi enviado um relatório. Se um dos dispositivos de rede percebe que já foi enviado um *Membership Report* para o grupo ao qual pertence, o contador dele para automaticamente não gerando nenhum *Membership Report*. É então enviado um *Membership Report* apenas para cada grupo dentro de uma sub-rede.

A figura seguinte mostra um diagrama de tempo de tempo de operação do IGMPv2.

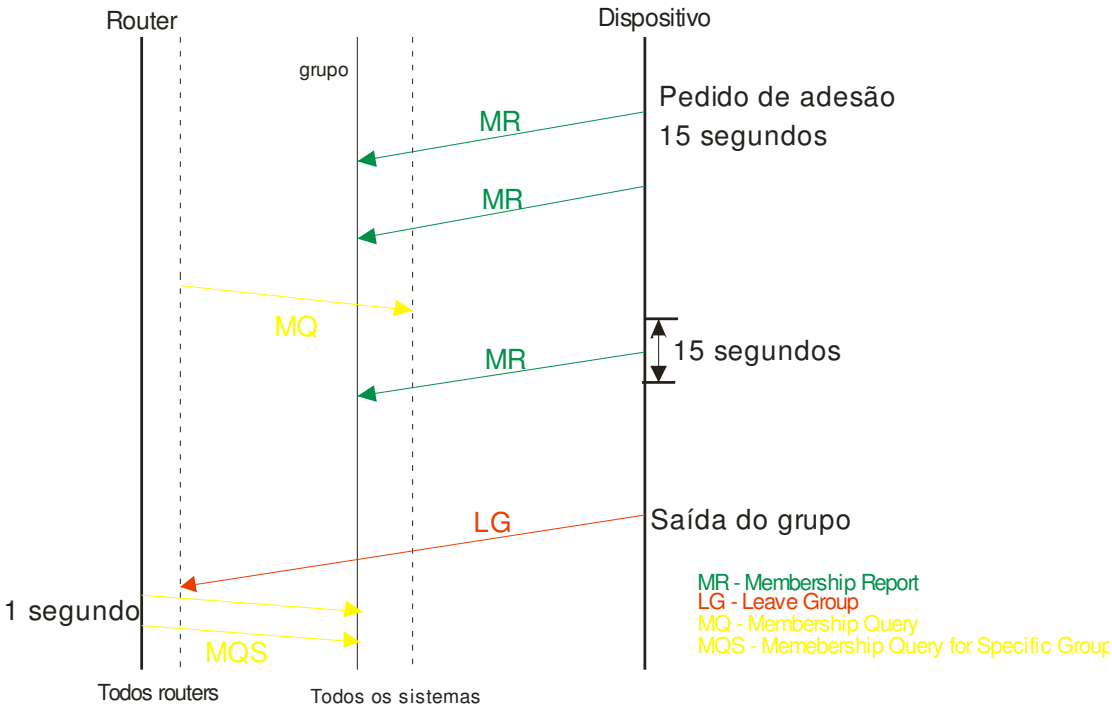

Figura 2.T: Diagrama de tempo de operação do IGMPv2

O pacote IGMPv2 possui 64 *bits*. Apesar e ter o mesmo número de *bits*, tem campos diferentes, sendo retirado o campo da versão, foi adicionado o campo de tempo máximo de resposta, e o campo tipo pode suportar mais tipos de mensagens.

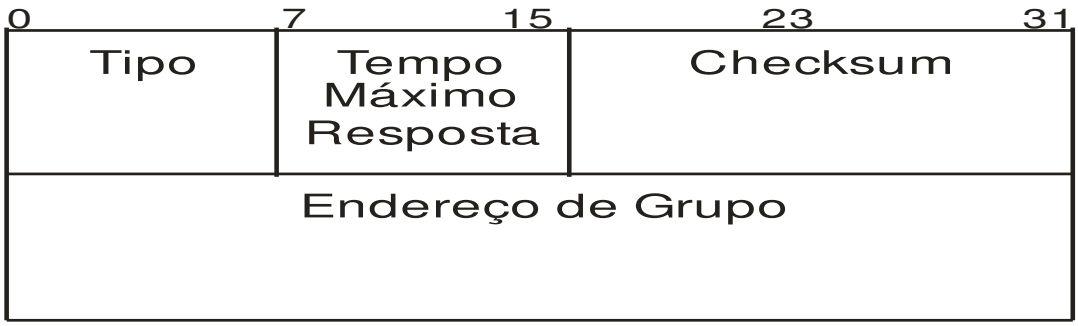

Figura 2.U: Formato da mensagem IGMPv2

A figura 2.U apresenta os campos constituintes de um pacote IGMPv2, mostrando quantos *bits* são reservados para cada campo. Os campos do pacote IGMPv2 possuem as seguintes informações:

- O campo tipo especifica o tipo de mensagem, podendo assumir os seguintes valores:
  - 0x11: específica uma mensagem do tipo *Membership Query*. Esta é enviada pelo *router Multicast* existindo *Membership Query* Comum que é utilizado para reconhecer quais os grupos que possuem membros em determinada rede e o *Membership Group-Specific Query* que é utilizado para determinar se um grupo específico possui membros em determinada interface.
  - 0x16: especifica uma mensagem do tipo *Membership Report*, sendo esta enviada por um dispositivo com o intuito de sinalizar a sua participação num grupo específico.
  - 0x17: especifica uma mensagem do tipo *Leave Group*, sendo enviada por um dispositivo *Multicast*.
- O campo Tempo Máximo de Resposta é um campo utilizado em mensagens do tipo *Membership Query* e especifica o tempo máximo que um dispositivo *Multicast* espera até ao envio de um *Membership Report* correspondente. É este campo que permite ao *router* acertar a latência de saída.
- O campo *Checksum* é um campo constituído por 16 *bits* e é utilizado em tipos de mensagens, encarregando-se de evitar falhas nas mensagens.
- O campo Endereço de Grupo especifica um endereço válido de um grupo *Multicast*, sendo utilizado nas mensagens do tipo *Membership Report*.

2.6.3 IGMPv3

A versão 3 do protocolo IGMP [17], encontra-se implementada desde de 2002. Com esta versão é possível os hosts especificarem a lista de hosts dos quais pretentem receber tráfego. O tráfego é então bloqueado dentro da mesma rede. As alterações nesta versão do protocolo em relação ao anterior também contemplam a possibilidade dos hosts poderem bloquear os pacotes de rede proveniente de fontes que enviem tráfego indesejado. Este protocolo já possui implementações nas versões mais actualizadas do *Linux*, *Mac* e *Windows*.

Este protocolo é compatível com as versões anteriores de IGMP e permite um uso mais eficiente da largura de banda.

Existe em Anexo A desta dissertação um artigo, que se estudou sobre uma das formas de implementação de uma API para o protocolo de IGMP, que poderá ser utilizada como forma de partida, para uma implementação posterior no âmbito deste projecto.

3 Descrição genérica da solução implementada

3.1 Conceito de *Application Programming Interface* (API)

Uma *Application Programming Interface* (API) é um conjunto de funções e procedimentos acessíveis apenas por programação de modo a constituir um *Software* que permita interagir com um *Hardware* sem que as implementações a nível de *Software* sejam evidentes a um utilizador comum.

Um exemplo de um *Software* que apresenta um grande conjunto de funções e procedimentos na sua API é um sistema de operação. A sua API permite a um programador ter acesso a ficheiros, codificar dados ou por exemplo, criar janelas. Uma API referente a um sistema de operação não se ocupa de tarefas cruciais para estes, como por exemplo manipulação de blocos de memória e acesso a dispositivos, sendo estas tarefas atribuídas ao *Kernel* e quase nunca programáveis.

A utilização de APIs tem-se verificado na introdução de *Plugins* que complementam a funcionalidade de um programa, sendo que os programadores do *Software* principal fornecem uma API específica para que os outros autores criem *Plugins*, podendo desta forma estender as funcionalidades do programa aos utilizadores comuns.

3.2 Descrição genérica do *Software* api_sw para programar um *switch Ethernet*

O objectivo principal desta dissertação era implementar um *Software* especial que faça a programação de uma FPGA de modo a que esta funcione como um *switch Ethernet*. O objectivo seria criar um *Software* que abstraisse o utilizador de todas as rotinas que terão que ser feitas para a programação do *Switch Ethernet* sobre uma FPGA. Para isso é necessário criar um *Software* de fácil uso e com rotinas de programação que possibilitem uma correcta configuração da FPGA, constituindo estas rotinas a API.

Esta API deveria ser responsável pela configuração das funcionalidades de *Switching*, bem como os protocolos de nível 2, nomeadamente as VLANs e o *Spanning Tree*, e os protocolos de nível 3, o IGMP e *routing* estático (RIP). Isto envolve um grande conjunto de funções programadas a nível da API para que se consigam ter todos os protocolos a funcionar correctamente.

Na prática, o trabalho conduncente a esta dissertação resumiu-se ao estudo dos protocolos mencionados anteriormente e a uma simulação de comunicação entre um processador e uma FPGA, utilizando um *Software* que inclui um conjunto de rotinas que permite a FPGA configurar portas, associar uma VLAN a uma porta, bem como configurar o aging time de uma porta (ou seja, é necessário apagar uma entrada de MAC do *switch* sempre que não se receba um pacote em X segundos).

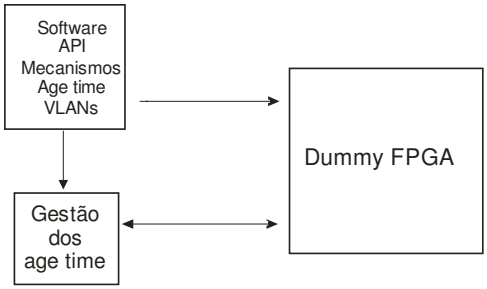

Figura 3.A: Diagrama de blocos sobre a solução implementada

A figura 3.A apresenta um diagrama de blocos respeitante à solução implementada. Foram implementados três processos. Um *Software* configurador da *dummy* da FPGA que comunica com esta por intermédio de uma memória partilhada e que configura todas as tabelas presentes na *dummy* com os parâmetros que o utilizador introduz no *Software*. A comunicação entre a API e a *dummy* da FPGA é feita apenas num sentido, já que é suposto a FPGA ter autonomia nos seus mecanismos associados. A decrementação do *agetime* das portas é feita ao nível da API e foi feito um processo de gestão destes. Como é necessária a configuração por parte do *Software* api_sw de todos os *agetime*s das portas presentes na tabela do processo de gestão dos *agetime* das portas, então é efectuada uma comunicação apenas num sentido, do bloco de *Software* api_sw para o bloco de gestão dos *agetime*. É suposto também, o processo de gestão dos *agetime* das portas comunicar com a *dummy* da FPGA, na medida em que é necessário sempre que o *agetime* de uma porta expirar comunicar à FPGA que é preciso apagar as entradas de endereço MAC respeitantes à porta. Este processo deverá também receber comunicação proveniente da *dummy* da FPGA, já que sempre que esta aprenda um endereço MAC deverá comunicar ao processo de gestão de *agetime* das portas que é necessário começar a decrementar o *agetime* respeitante à porta, ou então actualizar o *agetime* da porta sempre que se verifique a recepção ou o envio de um pacote na porta. Daí haver comunicação nos dois sentidos.

De referir que a primeira fase do trabalho descrito nesta dissertação foi desenvolver uma simulação de um dispositivo de *hub* em que o *Software* api_sw de programação consegue programar quais as portas a funcionar como *hub* e programar a FPGA de modo a que quando entra um pacote numa porta este seja retransmitido em todas as portas configuradas com o mecanismo de *hub*. Para isso criou-se um vector de 0's e 1's na FPGA guardada numa memória, em que o endereço é o número da porta e todos os 1's representam as portas a que se deve retransmitir a mensagem sempre que entre um pacote nessa porta.

Para se fazer uma descrição genérica do que foi implementado é preciso ainda fazer uma referência aos menus que fazem parte desta implementação.

O *Software* api_sw desenvolvido inclui um menu que permite a um utilizador comum programar a FPGA consoante as suas necessidades de criação de uma rede de *switching*, podendo este usufruir de toda a capacidade que um *switch* comum oferece. De referir que se programaram os mecanismos de *switch* com capacidade de aging nas portas, bem como a capacidade deste criar redes virtuais (VLANs) nas suas portas, com diferentes identificadores de VLANs abrangendo a gama completa de IDs que o protocolo de VLANs aceita ou seja, de 1 a 4095.

Para que seja um *Software* completo é necessário que o utilizador possa saber como a FPGA está configurada a cada momento, tendo que possuir algumas funções de impressão no monitor do

estado actual das portas. É necessário saber quais são aquelas que estão configuradas com o mecanismo de *switch*, qual o aging time das portas, bem como qual a configuração actual dos identificadores de redes virtuais quando o protocolo de FPGAs está activo.

Este *Software* api_sw também deve ser de simples compreensão quando se pretende associar um mecanismo a uma porta, configurar ou actualizar o aging time de uma ou de todas as portas, associar o mecanismo de VLANs à FPGA, deve ainda ser expedito na configuração da associciação a um VLAN ID. Introduziu-se um novo conceito nesta implementação da API deste *switch* que é o de se poder associar o mecanismo de VLANs com a configuração dos identificadores de redes que já tenham sido previamente configuradas.

3.2.1 Menu do *Software api_sw*

Esta secção descreve as opções que estão disponíveis no menu e qual é a funcionalidade de cada uma para que se possa fazer um usufruto completo das capacidades do *switch* e de todos os protocolos relacionados (neste caso apenas as VLANs).

O menu é constituído por 10 opções, sendo que ao escolher a opção VLANs, o utilizador tem acesso a um submenu que lhe apresenta mais 7 opções que permitem ao utilizador de forma simples e possuindo apenas o conhecimento de redes, configurar correctamente a simulação da FPGA.

A tabela seguinte apresenta todas as opções que o menu principal do *Software* api_sw desenvolvido inclui.

Opção	Breve descrição
Imprimir mecanismos das portas	Imprime os mecanismos associados a cada uma das portas.
Configurar mecanismo de uma porta	Associa um mecanismo a uma porta, sendo que ao inserir o valor 0 o utilizador desconfigura uma porta, se inserir o valor 1 o utilizador configura esta como *hub* e se inserir o valor 2 configura esta como *switch*.
Desconfigurar todas as portas	Basicamente esta função para além de desconfigurar todas as portas, faz um reset ao *switch*.
Imprimir o *agetime* das portas	Imprime o *agetime* actual das portas, bem como o *agetime* anteriormente configurado
Configurar o aging time de todas as portas	As portas são inicializadas com um *agetime* configurado por omissão de 300 segundos. Esta função altera o valor do *agetime* de todas as portas para os segundos que o utilizador pretender.
Configurar o aging time de uma porta	Esta função altera o *agetime* de uma porta.
Actualizar o aging time de todas portas	Esta função altera o *agetime* de todas as portas considerando o tempo que já foi descontado na anterior configuração de *agetime* das portas.
Actualizar o aging time de uma porta	Esta função altera o *agetime* de uma porta considerando o tempo tempo que já foi descontado na anterior configuração de *agetime* da porta.
VLANs	Faz o utilizador aceder ao menu respeitante a VLANs.
Sair do programa de configuração	Encerra o *Software*.

Tabela 3.A: Opções presentes no *Software* desenvolvido para programar a FPGA como *switch*

Resta então identificar e explicar as opções que o *Software* api_sw desenvolvido proporciona para a configuração de VLANs. A tabela abaixo faz referência às opções disponíveis neste submenu, fazendo uma breve descrição de cada uma.

Opção	Breve descrição
Associar o mecanismo de VLANs ao *switch*	Configura a FPGA de modo a que esta funcione com VLANs.
Associar VLANs à *brigde* com a configuração anterior	Associa o mecanismo de VLANs e configura-o com as VLAN IDs da anterior configuração.
Retirar o mecanismo de VLANs sem guardar a configuração	Desconfigura o mecanismo anterior, não guardando a configuração anterior.
Retirar o mecanismo de VLANs e guardar a configuração actual	Desconfigura o mecanismo anterior, guardando a configuração anterior, podendo esta ser utilizada numa próxima configuração.
Associar uma porta a uma VLAN ID	Configura uma porta do *switch* com um VLAN ID.
Imprimir VLANs associada a cada porta	Imprime as portas e seus respectivos identificadores de redes virtuais.
Voltar ao menu anterior	Opção que faz voltar ao menu anterior.

Tabela 3.B: Opções presentes no submenu de VLANs do *Software* desenvolvido

Através dos menus descritos, é possível programar a FPGA com os mecanismos de *switching* de uma forma simples e rápida.

No capítulo seguinte será explicado em detalhe tudo o que foi implementado, inclusive as rotinas que se ocupam de programar a FPGA da forma que pretendida. É de referir que a linguagem de programação utilizada para a programação de rotinas foi o C e um dos conceitos mais utilizados para fazer a troca de dados entre processos foi a de memória partilhada.

4 Descrição detalhada da implementação

4.1 Conceito de memória partilhada em programação

A partilha de uma região de memória entre dois ou mais processos corresponde a maneira mais rápida de se poder efectuar uma troca de dados entre eles. O segmento de memória partilhado é utilizado por cada um dos processos como se fosse um espaço de endereçamento pertencente a cada um dos processos a correr simultaneamente. A partilha de memória possibilita aos vários processos trabalhar sob um espaço de endereçamento comum em memória virtual. Há que notar que este mecanismo de partilha depende da forma de gestão de memória, pelo que significa que as funcionalidades de intercomunicação entre processos dependem do tipo de arquitectura sobre a qual a implementação é efectuada.

Todas as funções de partilha de memória entre processos encontram-se nas bibliotecas **sys/shm.h** e **sys/ipc.h**, sendo necessária a inclusão desta biblioteca em qualquer ficheiro de programação que se pretenda utilizar partilha de dados entre processos.

A função **shmget()** presente na biblioteca **sys/shm.h** permite criar um segmento de memória partilhada e todas as estruturas para o controlo da mesma. Os processos ao criar uma memória partilhada devem definir as permissões de acesso ao segmento de memória, o tamanho de *bytes* desse segmento e a possibilidade de especificar que a forma de acesso de um processo ao segmento será apenas no modo de leitura. O identificador que permite a um processo poder ler e escrever na zona de memória reservada é denominado por **shmid**, sendo este fornecido pelo sistema durante a chamada da função **shmget()** para todo o processo.

A criação de um segmento de memória partilhada, implica duas operações que podem ser executadas por um processo:

- A função **shmat()** faz o acoplamento de um processo à memória partilhada;
- A função **shmdt()** faz o desacoplamento de um processo à memória partilhada.

Quando um processo executa a função **shmat()**, um ponteiro aponta para o inicio da zona de memória partilhada que é permitido ao processo usar. Todos os ponteiros de leitura e escrita apontam para essa zona.

Quando se efectua o desacoplamento de um processo por intermédio da função **shmdt()**, o processo perde a possibilidade de ler ou escrever no segmento de memória partilhada.

No caso particular deste trabalho, utilizou-se uma memória partilhada com o tamanho de 2048 *bytes*, acoplada a um segmento de memória identificado por **shmid** e com uma **KEY** de valor 123.

A função **shmget(key_t key, int size, int shmflg)** ocupa-se da criação do segmento de memória que se pretende criar. Os campos size é preenchido com uma variável inteira de valor 2048, e as flags que se incluiu no campo int shmflg, estão presentes na biblioteca **sys/ipc.h**. Como se pretende uma memória partilhada com direito de leitura e escrita neste campo devem constar as *flags* **SHM_R (0400)** e **SHM_W (200)**. A constante **IPC_CREAT** assegura a criação do segmento se este ainda não existir e a constante **IPC_EXCL** assegura um retorna de falha sempre que se tente criar um segmento com uma chave semelhante.

O argumento **KEY** pode conter dois valores distintos, se for 0 indica que a zona de memória não tem **KEY**, e somente o processo proprietário tem acesso ao segmento de zona partilhada. Mas

neste caso, usou-se uma **KEY** criada pela função **ftok()** que se baseia no campo **path** que se pôs e na chave definida.

É de referir também que a função **shmget()** retorna um inteiro, pelo que se retornar -1 indica que houve um erro ao criar o segmento de memória.

Como só nos interessa acoplar processos ao segmento de memória só será feita referência aqui à forma como se acoplou cada um dos processos integrantes da implementação da API referente a esta dissertação.

Para além de se efectuar a chamada a função **shmget()** que foi utilizada para criar o segmento de memória, cada processo deverá chamar a função:

void *shmat(int shmid, const void *shmaddr,int shmflg)

Esta função é responsável pelo acoplamento do processo ao segmento de memória partilhada com o identificador **shmid**. De notar que todos os processos devem possuir o mesmo **shmid**. Como interessa acoplar o segmento todo em todos os processos então o campo **shmaddr** é 0, caso contrário ter-se-ia que utilizar outro valor.

Por fim é necessário referir que se pretende acoplar uma estrutura de dados **(struct)** a este segmento de dados. Como esta função aponta para o endereço 0 do segmento de memória é necessário criar um ponteiro do tipo void para igualar a esta função.

É também inicializada uma variável antes da introdução de código do tipo **struct swstruct *mem**, já que se pretende que o segmento de memória aponte para uma estrutura de dados.

No código é entrão realizada as seguintes operações:

shmPtr=shmat(shmid,0,0);

mem=(struct swstruct*)shmPtr.

A estrutura de dados **swstruct** encontra-se definida no ficheiro **"estruturas.h"** do qual se fará uma descrição completa mais adiante.

4.2 API implementada para a programação dos mecanismos de *Hub* e *Switch*

4.2.1 Estrutura de dados da memória partilhada

Da memória partilhada definida constam as estruturas responsáveis pelas variáveis partilhadas entre a API que configura a FPGA como *hub* e o *dummy* da FPGA para funcionar como *hub* (**hubstruct**) e as variáveis partilhadas entre a API que configura a FPGA como *switch* e o *dummy* da FPGA para funcionar como *switch* (**swstruct**).

Fazem parte da estrutura de dados da memória partilhada entre processos de configuração de *hubbing* (**hubstruct**) as seguintes variáveis:

- **Conf*Hub***, que é um *unsigned short* que pode assumir os valores "**0**" e "**1**" e que se ocupa de informar a FPGA que uma mensagem de configuração de *hubbing* foi enviada pela API. Esta variável quando está a "**1**" serve para indicar a FPGA que uma porta foi configurada ou desconfigurada como *hub*;
- **TabelaACT**: como a FPGA tem que ter sempre duas memórias em que uma apresenta a configuração actual das portas e outra que é programável quando há uma mudança na configuração das portas, o *unsigned short* **tabelaACT** encarrega-se de indicar na mensagem de configuração da FPGA, o endereço de qual memória estará activa na próxima configuração das portas. Desta forma evita-se perca de informação, já que antes de se trocar de memória uma continua com a configuração anterior, só depois quando se acabar de configurar a outra memória é que se efectua a troca de endereços, indicando que é esta nova memória que contem a nova configuração;
- **regin**: é um *unsigned int* que indica na mensagem de configuração enviada para a FPGA, o endereço de memória a ser configurado. De referir também que o número atribuído a regin equivale à porta que vai ser configurada com um vector de "**0's**" e "**1's**". Sendo que todos os "**1's**" ao serem filtrados indicam o número das portas de destino quando chega alguma mensagem à porta **regin**.
- **portain**: é um vector de *unsigned shorts* com a dimensão do número portas presentes na FPGA. Este vector é preenchido com "**0's**" e "**1's**", sendo que todos os "**1's**" dizem respeito à porta **regin** que está a ser configurada, e indicam a quais portas devem ser reencaminhada a mensagem que chega a porta **regin**. De notar que este vector deverá apresentar um "**0**" na posição de **regin** já que a mensagem que chega a esta porta não pode ser reencaminhada para a mesma porta;

Fazem parte da estrutura de dados da memória partilhada entre processos de configuração de *switching* (**swstruct**) as seguintes variáveis:

- **ConfSw**: esta variável *unsigned short* pode tomar os valores "**0**" e "**1**", sempre que se pretenda configurar a tabela de mecanismos da FPGA, associando o mecanismo de *switching* a uma porta, basicamente é colocar um "**1**" no endereço de memória correspondente a porta que se deseja configurar com mecanismo de *switch*. Obviamente, sempre que seja enviada uma mensagem de configuração com este *short* a "**1**", terá que vir associado um vector (**mecsw**) com um "**1**" na posição do vector correspondente a porta que se deseja configurar como *switch*;
- **mecsw**: é um vector unidimensional com o tamanho do número de portas que é possível configurar no *switch*, sendo que a posição x do vector é colocada sempre a "**1**" sempre que se associa à **porta x** o mecanismo de *switch*. Desta forma, quando a memória da FPGA for reconfigurada por

intermédio da recepção de uma mensagem de configuração proveniente da API, já coloca na tabela **(memória) x** um vector com um "**1**", indicando que essa porta tem associado o mecanismo de *switch*.

- **confage**: é colocado a "**1**" sempre que se pretenda configurar ou actualizar o *aging time* das portas configuradas com o mecanismo de *switch*;
- ***agetime***: vector unidimensional de *longs* da dimensão do número de portas do *switch*, contendo o valor de *aging time* actual de cada uma das portas que já se encontram com MAC configurado;
- ***agetime*_old**: vector unidimensional de *longs* da dimensão do número de portas do *switch* que contém o valor anteriormente programado para *aging time*, para que quando se faça actualização de *aging time* para um tempo, o tempo que já tinha sido descontado seja contabilizado no novo *aging time* configurado.
- **confMAC**: é um *unsigned short* que estará a "**1**" quando é feita a aprendizagem de um endereço MAC da FPGA. A FPGA fica encarregue de enviar uma mensagem com destino a API com todos os MACs aprendidos para que esta cada vez que detecte um MAC diferente de 0 comece a descontar o *aging time* até "**0**" sempre que não seja recebido nenhum pacote nessa porta;
- MAC é um vector unidimensional de *longs*, que inclui todos os macs que foram associados às portas, sendo que o índice de cada posição deste vector corresponde ao MAC aprendido pelo número da porta do índice. Se houver mais do que um MAC aprendido por uma porta, então o MAC é guardado na posição correspondente ao número da porta somado a "**n**" vezes o número de portas que constituem o *switch*;
- **confaf** é um *unsigned short* que toma os valores "**0**" e "**1**". Quando é verificado pela API que o *aging time* de um MAC expirou o seu tempo, ou seja, que não foi recebido entretanto outro pacote com essa origem ou destino, então este *bit* é colocado a "**1**" e é enviada uma mensagem para a FPGA com o vector MAC nessa posição a "**0**", que indica que a porta não está associada a nenhum MAC;
- **VLANconf** é um *unsigned short* que sempre que haja qualquer alteração a efectuas na configuração de VLANs a nível da FPGA, é posto a "**1**" numa mensagem de configuração enviada pela API. Estas alterações envolvem as tarefas de associar o mecanismo de VLANs á FPGA, bem como associar uma porta a determinada **VLAN_ID**;
- **VLAN_def** é um *unsigned short* que pode tomar dois valores: "**0**" ou "**1**". Sempre que é colocado a "**1**" numa mensagem de configuração proveniente da API, serve para informar a FPGA que o mecanismo de VLANs deverá ser activado ou desactivado, dependendo do conteúdo do **short VALNst**, colocando um *bit* indicador de mecanismo de VLANs activo (*flag*) em memória da FPGA a "**1**" ou a "**0**" se for para estar desactivado. A tabela de VLANs (memória na FPGA) assume agora um papel importante na selecção de mensagens originárias das portas;
- **VLANst** é um *unsigned short* que pode tomar os valores de "**0**" ou "**1**", mediante se pretenda activar ou desactivar os mecanismos de VLANs na FPGA;
- **VLAN_mec** é um *unsigned short* que é colocado a "**1**" numa mensagem de configuração sempre que for enviada para a FPGA uma lista de identificadores de VLANs correspondentes às portas (**VLANlist**), indicando que a memória reservada para esta lista deve ser configurada com os campos referentes a **VLANlist**;
- **VLANlist** é um vector unidimensional da dimensão do número de portas presentes no *switch* e contém a informação relativa à **VLAN_ID** a que cada porta está associada. Esta é sempre usada

para configurar a tabela com os identificadores de VLANs associadas às portas, correspondendo o índice do vector ao número da porta do *switch*.

É através desta estrutura de dados que é possível fazer a comunicação entre os processos responsáveis pela API, pelo *Aging Time* e pela *dummy* da FPGA.

4.2.2 Mecanismo simulado para programar um *hub* numa FPGA

Considerando que temos um *switch* de 8 portas, apenas queremos que as portas 1,3,4 e 6 funcionem como *Hub*. Para isso temos que ter do lado do processador uma tabela que associa cada uma das portas a um mecanismo, sendo que atribui os seguintes valores:

- 0x0000 – não tem nenhum mecanismo atribuído;
- 0x0001 – mecanismo de *Hub* associado;
- 0x0002 – mecanismo de *Switch* associado.

Pelo exemplo apresentado ter-se-á a seguinte tabela:

Porta	MEC
1	0x0001
2	0x0000
3	0x0001
4	0x0001
5	0x0000
6	0x0001
7	0x0000
8	0x0000

Tabela 4.A: Tabela de mecanismos quando apenas associa-se *Hub*

Para se executar a tarefa de *Hub*, ter-se-á do lado da FPGA uma memória de registos que associa cada uma das portas (**regin**) às portas possíveis de envio (**portain**).

Logo será uma tabela deste tipo:

regin	Portain (P8..P1)
1	00101100
2	00000000
3	00101001
4	00100101
5	00000000
6	00001101
7	00000000
8	00000000

Tabela 4.B: Tabela após ter-se configurado as portas 1, 3, 4 e 6 com o mecanismo de *Hub*

Pelo verificado apenas as portas associadas ao mecanismo de *Hub* têm o registo **portain** configurado, também verifica-se que a própria porta não apresenta o valor "**1**", já que o mecanismo de *Hub*, só faz *flood* às portas configuradas como *Hub* exceptuando ela própria.

O pretendido com o registo **portain**, é que dentro da FPGA este seja filtrado, de modo a que quando chega uma mensagem na porta 1, esta é reencaminhada para todas as portas a "**1**" do registo **portain** associado a **regin**=1. Deste modo iriam receber as portas 3,4 e 6, como se pode facilmente verificar na tabela.

É necessário referir que temos que ter presente duas tabelas iguais na FPGA, já que ao efectuar-se qualquer alteração, a informação da tabela usada seja copiada para uma outra tabela, de forma a reflectir a alteração efectuada, e não afectar a actual configuração.

Temos então que ter presentes dois registos que indicam o "endereço" da tabela actual (registo de entrada) e um "endereço" da próxima tabela (registo de saída).

Imaginando que queríamos configurar a porta 7 como *Hub*. Teríamos o seguinte mecanismo de configuração.

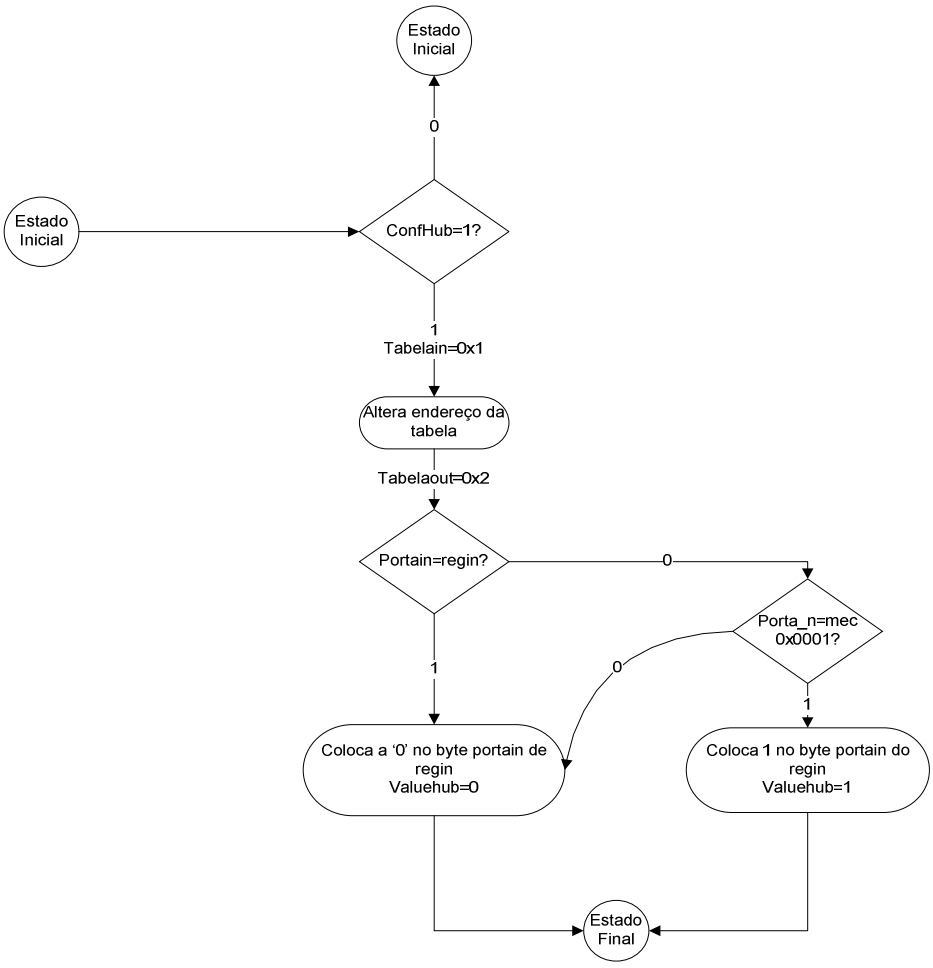

Figura 4.A: Diagrama de fluxo que mostra como a API configura cada posição de memória da FPGA quando o mecanismo da porta é *hub*bing

4.2.3 Mecanismo simulado para programar um *switch* numa FPGA

4.2.3.1 Software desenvolvido de modo a programar a FPGA como switch

Desenvolveu-se e implementou-se um conjunto de funções que constituem uma *Application Programming Interface* (API) permitindo estas interagir com uma FPGA, configurando-a do modo como o utilizador pretende. De referir que a comunicação entre a API e a FPGA é feita por intermédio de *Gigabit Ethernet*, pelo que será adicionado alguns pressupostos da forma como devem ser encapsuladas as mensagens provenientes da API e da FPGA para que a comunicação e a configuração seja feita da forma mais correcta e expedita.

4.2.3.2 API implementada

A API apresenta várias variáveis e vectores globais que devem ser descritos antes de se explicarem as funções que fazem parte constituinte da API. Também, foram definidas três constantes globais:

- **nportas**, representa uma constante que indica qual o índice máximo dos vectores presentes no código da API. Cada vector criado poderá ter nportas entradas cujo o valor de cada índice do vector corresponde ao número da porta;
- **KEY**, representa uma constante que identifica o segmento de memória. Esta constante **KEY** será um dos parâmetros que entrará na função **ftok()** que se encarrega de criar uma chave única do segmento de memória. Se for pretendido uma mesma memória partilhada a diferentes processos esta constante **KEY** deverá ser a mesma.
- **nmacs**, representa uma constante que deverá ser múltipla de **nportas**, e que corresponde ao número de MACs que poderá ser aprendido por esta simulação.

As variáveis globais presentes na implementação da API são as seguintes:

- **mec[nportas]** é um vector unidimensional de *unsigned shorts* e da dimensão do número de portas presentes na FPGA e está encarregue de guardar o mecanismo associado a cada porta para que a API possa informar ao utilizador quais as portas que estão desconfiguradas, associadas a um mecanismo de *hub* ou de *switch*. Faz também a salvaguarda dos mecanismos para que quando haja uma nova configuração do *switch* seja transmitida na trama todos os mecanismos associado a cada porta;
- **list_VLAN[nportas]** é um vector unidimensional de *unsigned ints* e da dimensão do número de portas presentes na FPGA e está encarregue de guardar os VLAN IDs de cada porta, correspondendo cada índice do vector ao número da porta.
- **list_VLANOLD[nportas]** é um vector unidimensional de *unsigned ints* e da dimensão do número de portas presentes na FPGA e está encarregue de guardar os VLAN IDs de uma configuração anterior. Isto vai permitir que ao desassociar o mecanismo de VLANs e voltar a associa-lo com a opção de configurar as portas com os VLAN IDs anteriores seja possível;
- **mecVLAN** é uma variavél do tipo *unsigned short* que assume o valor de "**1**" ou "**0**", mediante o mecanismo de VLANs esteja ou não associado, respectivamente;

- As váriaveis inteiras **shmid** e **size**, a variável de caracteres **path** e o ponteiro **void shmPtr** servem para conseguir criar uma memória partilhada com o tamanho **size** e de identificador **shmid**. Já foi explicado anteriormente como se cria a memória variável em cada processo;

Depois de se fazer referência às variáveis presentes nesta implementação é necessário explicar o que cada função se encarrega de fazer e os procedimentos que encadeia para programar correctamente a *dummy* da FPGA.

Inicialmente é necessário colocar a "**0**" os vectores responsáveis por guardar os mecanismos, os VLAN IDs actuais e os VLAN IDs da anterior configuração. A FPGA deve também garantir que as suas memórias relativas aos mecanismos e VLANs estejam a 0 assegurando que não há falhas na configuração, aquando da inicialização do dispositivo. As funções que têm um papel preponderante na inicialização destes vectores são:

- **void init_mec (unsigned short mecanismo[nportas])** – função que se resume a um "ciclo **for**" que inicializa a "**0**" cada posição do *array* respeitante ao mecanismo;
- **void init_VLANs(unsigned int list_VLAN[nportas],unsigned int list_VLANOLD[nportas])** – função que se resume a um "ciclo **for**" que inicializa a "**0**" cada posição dos *array*s respeitantes aos VLAN IDs actuais e da configuração anterior.

- **void conf_port (unsigned short mecanismo[nportas])** – é o procedimento que se encarrega de associar ou dissociar o mecanismo de *switch* a uma porta e comunicar ao *dummy* da FPGA as alterações nas portas do mecanismo de *switch* .

Esta função pergunta ao utilizador qual a porta que deseja configurar, devendo considerar que só serão configuráveis portas que existam fisicamente na FPGA, ou seja, se temos **nportas** e o vector de mecanismo da API está definido para **nportas**, só deverão ser configuráveis as portas de "**0**" até "**nportas-1**". Abaixo é apresentado o diagrama de fluxo respeitante a esta função tornando mais fácil a compreensão da forma como se processa esta função.

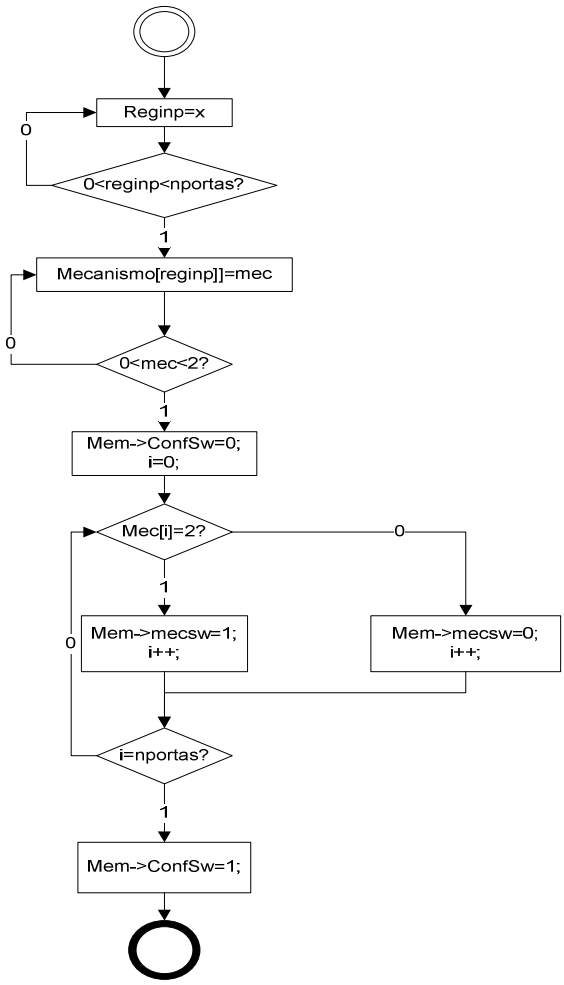

Figura 4.B: Diagrama de fluxo respeitante ao procedimento conf_port

Pela figura é possível verificar que após se ter definido se é pretendido associar ou dissociar uma porta ao mecanismo de *switch*, a função certifica-se que o *bit* (**ConfSw**) que indica uma alteração no *array* de mecanismos está a "**0**", depois copia o *array* de mecanismo para o *array* de mecanismos da memória partilhada. Só quando é efectuada a cópia integral deste *array* é que é colocado a "**1**" o *bit* **ConfSw** da memória partilhada, indicando que o *dummy* da FPGA pode então adquirir a tabela de mecanismos da memória partilhada, copiando para a tabela de mecanismos do *dummy*. Na figura seguinte, verifica-se como é feita a aquisição da tabela de mecanismos por parte do *dummy* da FPGA através de um diagrama de fluxo.

Na prática a comunicação entre a API e a FPGA é feita via *Gigabit Ethernet*, pelo que é necessário encapsular a *flag* **ConfSw** e o vector mecanismo no campo de dados das mensagens de configuração que são enviadas para a FPGA. A figura abaixo indica a forma como se deve encapsular este tipo de mensagens quando se verifica uma mensagem de configuração proveniente da API para a FPGA.

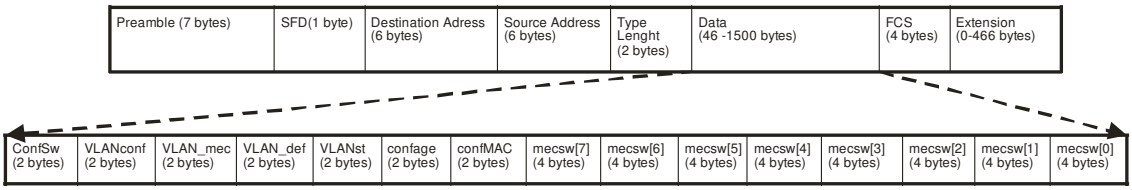

Figura 4.C: Pacote *Gigabit Ethernet* que contem o campo de dados para configurar a FPGA, neste exemplo o caso dos mecanismo

Verifica-se na figura em cima, um exemplo para configurar a tabela de mecanismos de uma FPGA com 8 portas, por intermédio de uma mensagem de configuração proveniente da API. Nos campos **mecsw[7..0]** poderá estar **VLANlist[7..0]** para configurar a tabela de **VLAN IDs** da FPGA.

A FPGA também deverá emitir mensagens deste tipo quando se trata de informar a API que foi recebido um MAC na sua tabela de MACs. Esta mensagem faz com que a API comece a decrementar o *agetime* da porta.

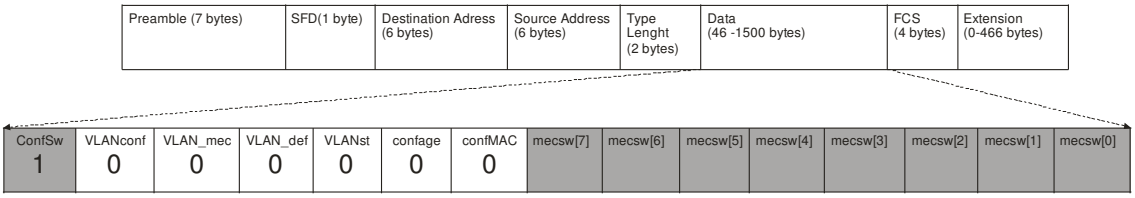

Figura 4.D: Pacote *Gigabit Ethernet* que contem o campo de dados para configurar a FPGA para o caso dos mecanismos

A sombreado, encontram-se os campos relevantes quando se pretende configurar a tabela de mecanismos de uma FPGA. É posto a 1 o campo **ConfSw** e os campos **mecsw[7..0]** apresentam um "**1**" ou "**0**" consoante tenham associado ou não o mecanismo de *switch*, estes são filtrados pela FPGA e ela a detectar o campo **ConfSw** copia para a memória interna os valores presentes nos campos **mecsw[7..0]**.

Abordada que está a escrita na tabela de mecanismos do *dummy* da FPGA e da forma como se poderia fazer na realidade a configuração da memória respeitante aos mecanismos, far-se-á agora uma abordagem das funções implementadas para o *aging time*.

As funções que fazem parte da API e que interagem com o processo relativo ao *aging time* das portas, permitindo configurar o *agetime* das portas são as seguintes:

- **void conf_age_new()** – Função que permite configurar o *agetime* de todas as portas com um valor escolhido pelo utilizador. Esta função força um *agetime* sem contar com o que já foi contabilizado;
- **void confport_age_new** - Função que permite configurar o *agetime* de uma porta com um valor escolhido pelo utilizador. Esta função força um *agetime* sem contar com o que já foi contabilizado;
- **void conf_age()** – Função que configura o *agetime* de todas as portas com o valor escolhido pelo utilizador, a função contabiliza já o tempo que foi descontado da anterior configuração;
- **void confport_age()** – Função que configura o *agetime* de uma porta com o valor escolhido pelo utilizador, a função contabiliza já o tempo que foi descontado da anterior configuração.

Todas estas funções interagem com o processo responsável pelo *aging time* das portas. De referir que o processo de *aging time* é anexo ao processo da API, pelo que a comunicação entre eles faz-se por intermédio de memória partilhada.

Cada uma destas funções activa uma *flag* (**confage**) que indica que se está para configurar o *aging time* de todas as portas ou de uma só porta.

Quando se pretende configurar uma só porta com um *aging time*, a função deverá perguntar ao utilizador que porta deseja configurar e o *aging time* a aplicar nesta. Depois deverá colocar o *aging time* configurado para essa porta no *array* de inteiros ***agetime*[nportas]** da memória partilhada. O *aging time* da porta colocado neste *array* difere consoante se pretenda forçar um valor de *aging time* (**confport_age_new**) ou apenas actualizá-lo (**confport_age**).

- Se uma porta for forçada a adquirir um *aging time*, o valor decrementado da anterior configuração não é contabilizado, pelo que o *aging time* colocado pelo utilizador é posto nos *array*s de inteiros ***agetime*[nportas]** e ***agetime*_old[nportas]**;
- Se uma porta for actualizada com um novo *aging time*, o valor decrementado da anterior configuração é contabilizado. O *aging time* que será ado no *array* ***agetime*[nportas]** é obtido pela soma do *aging time* que o utilizador introduziu com o *aging time* actual subtraindo ainda o *aging time* da configuração anterior presente no *array* de inteiros ***agetime*_old[nportas]** da memória partilhada entre a API e o processo "**agetimeconf**". De notar também, que só após se ter efectuado a operação aritmética anterior para o cálculo do *aging time* "actualizado" é que se poderá introduzir no *array* de inteiros ***agetime*_old[nportas]** o valor que o utilizador introduziu para configurar o *aging time* da porta.

Após colocar o valor correspondente ao *aging time* nos vectores da memória partilhada reservados, o **unsigned short "confage"** da memória partilhada toma o valor de "**1**" para indicar ao processo anexo "**agetimeconf**" que se efectuou actualização na memória partilhada.

O processo realizado pelas funções que forçam ou actualizam o mesmo *aging* em todas as portas (**conf_age()** e **conf_age_new()**, respectivamente) é realizado da mesma forma que para uma porta só que é feito um "ciclo **for**" a todas as portas para forçar ou actualizar o *aging time*.

Nos diagramas de fluxo em baixo pode-se verificar a forma de processamento da função que força e actualiza uma porta, possibilitando uma melhor percepçãodo modo de interacção com o processo "**agetimeconf**".

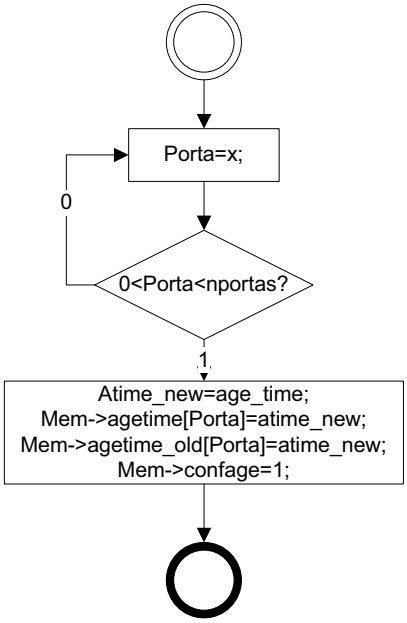

Figura 4.E: Diagrama de fluxo do procedimento confport_age_new

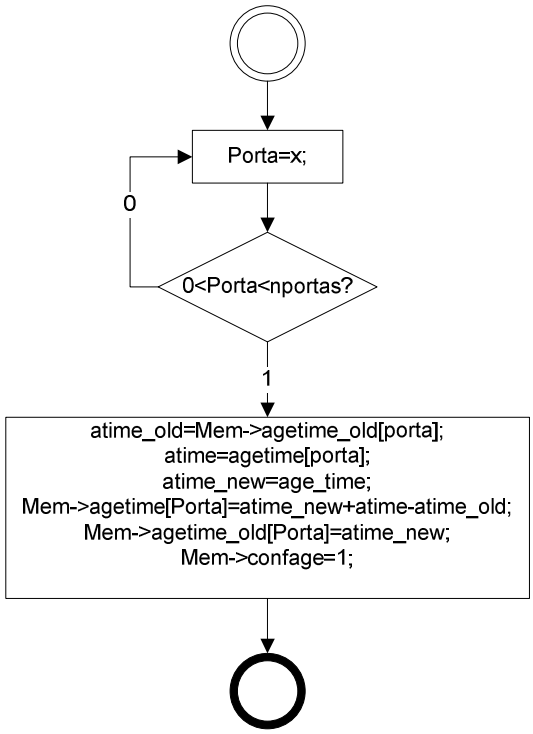

Figura 4.F: Diagrama de fluxo do procedimento confport_age

Do que foi implementado falta apenas mencionar e explicar detalhadamente quais os mecanismos utilizados para activar o mecanismo de VLANs e associar uma VLAN a uma porta do *dummy* da FPGA por parte da API.

As funções da API responsáveis por uma correcta configuração do mecanismo de VLANs são as seguintes:

- **unsigned short VLAN_mech(unsigned int list_VLAN[nportas], unsigned short VLANmec)**

É uma função responsável por definir na API que o mecanismo de VLANs está activo, bem como comunicar ao *dummy* da FPGA que o mecanismo de VLANs está activo, utilizando as *flags* da memória partilhada **VLANconf**, **VLAN_def** e **VLANst**. Deve também transmitir a lista de VLAN IDs ao *dummy* da FPGA com o valor "**0**" em todas os índices do *array* **VLANlist[nportas]** da memória partilhada , para se assegurar que estas inicialmente não se encontram associadas a qualquer VLAN. É de referir também que a função retorna um *unsigned short* a "**1**" para que a variável global responsável por informar a API qual o estado do mecanismo de VLANs esteja activo;

- **void portVLAN(unsigned int list_VLAN[nportas], unsigned short mec[nportas],unsigned short VLANmec)**

É um procedimento responsável por associar uma VLAN a uma porta, colocando o valor do VLAN ID num vector unidimensional que contém todos os VLAN IDs das portas, sendo o índice deste *array* o número da porta a configurar. Cada posição tem o valor "**0**" quando não tem VLAN IDs associados ou um valor inteiro entre "**1**" e "**4095**" (12 *bits* do campo do protocolo de VLANS). Os outros parâmetros servem para assegurar que quando se configura uma porta, esta está associada ao mecanismo de *switch* (*array* mec[nportas]) e que o mecanismo de VLANs encontra-se activo. Ao não verificar qualquer uma destas condições, a porta não deverá ser configurada. Pelo contrário se for verificando que estas condições são respeitadas, então o procedimento fica encarregue de comunicar com o *dummy* da FPGA, activando as flags unsigned shorts VLANconf e VLAN_mec da memória partilhada entre os processos e enviando para o *dummy* os valores correspondentes a cada VLAN ID de cada porta pelo vector VLANlist[nportas] da memória partilhada. O *dummy* ao receber uma mensagem deste tipo deve copiar a lista das VLANs para o *array* interno ao seu processo que guarda as VLAN IDs. Na prática numa FPGA, as VLAN IDs seriam guardadas na memória sendo que cada endereço de memória correspondia a uma porta presente no *switch*;

- **unsigned short VLAN_mech0(unsigned int list_VLAN[nportas], unsigned int list_VLANOLD[nportas], unsigned short VLANmec)**

É uma função responsável por dissociar o mecanismo de VLANs do *dummy* da FPGA sem guardar a configuração dos VLAN IDs das portas. Encarrega-se também de retornar um *unsigned short* para que se possa configurar a variável global da API que define o estado inactivo do mecanismo de VLANs. Esta função deve retornar um *short* a "**0**" e comunicar com o *dummy* da FPGA por intermédio dos *unsigned shorts* **VLANconf**, **VLAN_def** e **VLANst** que é para desactivar o mecanismo de VLANs no processo do *dummy*. Na prática seria enviada uma mensagem de configuração para a FPGA para que esta ao filtrar os campos da mensagem torne inactivo o mecanismo de VLANs;

- **unsigned short VLAN_mech0_OLD (unsigned short mec[nportas], unsigned int list_VLAN[nportas], unsigned int list_VLANOLD[nportas], unsigned short VLANmec)**

É uma função responsável por dissociar o mecanismo de VLANs do *dummy* da FPGA guardando a configuração dos VLAN IDs das portas. Encarrega-se também de retornar um *unsigned short* para que se possa configurar a variável global da API que define o estado inactivo do mecanismo de VLANs. Deve guardar no *array* **list_VLANOLD[nportas]** a actual configuração dos VLAN IDs para que mais tarde possibilite activar o mecanismo e associar os VLAN IDs previamente configurados ao *dummy* da FPGA. Para isso é copiado por intermédio de um "ciclo **for**" a informação que está no *array* **list_VLAN** para o *array* **list_VLANOLD**. Esta função deve retornar um short a 0 e comunicar com o *dummy* da FPGA por intermédio dos *unsigned shorts* **VLANconf**, **VLAN_def** e **VLANst** que é para desactivar o mecanismo de VLANs no processo do *dummy*. Na prática seria enviada uma mensagem de configuração para a FPGA para que esta ao filtrar os campos da mensagem torne inactivo o mecanismo de VLANs;

- **unsigned short VLAN_mech1_OLD (unsigned short mec[nportas], unsigned int list_VLAN[nportas], unsigned int list_VLANOLD[nportas], unsigned short VLANmec)**

É uma função responsável por associar o mecanismo de VLANs do *dummy* da FPGA, configurando as portas com os VLAN IDs de uma situação anterior. Encarrega-se também de retornar um *unsigned short* para que se possa configurar a variável global da API que define o estado activo do mecanismo de VLANs

Esta função deve retornar um short a 0 e comunicar com o *dummy* da FPGA por intermédio dos *unsigned shorts* **VLANconf**, **VLAN_def** e **VLANst** que é para activar o mecanismo de VLANs no processo do *dummy*. Deve guardar no *array* **list_VLAN[nportas]** a informação contida no *array* **list_VLANOLD[nportas]** e colocar no *array* **VLANlist[nportas]** da memória partilhada entre processos essa informação também, para que quando seja activada a flag **VLAN_mec** a informação seja transmitada para o *dummy* da FPGA e processo responsável por esta adquira para o seu *array* dedicado aos VLAN IDs. Na prática seria enviada uma mensagem de configuração para a FPGA para que esta ao filtrar os campos da mensagem torne activo o mecanismo de VLANs e copie para a memória relativa aos VLAN IDS os campos respeitantes ao vector **VLANlist[nportas]**;

Acabada esta descrição minuciosa de todas as funções que fazem parte da configuração do mecanismo de VLANs, vai se apresentar o digrama de fluxo respeitante a cada uma das funções programadas, bem como o encapsulamento das mensagens *Gigabit Ethernet* que podem efectuar a transmissão de dados entre a API e a FPGA.

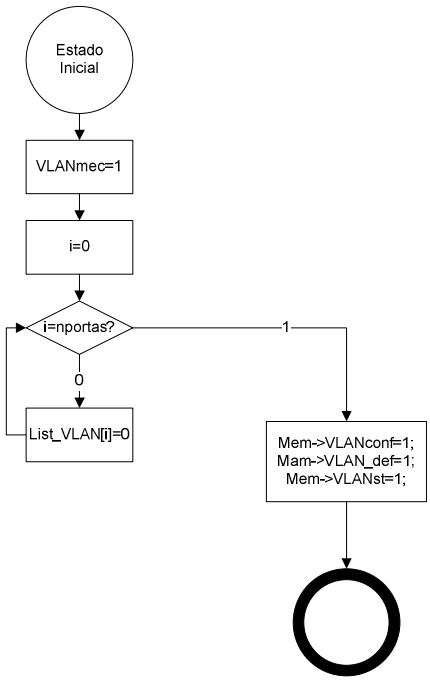

Figura 4.G: Diagrama de fluxo relacionado com o procedimento VLAN_mech

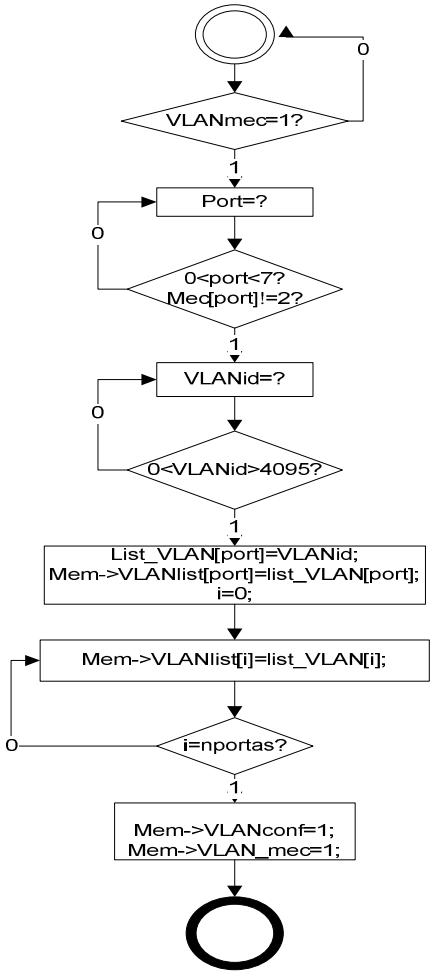

Figura 4.H: Diagrama de fluxo relacionado com o procedimento portVLAN

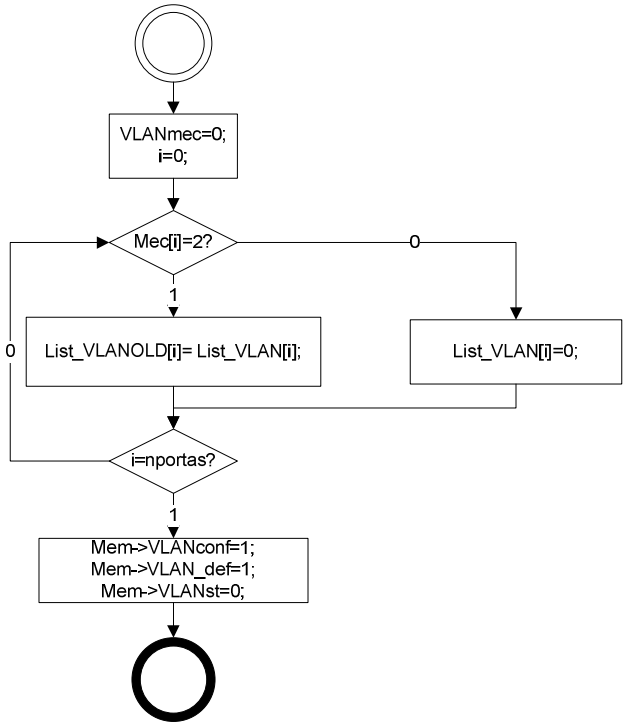

Figura 4.I: Diagrama de fluxo relacionado com o procedimento VLAN_mech0_old

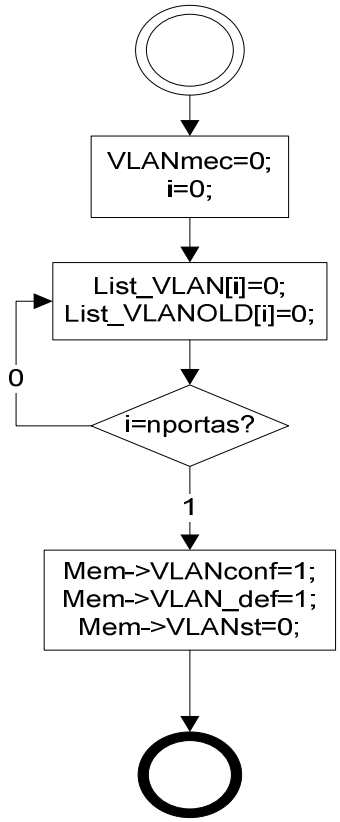

Figura 4.J: Diagrama de fluxo relacionado com o procedimento VLAN_mech0

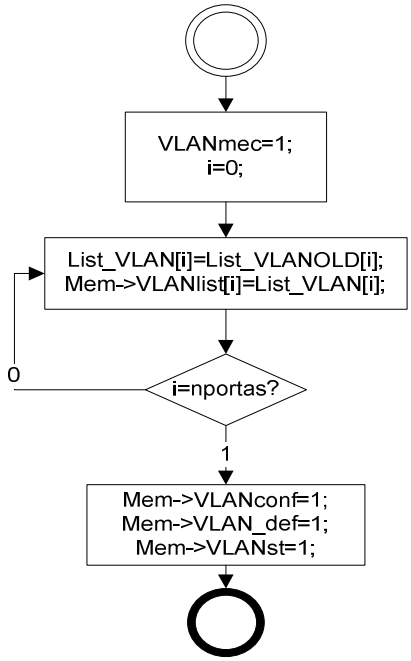

Figura 4.K: Diagrama de fluxo relacionado com o procedimento VLAN_mech1_OLD

O encapsulamento *Gigabit Ethernet* de mensagens de configuração de VLANs provenientes da API e com destino a FPGA deve ser feito da forma como as imagens abaixo ilustram.

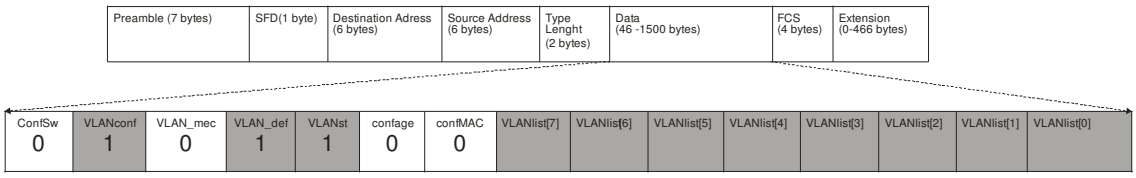

Figura 4.L: Pacote *Gigabit Ethernet* que contem os campos necessários para que se active a FPGA com o mecanismo de VLANs (função VLAN_mech)

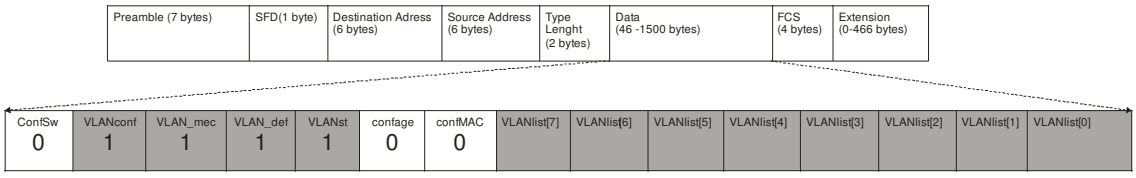

Figura 4.M: Pacote *Gigabit Ethernet* que contem os campos necessários para que se active o mecanismo de VLANs e se configure a memória da FPGA relativa as VLAN IDs seja configurada com os valores de VLANlist[7..0]

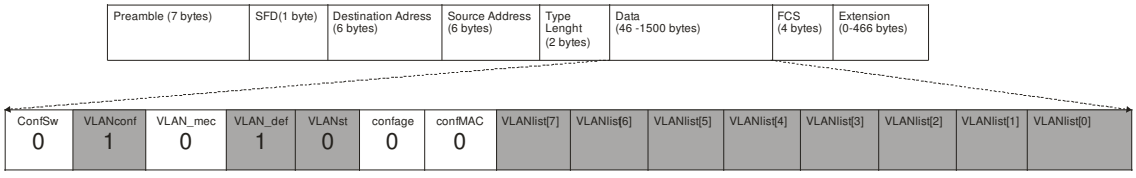

Figura 4.N: Pacote *Gigabit Ethernet* que contem os campos necessários para que se desactive o mecanismo de VLANs (função VLAN_mech0 e VLAN_mech0_OLD)

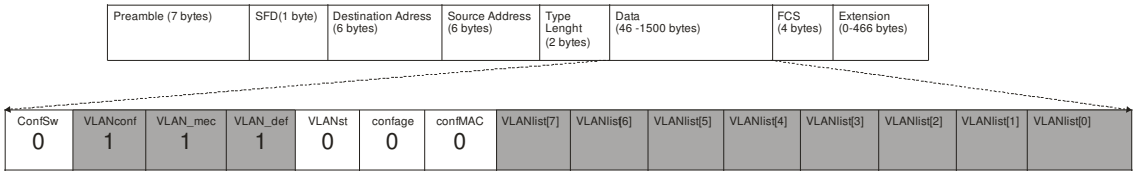

Figura 4.O: Pacote *Gigabit Ethernet* que contem os campos necessários para que se configure a memória da FPGA relativa as VLAN IDs com os valores de VLANlist[7..0]

Nas figuras acima todos os campos a sombreado são necessários para se fazer uma correcta comunicação entre a API e a FPGA, embora deva-se referir que quando se pretende desconfigurar o mecanismo de VLANs, os campos a sombreado de **VLANlist[7..0]** devem ser postos a "**0**", já que o mínimo de *bytes* que pode-se transmitir na mensagem de dados são 48 *bytes*. Isto não implica uma nova configuração de VLAN IDs na memória da FPGA com tudo a "**0**", já que nessas mensagens de configuração o campo **VLAN_mec** que se encarrega de avisar a FPGA que deve escrever os valores dos campos **VLANlist[7..0]** é colocado a "**0**".

*4.2.3.3 Processo de configuração do aging time das portas ("***agetimeconf***")*

O processo "**agetimeconf**" foi criado para ser um processo apenas dedicado ao tratamento dos *aging times* das portas, interagindo com o processo relativo a API e com a *dummy* da FPGA. Sempre que é detectado alterações num *aging* de uma porta na API ou na tabela de MACs da *dummy* da FPGA ele deverá entrar em acção, copiando para o *array* presente no processo **lista_macs[nportas][2]**, os MACs aprendidos pelas portas do *dummy* da FPGA na primeira coluna deste *array*. A segunda coluna é dedicada somente aos aging times actuais de cada porta contendo o *agetime* actual de cada porta.

Este processo deve ocupar-se de decrementar o valor de *aging times* sempre que o MAC de uma porta for diferente de "**0**", bem como comunicar a FPGA *dummy* que é necessário apagar uma entrada do MAC quando o *agetime* expira.

O processo "**agetimeconf**" é um ciclo **while** infinito que tem dois tipos de procedimentos:

- A condição de um ciclo **while** é verificar se os **unsigned shorts confage** e **confMAC** apresentam-se a "**0**". Se apresentarem-se a "**0**" é porque não houve alteração tanto na tabela de *agetime* das portas, como não foi aprendido nenhum MAC no *dummy* da FPGA. De seguida é feito um "ciclo **for**" a todas as portas para que seja decrementada uma unidade de tempo em cada porta com MAC diferente de "**0**", também é verificado se confage da memória partilhada não apresenta o valor "**1**" para colocar o valor decrementado do aging time da porta, já que é preciso salvaguardar que não foi feita nenhuma nova configuração de *aging time* por parte da API antes de se salvaguardar.

Quando o *agetime* de uma porta chega a "**0**", é necessário colocar nos índices relativos à porta do vector MAC da memória partilhada o valor de "**0**", actualizar o vector bidimensional **lista_macs** sendo colocado na primeira coluna desta o valor "**0**" respeitante ao MAC, e na segunda coluna o valor respeitante ao *agetime*.

É então colocado a "**1**" o *short* **confaf** da memória partilhada, que comunica à *dummy* da FPGA toda a tabela de MACs, já sem o MAC relativo a porta que expirou o aging time.

- Se for verificado pelo ciclo while descrito anteriormente que alguma das duas flags confage ou **confMAC** está a "**1**", este não se executa e salta para duas funções que se ocupam de fazer o

getting dos vectores ***agetime*[nportas]** e/ou **MAC[nmacs]** (**get_ages** e **get_mac**, respectivamente) a memória partilhada para o *array* interno de "**agetimeconf**" **listamacs[nmacs][2]**.

Foi feita referência anteriormente a duas funções: **get_ages(long lista_macs[nportas][2])** e **get_mac(long lista_macs[nportas][2])**. Resta saber o que elas fazem e como se processam.

- A função **void get_ages(long lista_macs[nmacs][2])** é a função responsável por fazer a actualização dos valores de *aging time* no *array* **lista_macs**.

Basicamente se há alguma alteração feita pelo utilizador no *aging time* de uma ou de todas as portas, é posto o sinal confage da memória partilhada a "**1**" do lado do processo da API. A função **get_ages** sempre que executada verifica se este está a "**1**", e se estiver faz uma cópia para a coluna respeitante aos *aging times* do *array* **lista_macs[nmacs][2]**. Feita a cópia, a função ocupa-se de colocar a variável confage a "**0**", para que garanta que o ciclo **while** dedicado a fazer o decremento dos *aging times* das portas com MAC diferente a "**0**" seja executado.

- A função **void get_mac(long lista_macs[nmacs][2])** é a função responsável por fazer a actualização dos MACs provenientes do *dummy* da FPGA no *array* **lista_macs**.

Se for aprendido um novo MAC numa porta, é posto o sinal **confMAC** da memória partilhada a "**1**" do lado do processo da FPGA *dummy*. A função **get_mac** sempre que executada verifica se este está a "**1**", e se estiver faz uma cópia de todos os MACs do *array* **MAC[nmacs]** da memória partilhada para a coluna respeitante aos MACs do *array* **lista_macs[nmacs][2]**. Feita a cópia a função ocupa-se de colocar a variável **confMAC** a "**0**" para que garanta que o ciclo while dedicado a fazer o decremento dos *aging times* das portas com MAC diferente a "**0**" seja executado.

Abaixo mostra-se um diagrama de fluxo de como se processa toda a informação proveniente da memória partilhada no processo "**agetimeconf**".

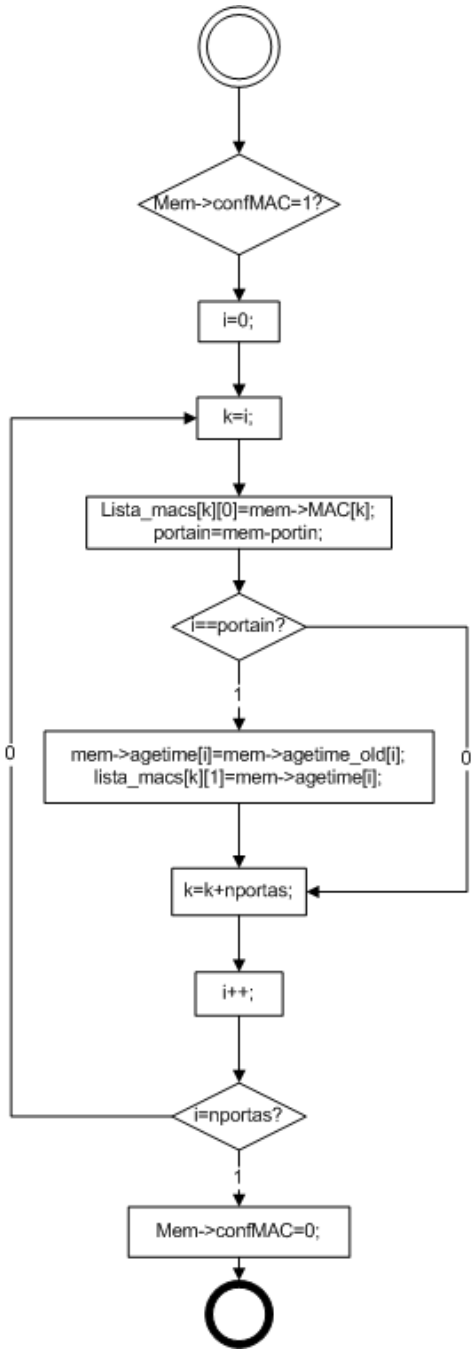

Figura 4.P: Diagrama de fluxo relativo ao procedimento get_mac

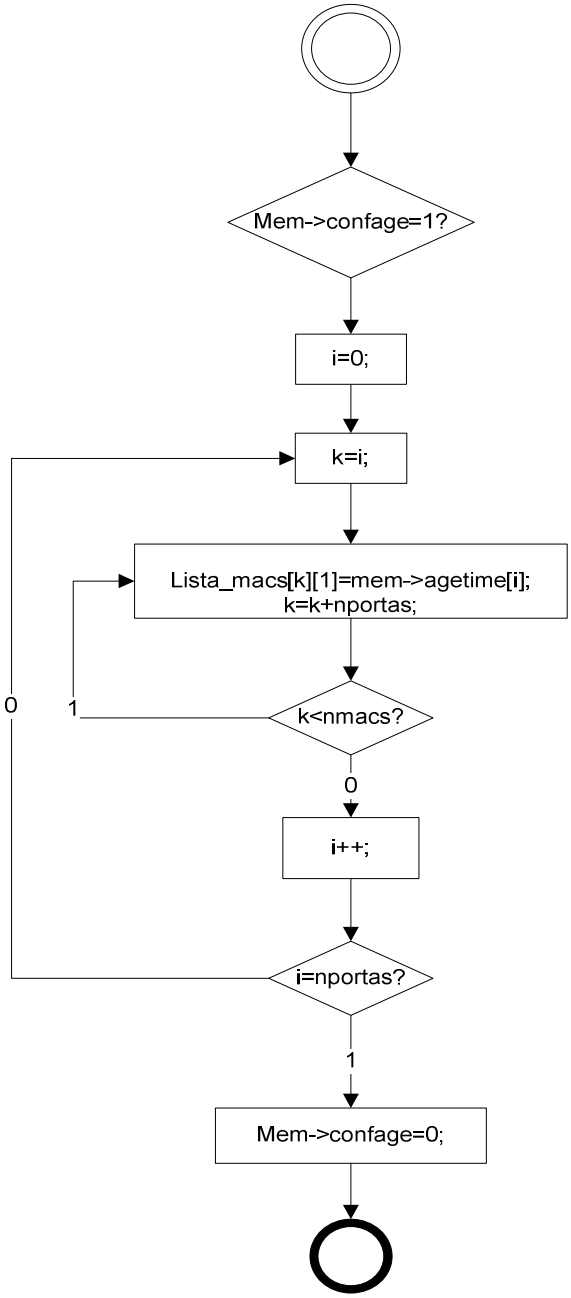

Figura 4.Q: Diagrama de fluxo relativo ao procedimento get_ages

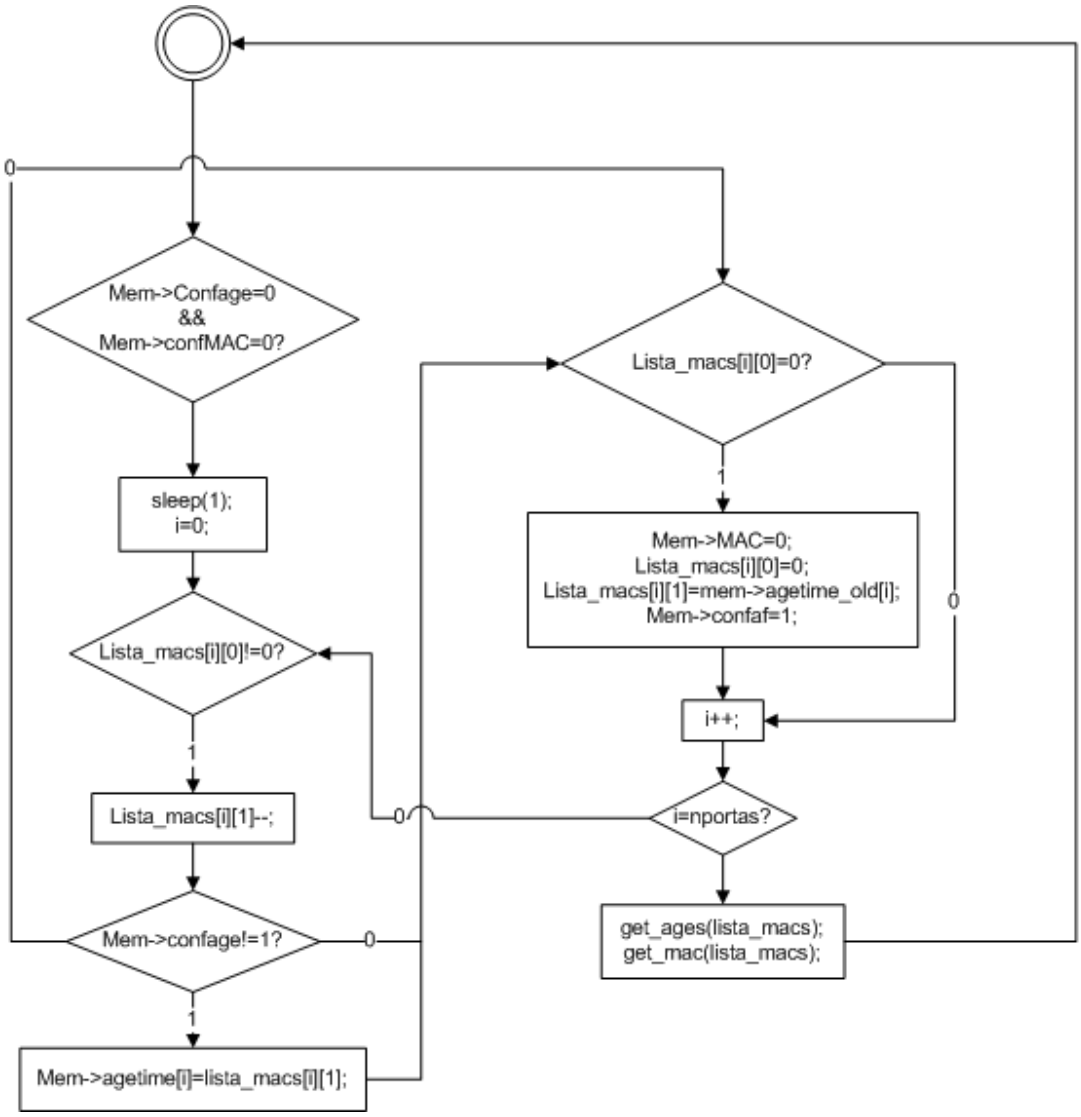

Figura 4.R: Diagrama de fluxo de como se processa o processo "agetimeconf" apenas para uma entrada por porta

4.2.3.4 O dummy da FPGA

A implementação deste *dummy* serve para simular a FPGA e testar a viabilidade dos mecanismos programados na API para a comunicação com esta.

Este *dummy* apresenta 4 variáveis globais de extrema importância que devem passar a ser explicadas seguidamente:

- **mac_list** é um *array* unidimensional de dimensão **nmacs+1**, sendo que deve ser inicializado a **0** todas as portas e a última entrada deve ter um valor estático de "**0xFFFFFF**". Esta entrada serve para quando houver uma mensagem com certo destino, for feita uma busca desse MAC e este não conste na lista, seja activado o mecanismo de *flooding* que envia a mensagem a todas as portas com o mecanismo de *switch* sem enviar para a própria porta;

- **mec_*switch*** é um *array* unidimensional de dimensão nportas, sendo que este deve ser inicializado a 0 em todos os índices, na medida que inicialmente cada porta não deve ter nenhum mecanismo associado, nem de *switch* nem de *hub*;
- **list_VLAN** é um *array* unidimensional de dimensão **nportas**, sendo que deve ser inicializada todas as entradas deste *array* a "**0**", assegurando que nenhuma porta se encontra inicialmente associada a nenhuma VLAN.
- **VLANactivo** é uma variável do tipo **unsigned short**, responsável por guardar o estado activo (valor "**1**") ou de inactividade (valor "**0**") do mecanismo de VLANs.

Os procedimentos que se encarregam de fazer a inicialização dos *array* globais são os seguintes:

init_mec(unsigned long mec_*switch*[nportas])

É um procedimento que faz com que todas as posições do *array* **mec_*switch*[nportas]** sejam inicializadas a "**0**", não tendo nenhum mecanismo associado.

Este procedimento utiliza um "ciclo **for**" de **nportas** iterações para colocar em cada índice correspondente a cada porta o valor "**0**", representando este valor que nenhuma porta encontra qualquer mecanismo associado.

Na prática numa FPGA é criada uma memória de **nportas** endereços dedicada aos mecanismos e em cada posição é escrito o valor "**0**".

- **init_list(unsigned long mac_list[nmacs+1])**

É um procedimento que faz com que todas as posições do *array* **mac_list[nportas+1]** sejam inicializadas a "**0**", não tendo nenhum MAC associado à posição do índice correspondente a cada uma das portas.

Este procedimento utiliza um "ciclo **for**" de **nmacs+1** iterações para colocar em cada um dos índices correspondentes a cada porta o valor "**0**" e no índice **nmacs+1** coloca o valor "**0xFFFFFF**", que serve para indicar que quando é recebida com um certo MAC de destino e este não consta do *array*, é feito o *Broadcast* das mensagens para todas as portas, também assegura a possibilidade de enviar mensagens em *Broadcast*.

Na prática numa FPGA é criada uma memória de **nmacs** endereços dedicada aos MACs e em cada posição é escrito o valor "**0**", sendo que na posição é escrita o valor "**0xFFFFFF**".

De referir que sempre que tivermos **nportas** numa FPGA, os MACs aprendidos e escritos nesta memória deverão estar escritos nas posições $\boldsymbol{n}$ú$\boldsymbol{mero\ da\ porta} + \boldsymbol{n} \times \boldsymbol{nportas}$, em que "**n**" é o número de iterações que desde que se verifique que o MAC de uma posição de memória é diferente de "**0**" faz com que seja lida a posição de memória **nportas** à frente até se encontrar um espaço de memória igual a "**0**", sendo escrito aí o novo MAC aprendido. Isto pode ser implementado por um ciclo **while** cuja condição de salto seja a verificação de um número superior a **nmacs**, e a cada iteração seja feita um incremento de **nportas**.

- **init_VLANlist(unsigned int list_VLAN[nportas])**

É um procedimento que faz com que todas as posições do *array* **list_VLAN[nportas]** sejam inicializadas a "**0**", assegurando que quando é inicializado o *dummy* da FPGA não haja nenhuma posição do *array* com um VLAN ID.

Este procedimento efectua um "ciclo **for**" de **nportas** iterações para colocar em cada índice correspondente a cada porta o valor "**0**".

Na prática programada numa FPGA uma memória de nportas endereços dedicada aos identificadores de rede virtual (VLAN IDs) e em cada posição é escrito o valor "**0**".

Não são apresentados os diagramas de fluxo relativos a estas três funções devido à sua simplicidade.

O *dummy* da FPGA é um ciclo **while** infinito que inicialmente procura na memória partilhada se alguns dos *bits* de configuração estão activos, ou seja, se estes estão a "**1**". Estes *bits* são o **confSw**, **VLANconf** e **confaf**. É necessário haver um procedimento que inicialmente coloque estes *bits* a "**0**" para que não ocorram erros na comunicação, já que o segmento de memória pode ter sido utilizado anteriormente. O procedimento desta *dummy* que se encarrega desta inicialização é o **init_val()**.

É necessário fazer referência agora aos procedimentos e funções que fazem o *getting* da memória partilhada dos *array*s de mecanismso, MACs e VLAN IDs de cada uma das portas. Sempre que um dos *bits* está activo indica que o *dummy* da FPGA terá que efectuar qualquer alteração nos seus *array*s ou na variável que indica o estado de activo ou inactividade do mecanismo de VLANs. Os procedimentos que se encarregam de adquirir a informação da memória partilhada são os seguintes:

- **void get_mec (unsigned long mecanismo[nportas]**

Resume-se a um procedimento que efectua um "ciclo **for**" para copiar os mecanismos do *array* **mecsw[nportas]** da memória partilhada para o *array* **mec_*switch*[nportas]** do processo do *dummy* da FPGA. Cada posição do índice do *array* **mec_*switch*[nportas]** fica com o valor de "**0**" se estiver desconfigurada ou com o valor "**1**" se for configurado o mecanismo de *switch* pelo *Software* api_sw desenvolvido;

Na realidade é enviada uma mensagem de configuração por *Gigabit Ethernet* já mencionada no subcapítulo da API em que o campo relativo á *flag* **ConfSw** está activo e cada mecanismo associado a uma porta faz parte dos campos **mecsw[nportas..0]**. Estes são todos filtrados na FPGA, e esta detectando que **ConfSw** está a "**1**" deverá colocar os valores relativos a **mecsw[nportas..0]**, nos endereços de memória **[nportas..0]** reservados para guardar os mecanismos;

- **void get_mac (unsigned long lista_mac[nmacs]**

Resume-se a um procedimento que efectua um "ciclo **for**" para copiar os MACs do *array* **MAC[nmacs]** da memória partilhada para o *array* **lista_mac[nmacs]** do processo do *dummy* da FPGA. Cada posição do índice do *array* **lista_mac[nmacs]** fica com o valor de "**0**" se não receber nenhum

pacote durante o tempo de *agetime*, ou com o valor do MAC se o valor de *agetime* ainda não tiver expirado;

Na realidade é enviada uma mensagem de configuração por *Gigabit Ethernet* já mencionada no subcapítulo da API em que o campo relativo á *flag* **confaf** está activo e cada mecanismo associado a uma porta faz parte dos campos **MAC[nnacs..0]**. Estes são todos filtrados na FPGA, e esta detectando que **confaf** está a **1** deverá colocar os valores relativos a **MAC[nmacs..0]**, nos endereços de memória **[nportas..0]** reservados para guardar os MACs;

- **unsigned short get_VLANs (unsigned int listaVLAN[nportas], unsigned short VLANmec)**

Resume-se a uma função que efectua um "ciclo **for**" para copiar as VLAN IDs do *array* **VLANlist[nportas]** da memória partilhada para o *array* **list_VLAN[nportas]** do processo do *dummy* da FPGA. Se as variáveis **VLANconf** e **VLAN_def** da memória partilhada estiverem activas (tomam valor **1**), é porque se quer activar ou desactivar o mecanismo de VLANs da FPGA*dummy*, consoante a variável **VLANst** esteja a **1** ou a **0**, respectivamente. Se as variáveis **VLANconf** e **VLAN_mec** da memória partilhada estiverem activas (tomam valor **1**), cada posição do índice do *array* **list_VLAN[nportas]** fica com o valor de **0** se não estiver associado a nenhuma rede virtual, ou com o valor do VLAN ID se esta tiver sido configurada na API. Esta função também está encarregue de associar ou de dissociar o mecanismo de VLANs, pelo que deve retornar um **unsigned short** com o valor "**1**" se o mecanismo for activado ou com o valor "**0**" se o mecanismo for dissociado.

Na realidade podem ser enviadas mensagens de configuração por *Gigabit Ethernet* já mencionadas no subcapítulo da API em que o campos relativos às *flags* **VLANconf**, **VLAN_def** e **VLAN_mec** podem estar ou não activos, mediante o que se queira configurar. Se os *bits* de configuração **VLANconf** e **VLAN_def** estiverem activos estamos perante uma mensagem de configuração em que se quer activar ou desactivar o mecanismo de VLANs, mediante o que esteja presente no *bit* **VLANst**, que está a "**0"** quando se quer desactivar ou a "**1**" quando se quer activar o mecanismo.

Se os *bits* de configuração **VLANconf** e **VLAN_mec** estão activos, estamos perante uma mensagem de configuração em que se pretende configurar a memória relativa aos VLAN IDs. Cada VLAN ID associado a uma porta faz parte dos campos **VLANlist[nportas..0]**. Estes são todos filtrados na FPGA, e esta detectando que **VLANconf** e **VLAN_mec** estão a "**1**" deverá colocar os valores relativos aos campos **VLANlist[nportas..0]**, nos endereços de memória **[nportas..0]** reservados para guardar os VLAN IDs associadas a cada porta.

De seguida serão apresentados os diagramas de fluxo relativos a cada um destes procedimentos e funções que fazem o *getting* dos mecanismos, MACs e VLAN IDs.

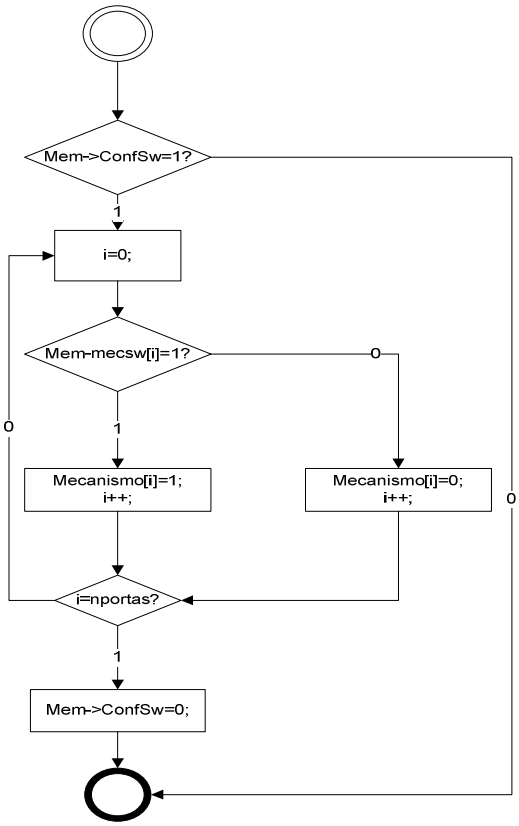

Figura 4.S: Diagrama de fluxo da função get_mec

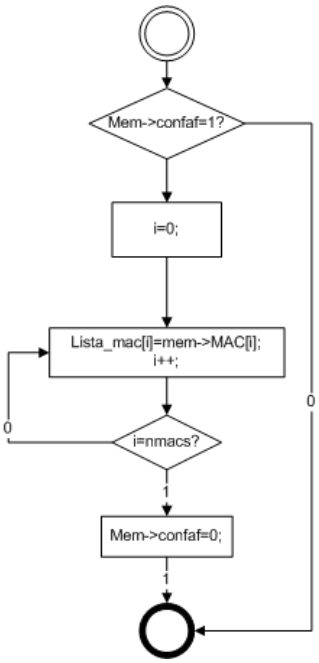

Figura 4.T: Diagrama de fluxo relativo ao prodecimento get_mac

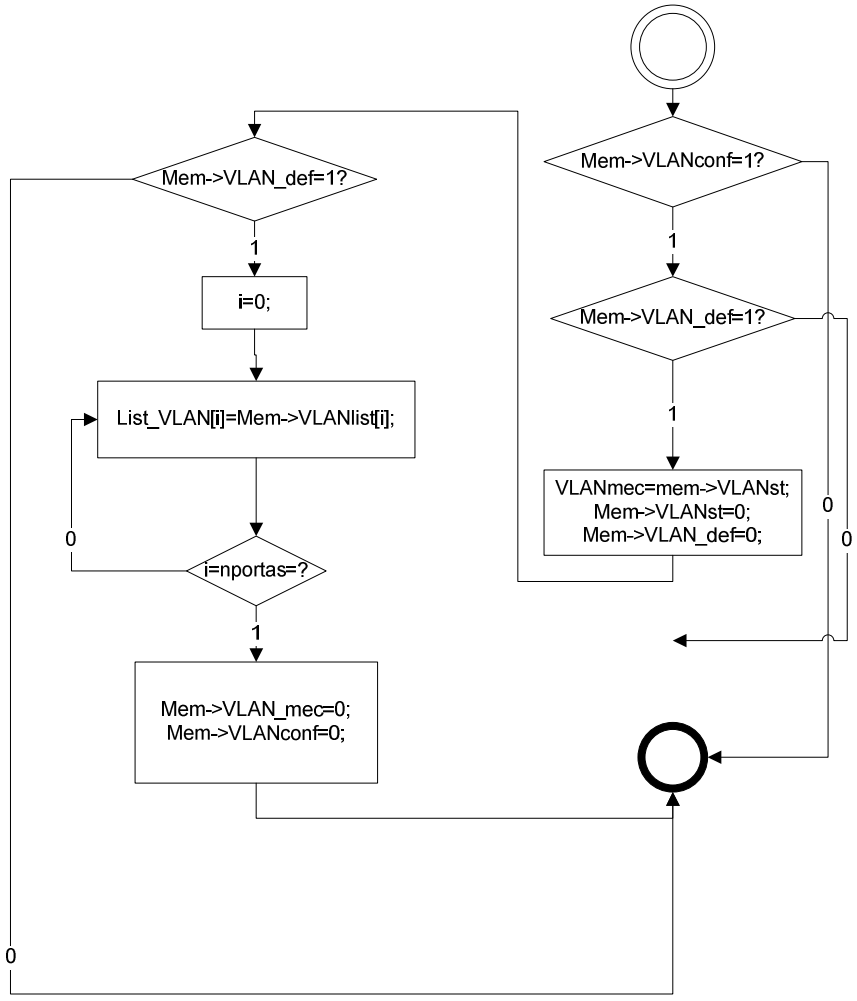

Figura 4.U: Diagrama de fluxo relativo a função get_vlan

Por fim é necessário um procedimento que insira um pacote na *dummy* da FPGA para que se possa testar o envio de mensagens com um MAC origem e um MAC destinatário e se o mecanismo de VLANs estiver activo, terá que ser possível também inserir o VLAN ID para a qual a mensagem deverá ser destinada. De referir que por omissão, um *switch* deve considerar que só se reencaminham as mensagens com o VLAN ID ao qual a porta de origem está associada. Esta *dummy* foi programada tendo em conta que se uma mensagem chegar a uma porta com VLAN ID diferente daquele que se pretende reencaminhar a mensagem, esta é reencaminhada para as portas com o VLAN ID que consta do pacote.

Os procedimentos responsáveis pela inserção de um pacote na *dummy* da FPGA, aprendizagem de MACs para o *array* **list_macs[macs+1]**, reencaminhamento de mensagens e comunicação com a API para que quando seja aprendido um MAC, esta posso começar a decrementar o *agetime* são os seguintes:

- **void insert_packet (unsigned int lista_vlan[nportas+1], unsigned long list_mac[nmacs+1],unsigned long mec[nportas],unsigned short VLANmec)**

Procedimento que permite inserir um pacote na *dummy* da FPGA, perguntando ao utilizador qual a porta de origem desta mensagem, bem como o MAC de origem e o MAC de destino. Se o

mecanismo de VLANs estiver activo será perguntado ao utilizador qual o VLAN ID para o qual deve ser encaminhado o pacote.

Este mecanismo inclui alguns procedimentos no seu conteúdo que fazem o reencaminhamento de mensagens quando o mecanismo de VLANs se encontra activo ou inactivo, bem como activam a decrementação de um *aging time* pela API quando não é recebido nenhum pacote pela *dummy* da FPGA.

Ao ser inserido um pacote por este procedimento é pedido ao utilizador que insira a porta de origem e o MAC de origem. Este par é guardado no *array* **lista_mac[nmacs+1]**, em que o MAC de origem é copiado para a posição de índice do *array* equivalente à porta de origem.

De seguida este procedimento encarrega-se de verificar se o mecanismo de VLANs está activo através do parâmetro de entrada **VLANmec**. Se este estiver a "**0**" é porque está inactivo o mecanismo e utiliza-se o procedimento **verifymac** para fazer o reencaminhamento da mensagem.

Se pelo contrário, o mecanismo de VLAN tiver o mecanismo de VLANs activado então utiliza-se o procedimento **verify_mac_vlans** para fazer o reencaminhamento da mensagem para as portas pertencentes à VLAN identificada pelo VLAN ID presente no pacote inserido.

Por fim ao ser aprendido um MAC a API deve ser informada que deve começar a decrementar o *agetime* respeitante a porta que foi aprendido o MAC pelo que o procedimento **ageconf** se destina aexecutar essa acção.

- **void verify_mac(unsigned long lista_mac[nmacs+1],unsigned long mecanismo[nportas], unsigned long mac, unsigned int portain)**

Este procedimento é constituído por dois "ciclos **for**" com **nportas+1** iterações sendo que o principal "ciclo **for**" verifica se a variável que apresenta o valor da porta que originou o pacote (**portain**) é diferente do valor da iteração. É também feito dentro do "ciclo **for** principal", um ciclo **while** que se se encarrega de verificar todas as posições de MAC correspondente a cada uma dar portas. Se o MAC de destino é igual ao que está presente na posição de índice da iteração **lista_mac** e for diferente de "**0xFFFFFF**" (posição de índice **nportas+1**), então é indicado ao utilizador que o pacote saiu na porta com o valor do índice da iteração e o seu respectivo MAC.

Se pelo contrário não for encontrada nenhuma entrada igual ao MAC destino em todas as iterações dos ciclos **while** executados para cada uma das portas até **nportas**, na iteração **nportas+1** o valor lido é "**0xFFFFFF**", pelo que salta para uma condição, que se o valor de **lista_mac[k]** for igual a "**0xFFFFFF**" será feito um "ciclo **for**" que verifica se o conteúdo de cada iteração do vector mecanismo **[nportas]** apresenta o mecanismo de *switch* associado (valor "**1**"), encaminhando para lá o pacote inserido. Não se deverá considerar a iteração que tem o valor igual à porta de origem. Desta forma é possível fazer o *flooding* quando não é detectado o MAC de destino na tabela **lista_macs[nmacs+1**] ou quando um pacote apresenta uma mensagem *Broadcast* com destino "**0xFFFFFF**" que deverá ser entregue a todos os *hosts*.

Na realidade este procedimento seria feito lendo uma memória da FPGA, se fosse detectado o MAC destino, o pacote guardado noutra memória seria enviado para a porta correspondente ao MAC de destino, caso contrário quando a FPGA lesse a última posição de memória detectaria que o seu conteúdo era "**0XFFFFFF**" sabendo que teria que enviar a todas as portas com o mecanismo de *switch* associado. De seguida faria a leitura da memória que continha os mecanismos associados a

cada porta e cada posição de memória que lesse e tivesse verificado que era um "**1**", teria que enviar para essa porta o pacote guardado em memória.

- **void verify_mac_vlans (unsigned int lista_vlan[nportas+1], unsigned long lista_mac[nportas+1], unsigned long mecanismo[nportas], unsigned long mac, unsigned int portain, unsigned int VLANid)**

Este procedimento é constituído por dois "ciclos **for**" com "**nportas+1**" iterações sendo que o principal "ciclo **for**" executa um ciclo **while** que verifica todas as entradas de MAC relativas a cada uma das portas. Iguala inicialmente uma variável "**k**" ao valor "**i**" da iteração do "ciclo **for**" e vai incrementando nportas em cada iteração até se achar o MAC correspondente ou até atingir um valor "**k**" superior ao valor de **nmacs**. Este processo verifica se a variável que apresenta alguma entrada do *array* de MACS corresponde a porta que originou o pacote (**portain**) é diferente do valor da iteração. Se o MAC de destino é igual ao que está presente na posição de índice da iteração **lista_mac**, for diferente de "**0xFFFFFF**" (posição de índice "**nmacs+1**") e o valor do parâmetro de entrada **VLANid** for igual ao valor que está na posição de índice "**i**" da iteração do "ciclo **for**" principal **lista_vlan**, então é indicado ao utilizador que o pacote saiu na porta com o valor do índice da iteração, o seu respectivo MAC e VLAN ID.

Se pelo contrário não for encontrado nenhuma entrada igual ao MAC destino em todas as iterações até **nportas**, na iteração "**nportas+1**" o valor lido é "**0xFFFFFF**", pelo que salta para uma condição, que se o valor de **lista_mac[k]** for igual "**0xFFFFFF**" será feito um "ciclo **for**" que verifica se o conteúdo de cada iteração do vector **mecanismo[nportas]** apresenta o mecanismo de *switch* associado (valor "**1**") e se o conteúdo de cada iteração do vector **lista_vlan[nportas]** é igual ao parâmetro de entrada **VLANid**, encaminhando para lá o pacote inserido. Não se deverá considerar a iteração que tem o valor igual à porta de origem. Desta forma é possível fazer o *flooding* quando não é detectado o MAC de destino na tabela **lista_macs[nmacs+1]** a um determinado VLAN ID ou quando um pacote apresenta uma mensagem *Broadcast* com destino "**0xFFFFFF**" que deverá ser entregue a todos os *hosts* que pertençam a determinada VLAN.

Na realidade este procedimento seria feito lendo uma memória da FPGA, se fosse detectado o MAC destino e o VLAN ID do pacote em cada memória destinada a guardar os MACs e VLAN IDs, o pacote guardado noutra memória era enviado para a porta correspondente ao MAC de destino de determinada VLAN, caso contrário quando a FPGA lesse a última posição de memória detectaria que o seu conteúdo era "**0xFFFFFF**", sabendo que teria que enviar a todas as portas com o mecanismo de *switch* associado e com o VLAN ID do pacote. De seguida faria a leitura da memória que continha os mecanismos e as VLANs associadas a cada porta e cada posição de memória que lesse e tivesse verificado que era um "**1**" ou que tinha o mesmo VLAN ID do pacote, teria que enviar para essa porta o pacote guardado em memória.

- **void ageconf(unsigned long list_mac[nmacs])**

Procedimento que é usado para comunicar à API o MAC que foi aprendido em determinada porta para que esta possa habilitar o mecanismo de decremento do *agetime* dessa porta.

Este procedimento começa por guardar no *array* **MAC[nmacs]** da memória partilhada os MACs presentes no *array* **mac_list[nmacs]** da FPGA *dummy*, depois activa a variável de configuração **confMAC** para indicar ao processo de *agetime* (**agetimeconf**) que houve alteração na tabela de MACs da *dummy* da FPGA e que pode receber a tabela de MACs por intermédio da memória partilhada. Este deverá receber esta lista e começar a decrementar todos os *agetimes* das portas que apresentem MAC diferente de "**0**".

Na realidade este procedimento terá que ser um encapsulamento de uma mensagem de configuração destinada a API com o *bit* de configuração **confMAC** activo e com os campos **MAC[32..0]** preenchido pelo que é lido em cada posição relativa à memória que guarda os MACs. Na figura em baixo mostra-se a forma como deve ser encapsulada a mensagem de configuração com destino a API.

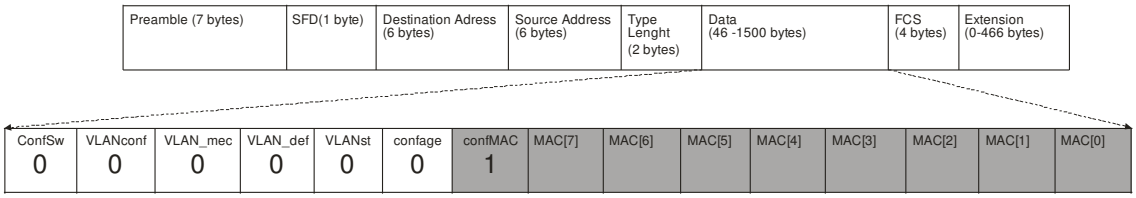

Figura 4.V: Pacote *Gigabit Ethernet* que contem o campo de dados para configurar o processo de *agetime* da API com os MACs aprendidos pela FPGA

Abaixo apresentar-se-ão os diagramas de fluxo respeitantes a cada um dos procedimentos descritos em cima para facilitar a compreensão destes procedimentos por parte do leitor.

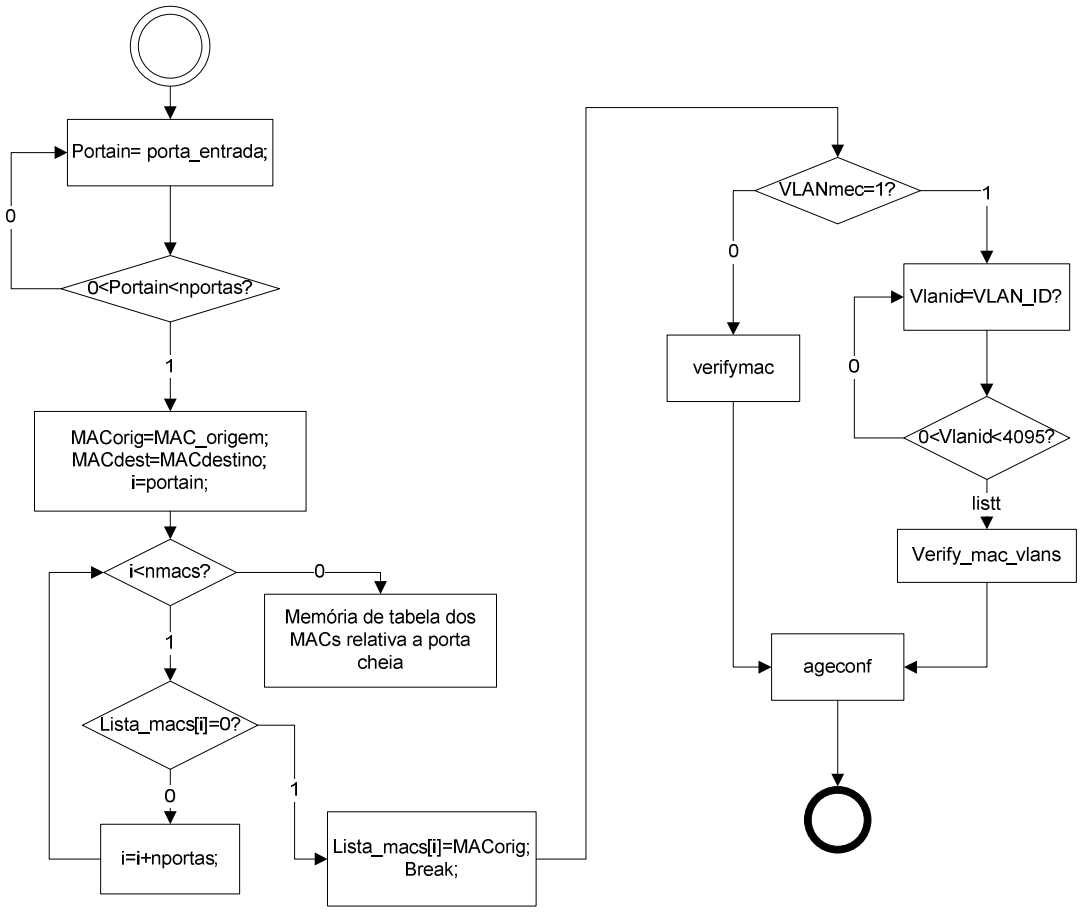

Figura 4.W: Diagrama de fluxo do procedimento insert_packet

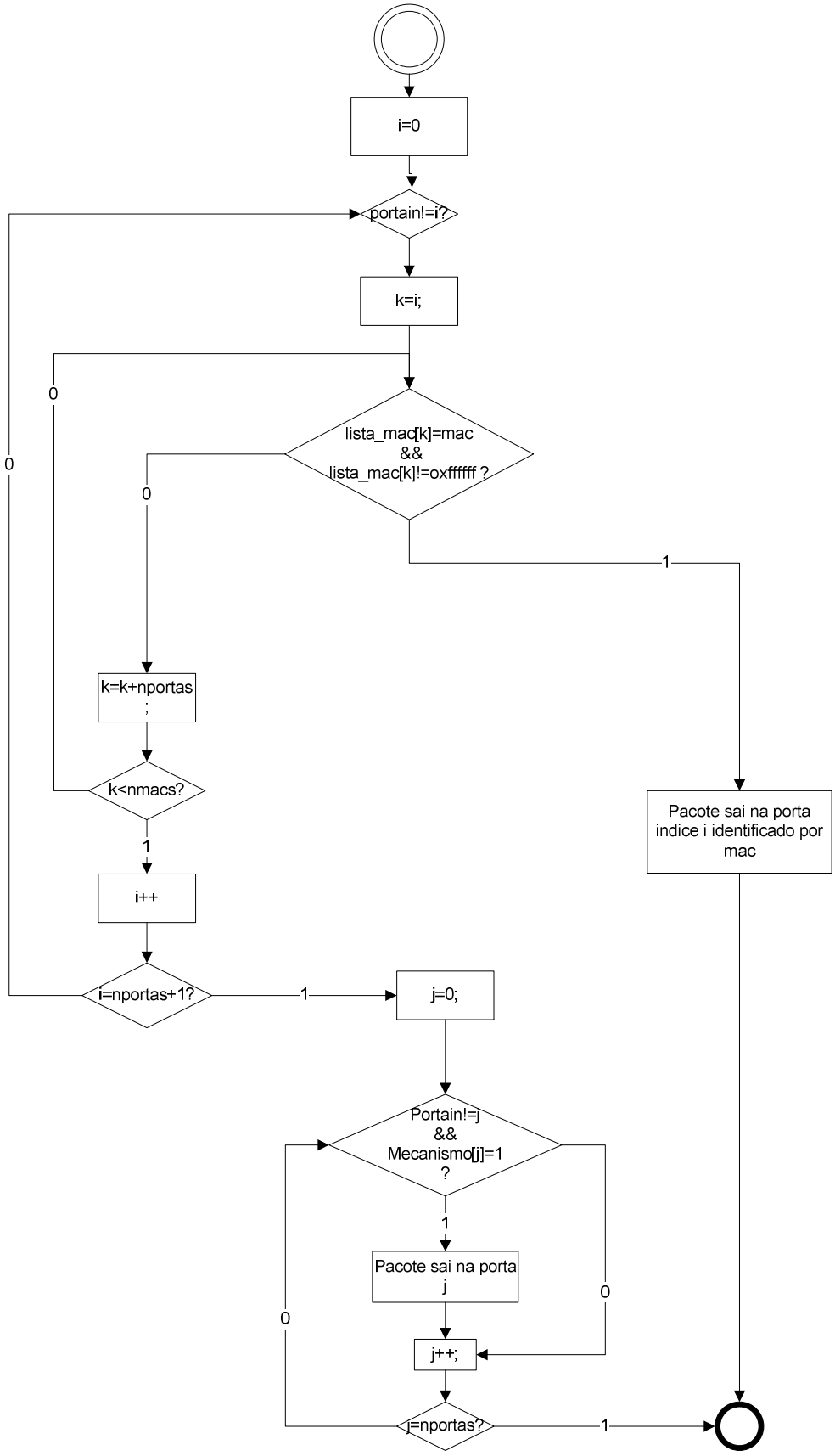

Figura 4.X: Diagrama de fluxo do procedimento verify_mac

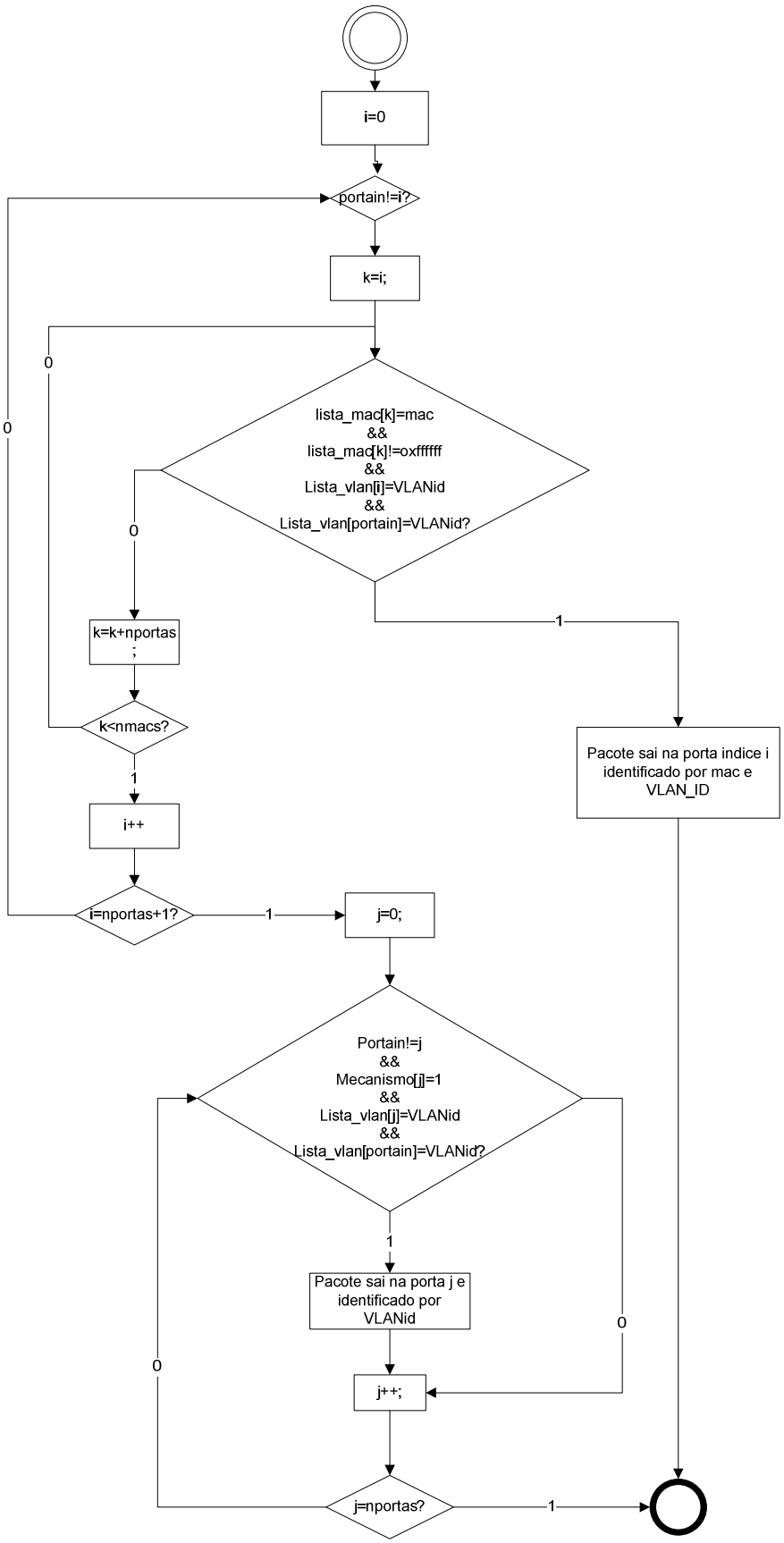

Figura 4.Y: Diagrama de fluxo relativo ao procedimento verify_mac_vlans

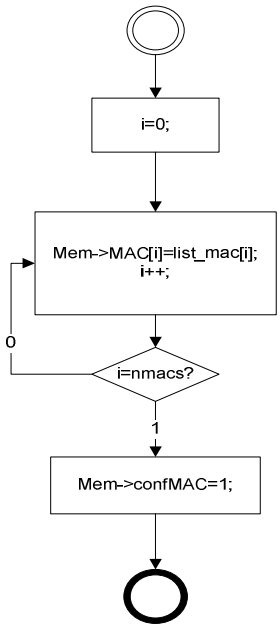

Figura 4.Z: Diagrama de fluxo relativo ao procedimento ageconf

Este capítulo procurou descrever com detalhe a implementação feita. O próximo capítulo irá apresentar e descrever os testes a que esta implementação foi sujeita de modo a comprovar a sua correcta operação.

5 Testes

Quando se desenvolve software é necessário fazer testes que confirmem se tudo o que foi implementado está a funcionar correctamente.

O software api_sw que foi implementado no âmbito deste mestrado e que está descrito nesta dissertação não é excepção e portanto foi sujeito à execução de testes que validam o seu correcto funcionamento de acordo com a especificação e descrição do capítulo anterior.

Utilizou-se o método de captura de imagens do ecrã (*screenshots*) para verificar se a comunicação entre a API, o processo de *agetime* e a *dummy* da FPGA é feita de forma correcta. Segue-se a explicação do que é apresentado nos *screenshots*.

A janela do lado superior esquerdo da figura seguinte mostra o que é obtido do processo "**agetimeconf**" que imprime uma tabela com três colunas de dimensão *nportas*. Nessa janela, a coluna da esquerda apresenta o número da porta, a do meio apresenta os MACs e a da direita apresenta o *agetime* configurado para cada porta, sendo que este por omissão é de 300 segundos.

A janela que se apresenta no lado inferior do *screenshots* de exemplo é o software api_sw desenvolvido para configurar a *dummy* da FPGA que inclui um conjunto de opções já explicadas na descrição sumaria da implementação.

A janela do lado superior da direita apresenta a *dummy* da FPGA que permite imprimir tabelas de MACs aprendidos pela *dummy*, tabela de mecanismos e de VLAN IDs associados a cada porta, bem como simular o envio de um pacote para o *dummy*. Ao ser inserido um pacote é introduzido a porta de entrada e o MAC origem, bem como é introduzido o MAC destino. No final é apresentada uma mensagem a indicar qual a porta (se o MAC já fizer parte da tabela de MACS) ou quais as portas onde o pacote foi retransmitido (mecanismo de *flooding*). É também disponibilizado opções para imprimir os mecanismos associados a cada uma das portas (ou seja, é impresso no ecrã o *array* relativo aos mecanismos indicando se está ou não associado o mecanismo de *switch*, apresentando em cada posição um "0" ou um "1), para imprimir os MACs associados a cada uma das portas (é impressa uma tabela com **nmacs** índices, definidos previamente e múltiplo da constante **nportas**) e para imprimir as VLAN IDs associadas a cada uma das portas (é apresentada uma tabela com o número da porta e a sua VLAN ID correspondente).

Na página seguinte mostra-se os *screenshots* de exemplo que mostram o que foi descrito acima.

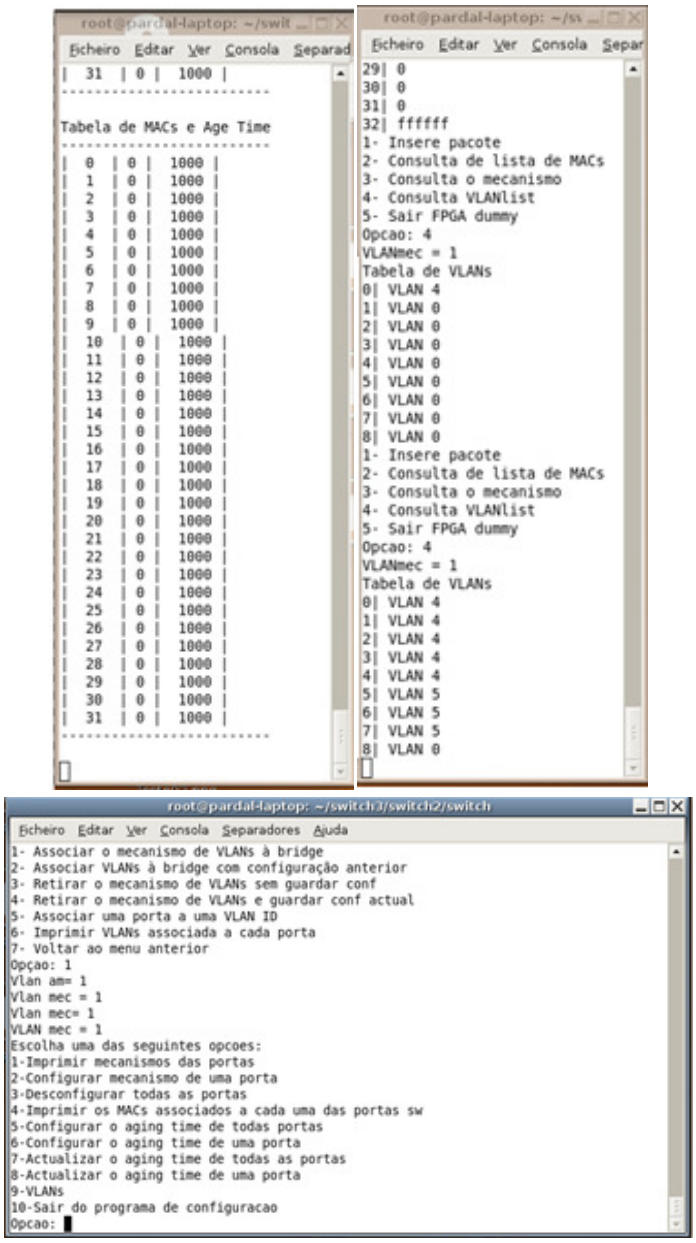

Figura 5.A: *Screenshots* exemplo para mostrar cada janela dos processos envolvidos nesta dissertação

O primeiro teste realizado foi configurar todas as portas com o mecanismo de *switch* (introduzir mecanismo 2 na API) e verificar se do lado da *dummy* da FPGA o vector que associa uma porta a um mecanismo apresenta todas as entradas a 1, indicando que o mecanismo de *switch* foi associado a todas as portas do *dummy*. Este teste foi realizado com êxito como mostra a figura em baixo.

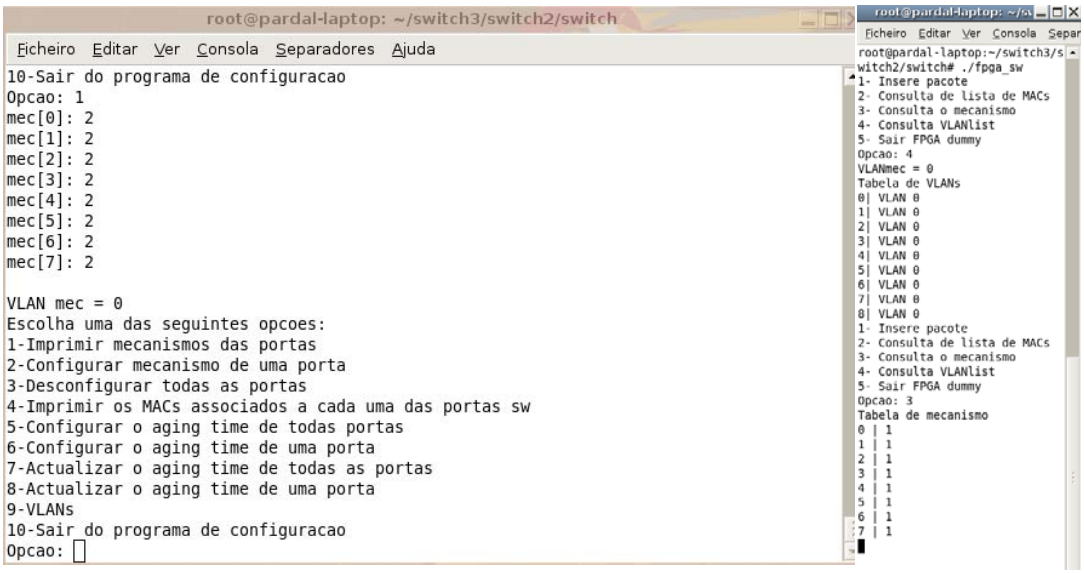

Figura 5.B: *Screenshots* das janelas que mostram o teste de configuração da tabela de mecanismos da dummy

O segundo teste implicou interagir entre o processo da API e o processo de *aging time*. Pretendeu-se escrever na tabela respeitante ao processo da esquerda um *agetime* de 250 segundos em todas portas. De referir também que a tabela de *agetime* contem 32 entradas de MACs para 8 portas, sendo que a cada porta, segundo o exemplo que se utilizou, pode ter até quatro entradas de MACs. Logo da forma como foi implementado, por exemplo as entradas 0,8,16, 24 da "**Tabela de MACs e *Agetime***" do processo "**agetimeconf**" são referentes à **porta 0** e o mesmo se passa com as outras portas, basta ter o número da porta e somar-lhe sempre o valor **nportas** (número de portas) para se saber quais são as entradas correspondentes a cada porta presentes na tabela.

A implementação que foi feita não é sequencial, já que é mais fácil ler ou escrever qualquer valor numa memória da FPGA da forma como foi implementada (implementação estática). Esta implementação associa uma constante a cada uma das portas, e cada vez que se pretenda ler (procura do MAC de destino na tabela de MACs da dummy da FPGA) ou escrever um valor na memória (aprendizagem do par (porta,MAC origem)), basta no caso de escrita saber a constante associada à porta e através da implementação de um contador em VHDL que incremente **nportas** a cada ciclo de relógio para determinar os endereços de memórias reservados a cada uma das portas presentes na FPGA. Deverão ser usados também comparadores para verificar se o valor que está presente no registo da memória que está a tentar ser escrito apresenta ou não o valor "0". Se apresentar "0" poder-se-á escrever o MAC Origem no registo, caso contrário o contador deverá ser incrementado nportas, e efectuar-se nova comparação até se encontrar um registo que não tenha sido utilizado (relembrar que a implementação também deve contemplar a inicialização dos registos das memórias a "0", tal como feito na *dummy* da FPGA). A leitura é bastante semelhante, apenas é feito e um varrimento a todos os registos da memória da FPGA até se encontrar o MAC destino, se não for encontrado, dever-se-á efectuar o mecanismo de *flooding*. Verificou-se neste exemplo que todas as entradas respeitantes a cada uma das portas foram configuradas de forma correcta pela API com um agetime de 250 segundos, pelo que este exemplo foi realizado com êxito.

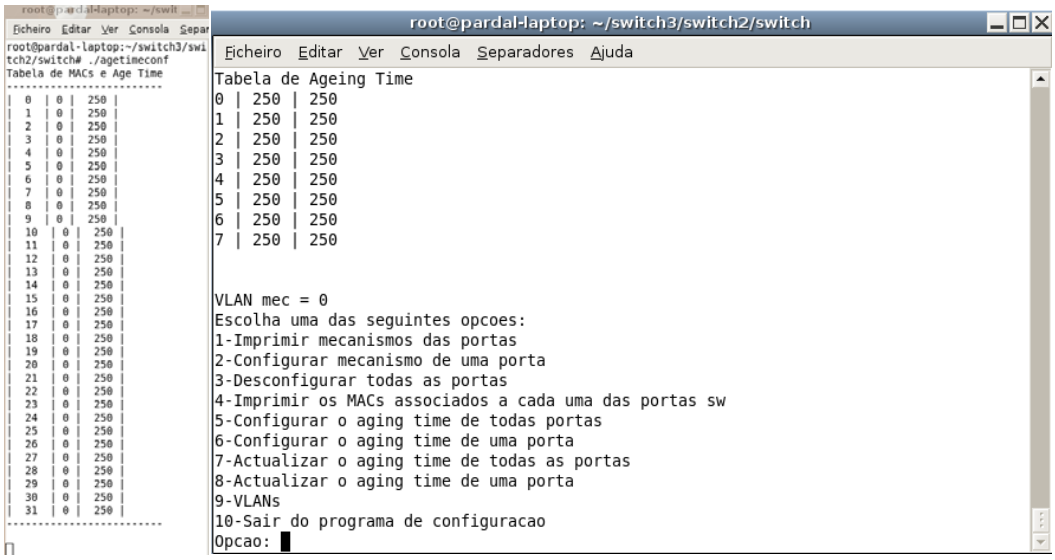

Figura 5.C: *Screenshot* que apresenta o teste efectuado para configurar o *agetime* de todas as portas do processo "agetimeconf" com o valor de 250 segundos

O terceiro teste serviu para testar uma das funções de *agetime* da API. Este teste serviu para confirmar se é possível configurar apenas uma porta com um *agetime* diferente das outras portas. Tentou programar-se a porta 2 com um *agetime* de 150 segundos, e pelo que se verifica na imagem abaixo, a janela respeitante ao processo "**agetimeconf**" apresenta em todas as entradas correpondentes à porta 2 (entradas 2, 10, 18 e 24) um tempo de 150 segundos. Pelo que este teste também foi concluído com êxito.

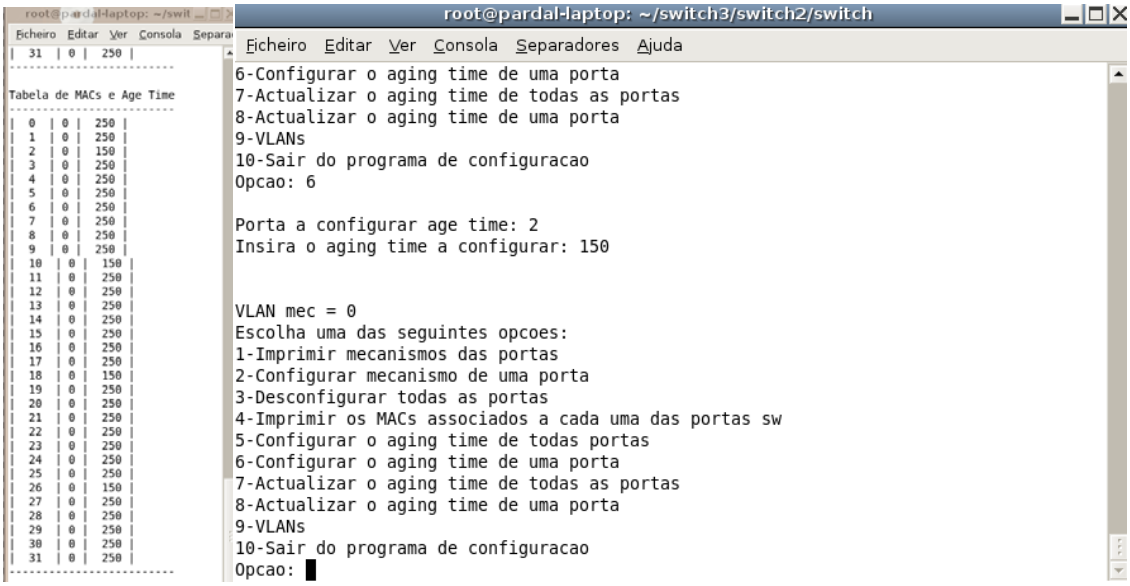

Figura 5.D: *Screenshots* das janelas que apresentam o teste efectuado para configurar o *agetime* da porta 2 com 150 segundos

Antes de se descrever o que foi realizado no quarto teste, é necessário referir que todas as entradas MAC usadas neste teste, apresentam um tamanho máximo de um **long** (32 bits). Um endereço MAC apresenta 48 bits, pelo que é necessário encontrar uma solução para este problema. A solução mais viável é ler os MACs como uma **string** e converter esta string em dois valores um long

(que apresenta os 32 bits menos significativos) e um short (que apresenta os 16 bits mais significativos). Esta escolha é feita a pensar no protocolo de *Spanning Tree*, já que torna-se mais rápido a comparação de 16 bits por parte de uma FPGA.

O quarto teste realizado foi efectuar um envio de um pacote proveniente da porta 2 e com MAC origem 22 e MAC destino 11. Como é óbvio, o menu que faz a inserção de um pacote não pergunta pelo VLAN ID do pacote, já que o mecanismo de VLANs não se encontra associado.

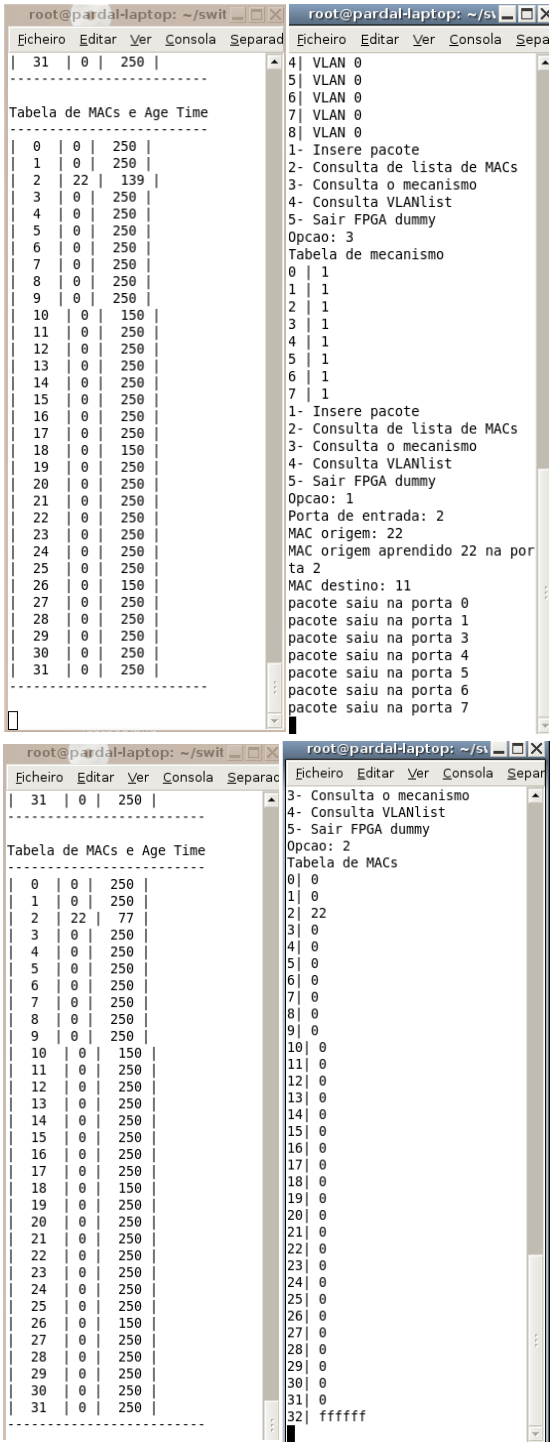

Figura 5.E: *Screenshots* relativos ao quarto teste

Se temos as portas todas configuradas com o mecanismo de *switch*, obviamente que se entra um pacote com MAC origem 22 na porta 2 este terá que ser retransmitido para todas as portas exceptuando ela própria (efectua o que se chama *flooding*), sendo o que se verifica no *screenshot* de cima (lado direito) da figura anterior. No *screenshot* de baixo (lado direito) verifica-se que a tabela de MACs da FPGA *dummy* está bem configurada, na medida em que a porta 2 está associada ao MAC 22. Pelos dois *screenshots* (lado esquerdo em cima e baixo) também se verifica que o processo "**agetimeconf**" está a funcionar correctamente, já que está a descontar o tempo relativo à porta 2 e recebeu correctamente a informação do MAC proveniente do processo da *dummy* da FPGA. Este teste também foi realizado com êxito.

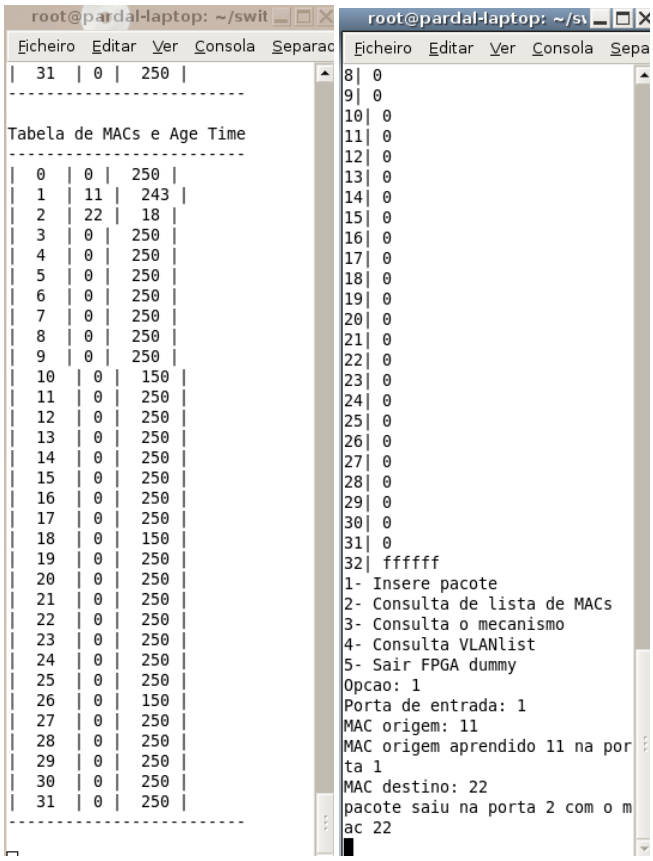

Figura 5.F: Primeiro screencshot relativo ao quinto teste

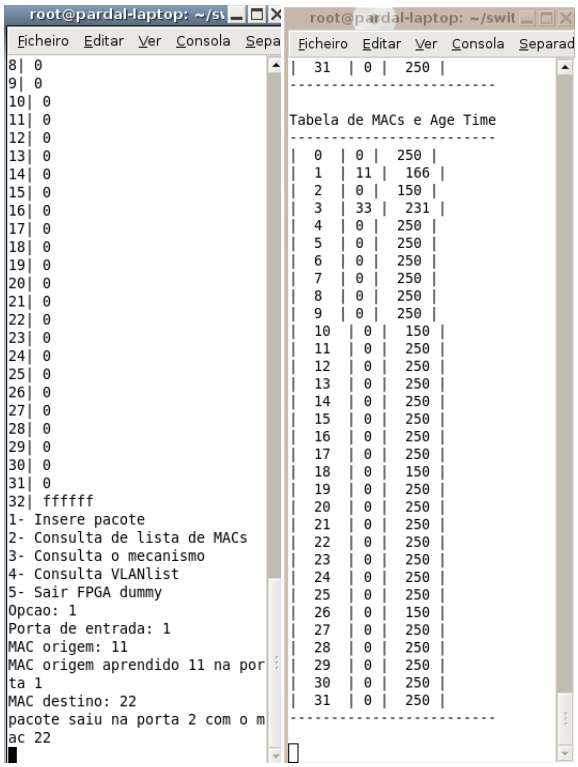

Figura 5.G: Segundo *screenshot* relativo ao quinto teste

O quinto teste pretende verificar se a *dummy* da FPGA está a retransmitir bem as mensagens quando o MAC de destino já foi aprendido pela sua tabela. Neste teste efectuou-se um envio de um pacote na porta de origem 1 com MAC origem 11 e MAC destino 22. O MAC destino 22 já tinha sido aprendido no quarto teste e já se tinha verificado que constava da tabela de MACs associada à porta 2 da *dummy* da FPGA. Verifica-se no primeiro *screenshot* que a mensagem foi bem entregue, saindo na porta 2.

De seguida fez-se um teste enviando um pacote originário da porta 3 com MAC origem 33 e MAC destino 11. Como o MAC de destino já foi aprendido pela tabela de MACs da *dummy* e consta da posição 1 dessa tabela, então só se espera que o pacote saia na porta 1. Entretanto verifica-se também que o tempo relativo ao *agetime* da porta 2 expira, pelo que é de esperar que as entradas da porta 2 da tabela relativa ao processo "**agetime**" sejam apagadas e o tempo de *agetime* previamente configurado pela API (150 segundos) seja reposto, verificando-se no *screenshot* da figura 5.G acima que isso acontece.

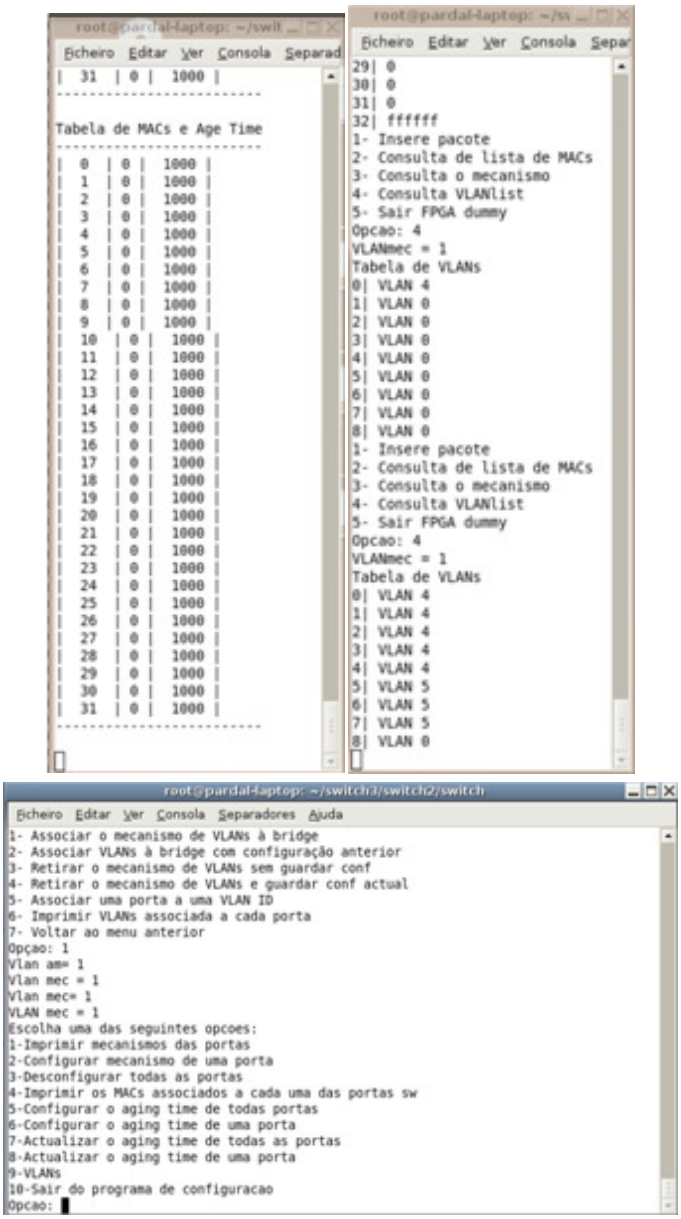

Figura 5.H: Primeiros *screenshots* das janelas do sexto teste

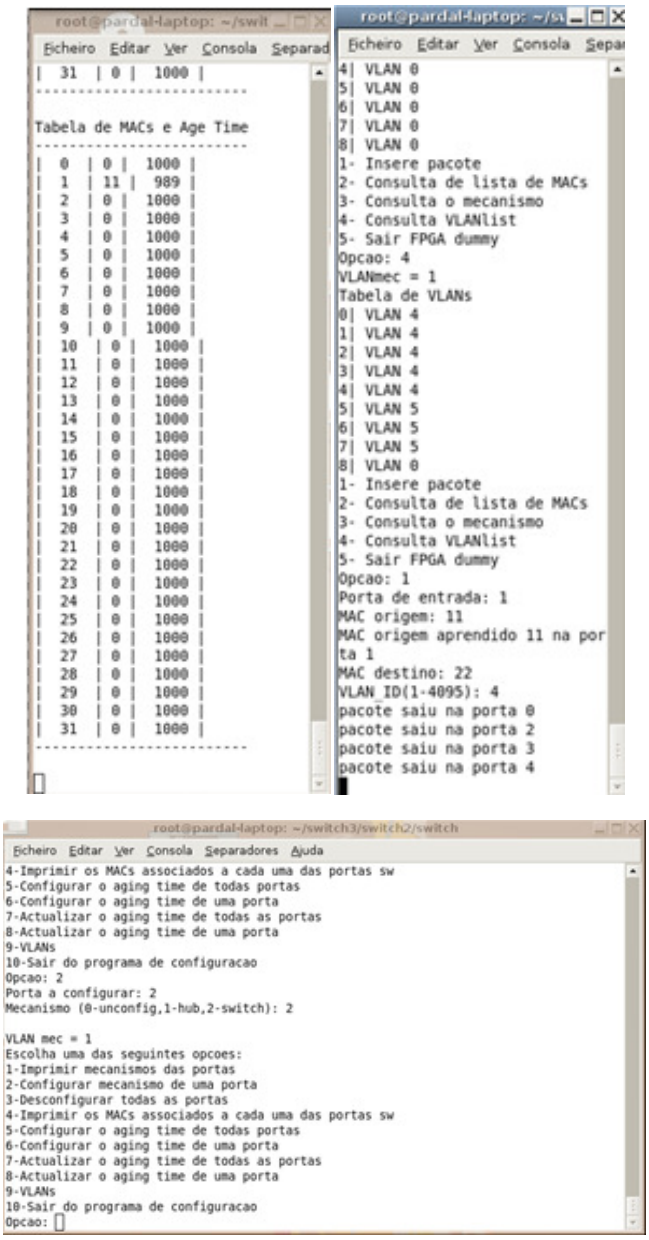

Figura 5.I: Segundos *screenshots* das janelas do sexto teste

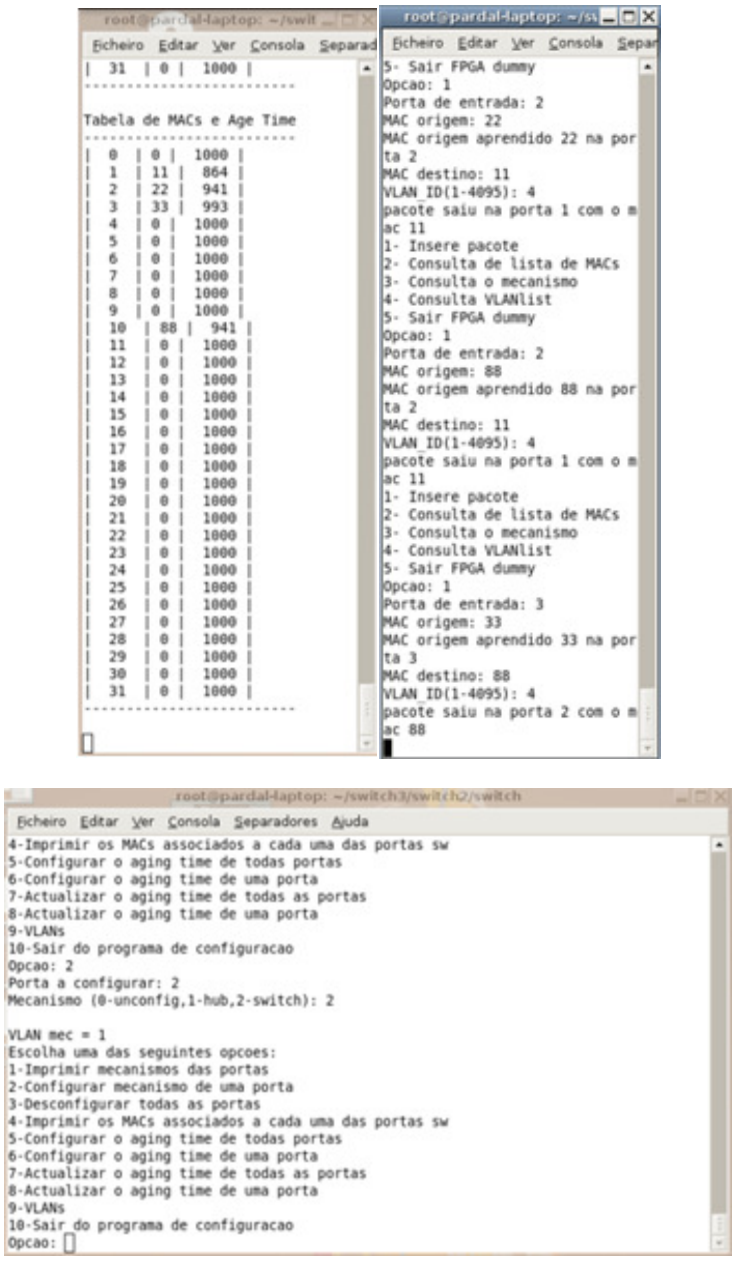

Figura 5.J: Terceiros *screenshots* das janelas do sexto teste

Para o sexto teste, configurou-se o *agetime* de todas as portas para 1000 segundos para dar tempo para efectuar alguns testes ao nível do protocolo de VLANS e configurou-se ainda os VLAN IDs correspondentes a cada uma das portas e apresentados na tabela 5.A em baixo. Pode-se verificar na *dummy* da FPGA à direita do primeiro *screenshot* da figura 5.H que foi tudo configurado de forma correcta.

Fez-se uma configuração utilizando a API desenvolvida, para configurar a tabela de VLAN IDs da *dummy* da FPGA que apresenta o seguinte cenário:

Porta	VLAN ID
0	4
1	4
2	4
3	4
4	4
5	5
6	5
7	5

Tabela 5.A: Configuração da tabela de VLAN IDs da dummy da FPGA

Após a *dummy* da FPGA ser configurada com os VLAN IDs tentou-se enviar um pacote da porta 1 com MAC origem 11 e MAC destino 22 e VLAN ID 4, pelo que se verifica no *screenshot* da figura acima que o pacote saiu nas portas correcta, ou seja, enviou o pacote para todas as portas associadas à VLAN 4. Isto pode verificar-se tudo no segundo *screenshot* da Figura 5.I.

No último *screenshot* presente na figura 5.J é possível verificar-se que entretanto foram aprendidos na tabela dois MACs originários da porta 2 (MAC 22 e 88), bem como um MAC da porta 3 (MAC 33). Pretendia-se verificar se a tabela do processo "**agetimeconf**" suportava mais que um MAC para uma porta bem como se as mensagens quando destinadas a um MAC especifico e este ter sido já aprendido pela *dummy* da FPGA, se era reencaminhado para a porta correcta. Ao enviarem-se dois pacotes consecutivos com o campo de VLAN_ID 4, com mac destino 11 (porta 1) e originários da porta 2 e com MACs de origem diferentes (primeiro pacote com MAC origem 22 e o segundo com MAC origem 88), foi possível verificar através do último *screenshot* que estes pacotes foram entregues com sucesso na porta 1 da VLAN 4. Também se testou o envio de um pacote destinado ao MAC 88 (porta 2), originário da porta 3 e MAC origem 33. Verifica-se também que este teste foi realizado com êxito, na medida em que o pacote é entregue na porta 2.

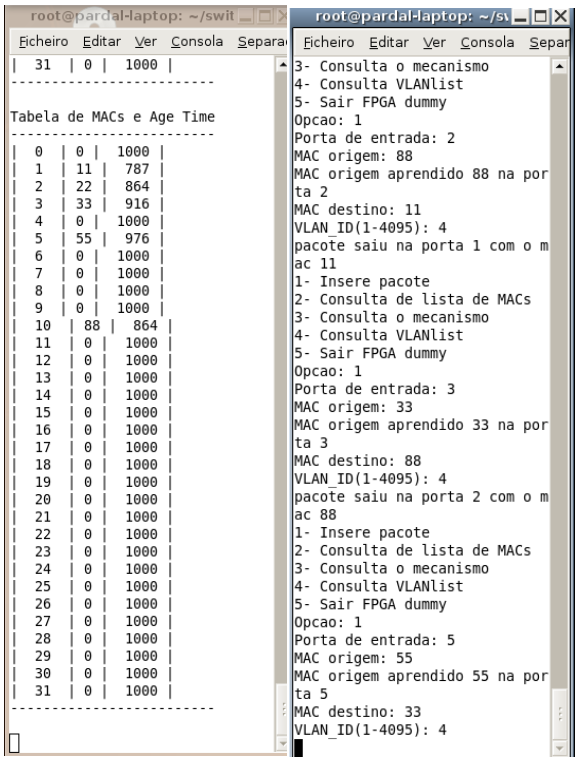

Figura 5.K: Primeiro *screenshot* do sétimo teste

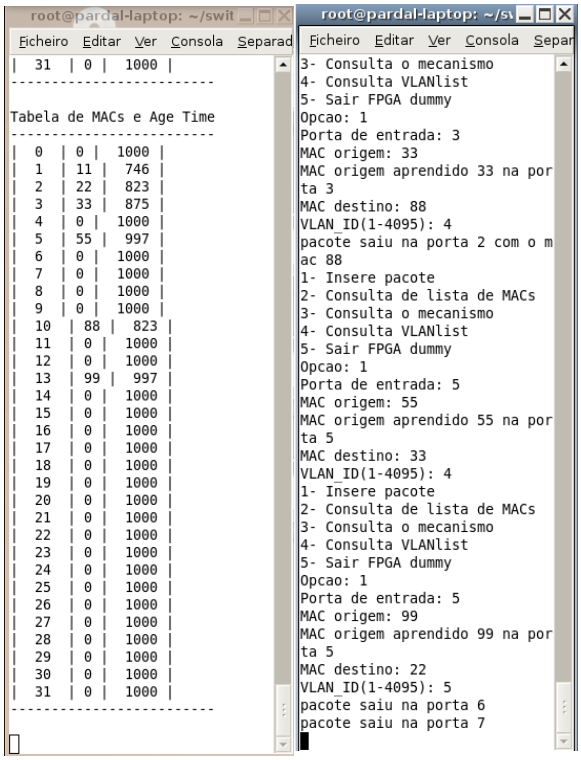

Figura 5.L: Segundo *screenshot* do sétimo teste

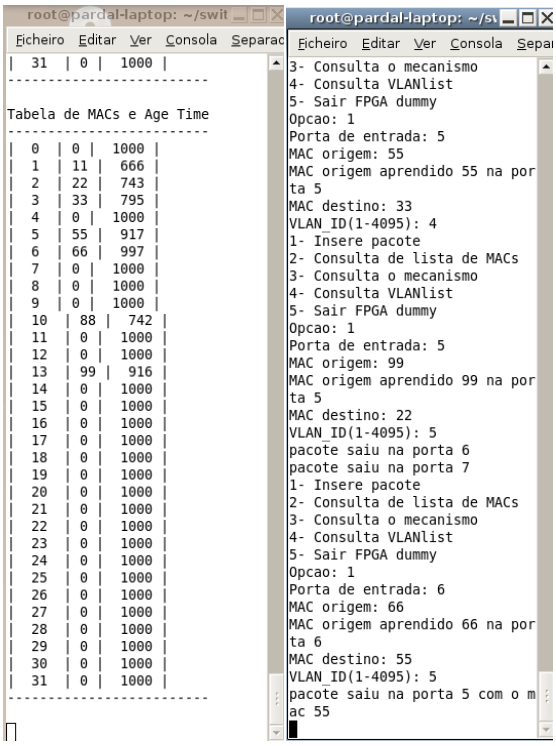

Figura 5.M: Último *screenshot* do sétimo teste

No primeiro *screenshot* do sétimo teste, verifica-se que foi enviado um pacote da porta 5 com MAC origem 55, sendo este MAC aprendido logo pela tabela do processo "**agetimeconf**". O MAC destino foi escolhido propositadamente como 33 e campo da mensagem VLAN_ID 4, já que o MAC 33 está na tabela de MACs e está associado à VLAN ID 4. O teste corre bem, já que o VLAN ID do campo do pacote é 4 e mesmo que o MAC 33 pertença à VLAN 4, como a porta 5 está associada à VLAN 5 o pacote não é entregue ao destino, tal como se esperava.

No segundo *screenshot* é possível verificar o envio de um pacote originário da porta 5 com MAC 99 e campo VLAN_ID 5 igual ao VLAN ID associado à porta de origem e com MAC destino 22 igual ao da porta 2 associada à VLAN 4. É de esperar que seja reenviado para todas as portas pertencentes à VLAN 5, já que apesar de o MAC 22 ter sido aprendido pela tabela de MACs, ele encontra-se associado a outra VLAN. Isso é verificado com êxito na medida em que o pacote é entregue a todas as portas (porta 6 e 7) da VLAN 5.

No último *screenshot* da figura 5.M é enviado um pacote originário da porta 6 associada à VLAN ID 5 e com MAC destino 55, MAC já aprendido pela tabela da *dummy* da FPGA e associado à porta 5 que por sua vez está associada à mesma VLAN do campo do pacote (VLAN_ID 5). Verifica-se que este pacote é também enviado com sucesso, estando o protocolo de VLANs correctamente programado.

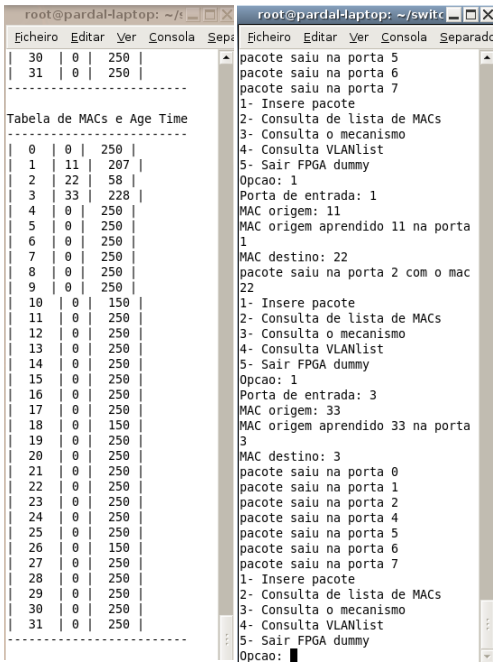

Figura 5.N: Primeiro *screenshot* do oitavo teste

Figura 5.O: Segundo *screenshot* do oitavo teste

Este oitavo teste permite visualizar a actualização do **agetime** quando há qualquer tráfego de pacotes numa porta que já tenha o seu *aging time* a ser decrementado. Como se pode observar nos *screenshots* em cima a porta 2 possui já um MAC aprendido na tabela (posição de índice 2 da tabela com o MAC 22), quando é inserido um pacote com origem na porta 2 e MAC origem 55 verifica-se que é adicionada uma nova entrada à tabela de MACs na posição de índice 10, e o **agetime** é actualizado com o anterior valor de **agetime** configurado pelo API, que neste caso é de 150 segundos. Os *screenshots* em cima mostram que o **agetime** relativo à porta 2 passa de 58 segundos para 143 segundos (no momento em que se tirou o *screenshot* já haviam sido descontados 7 segundos), mostrando que a actualização do **agetime** está a ser feita de forma incorrecta, já que

apenas a entrada de endereço MAC de origem do pacote é que deveria ser actualizada. Para isso o **array** correspondente aos **agetimes** da memória partilhada deveria ter o número de entradas de endereços MAC e sempre que houvesse recepção ou envio de um MAC deveria ser actualizada a posição de **agetime** referente a esses MACs na tabela do processo de "**agetimeconf**". De referir que o **array agetimeold** da memória partilhada está bem implementado e com a dimensão correcta, já que apresenta os valores de actualização do **agetime** de uma entrada MAC respeitante a uma porta.

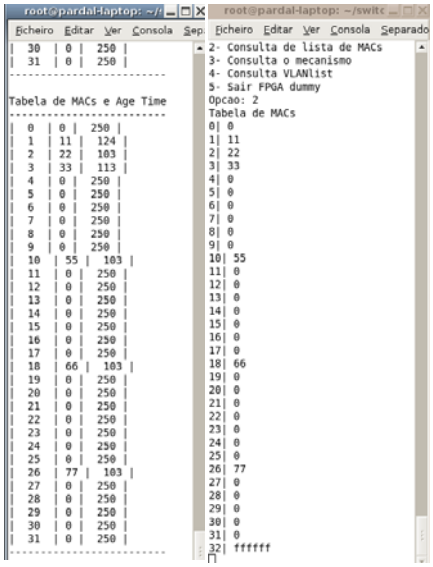

Figura 5.P: Primeiro *screenshot* do último teste

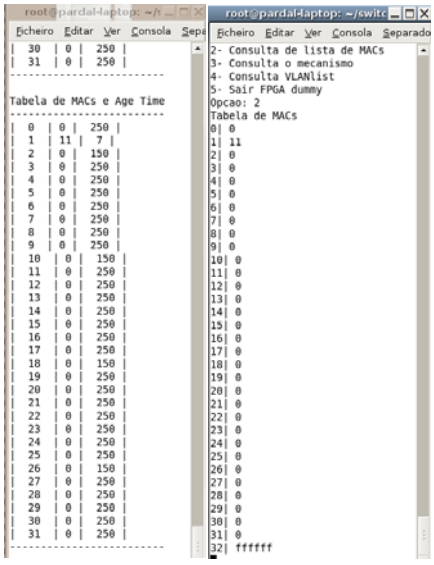

Figura 5.Q: Segundo *screenshot* do último teste

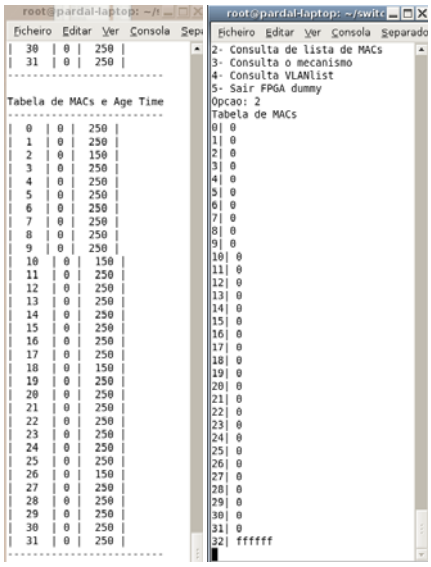

Figura 5.R: Terceiro *screenshot* do úlitmo teste

Os últimos *screenshots* pretendem verificar a interacção entre o processo "**agetimeconf**" e o processo relativo ao *dummy* da FPGA, já que é necessário que quando o **agetime** expire, seja comunicado à FPGA por parte do processo "**agetimeconf**" que é necessário apagar as entradas das portas cujo **agetime** expirou. Para isso inseriu-se numa das entradas da porta 1 o valor de MAC 11, na porta 2 inseriram-se os MACS 22,55,66,77 (colocaram-se todas as entradas de MAC relativas à porta 2 preenchidas) e por último na porta 3 inseriu-se numa das entradas da tabela de MACs o valor 33. Deixaram-se expirar os tempos, verificando como ficaria a tabela respectiva aos MACs na dummy. Verifica-se que o teste foi efectuado de acordo com o que foi implementado no capitulo anterior, mas de forma incorrecta. A interpretação desta matéria foi feita de forma errada, pois foi implementado o código de programação de forma a fazer a actualização do **agetime** ao nível das portas, devendo ter sido feito ao nível dos MACs. A solução implementada deverá ser alterada de modo a que se possa fazer a actualização do **agetime** de apenas uma entrada de MAC sempre que houver envio ou recepção de um pacote com origem ou destino desse MAC. Esta incorrecção deveu-se ao facto de se ter alterado o código na parte final do trabalho e não haver comunicação de minha parte ao meu orientador, coorientador e/ou colaboradores. Neste sentido e tomando por base o que foi implementado verifica-se que do primeiro *screenshot* para o segundo os **agetimes** relativos à porta 1 e 2 expiraram e a tabela de MACs da FPGA foi actualizada, ou seja nas posições onde constavam MACs associados à porta 1 e 2 foram apagadas. No último *screenshot* verifica-se que após expirar o **agetime** da porta 3, a tabela de MACs da FPGA fica toda a "**0**".

6 Conclusões e Trabalho Futuro

6.1 Conclusões

Inicialmente começou-se por programar uma API de modo a configurar uma simulação de um *hub* numa FPGA. Foram implementados dois processos, um software de configuração das tabelas que serve para guardar os conteúdos respeitantes a cada uma das portas e uma dummy que recebe por intermédio de uma memória partilhada a informação proveniente do software api_sw e apresenta o estado actual das tabelas de *hub*. É de relembrar que foram programadas duas tabelas, uma com a configuração actual das portas associadas ao mecanismo de *hub*, e uma com a configuração anterior. As duas tabelas foram programadas com o intuito de enquanto se fazia a escrita de uma tabela com os valores actuais de *hub*, a outra estaria activa até ao final desta estar programada, já que isto poderia induzir à perda de informação pois ter-se-ia que suspender o tráfego sempre que se efectuasse alguma alteração.

Quanto à segunda parte foi efectuada a implementação de uma API que fosse capaz de programar uma simulação de um Switch Ethernet numa FPGA. Foram necessários três processos, um relativo ao *software* api_sw configurador dos mecanismos (*Hub* e *Switch*), *agetime* e VLANs, outro processo de gestão de *agetime*s de portas (anexo à API) e um processo simulador de um Switch Ethernet baseando-se em determinados pressupostos apresentados no capítulo XXX.

A comunicação entre processos é efectuada por intermédio de uma memória partilhada, sendo que o processo relativo ao *software* comunica apenas num sentido, configurando os *agetimes* do processo de gestão de *agetime* das portas e dos mecanismos e VLANs relativos à *dummy* da FPGA. O processo de gestão de *agetime* comunica com a *dummy* da FPGA sempre que expire o *agetime* de uma porta, indicando que as entradas de MAC associadas a essa porta devem ser apagadas. Por sua vez, já o processo da *dummy* da FPGA apenas comunica com o processo de gestão de *agetime* sempre que haja uma alteração na sua tabela de MACs, indicando a este último processo que deverá começar a decrementar o *agetime* de uma porta ou actualizar o *agetime* para o tempo configurado pelo processo da API sempre que seja recebido ou enviado um pacote numa porta.

Após o capítulo de testes é possível verificar que no geral a comunicação efectuada entre a API e a *dummy* da FPGA é feita de forma correcta para os mecanismos que se implementaram. Quando se verifica a presença de várias entradas de endereços MAC por porta, houve um erro de interpretação, mas da forma como se interpretou é possível verificar que foi feita correctamente. A razão desta má interpretação foi referida anteriormente.

As funções e os procedimentos que foram programados para configurar os arrays de mecanismos e de VLAN IDs da *dummy* da FPGA por parte da API foram bem desenvolvidas, não apresentando qualquer erro na sua implementação, já que através dos testes foi possível confirmar que a comunicação foi feita de forma correcta, configurando os arrays com os valores pretendidos.

Há que referir que ao configurar o processo "**agetimeconf**" com o valor de "**n**" segundos introduzido por parte do utilizador também é feito de forma correcta, já que a coluna do array responsável por guardar os MACs e os *agetimes* associados a cada porta é bem configurada com os valores introduzidos pela API e com os MACs aprendidos pela FPGA *dummy* quando se insere um pacote nesta. Isto é possível também verificar no capítulo anterior.

A comunicação entre o processo "**agetimeconf**" e a *dummy* da FPGA deveria estar implementado de outra forma, já que pelos testes realizados verifica-se que quando o *agetime* de

uma MAC expira, ou seja, chega a "**0**", deveria ser comunicado ao processo da *dummy* da FPGA que era necessário apagar apenas a entrada que contem o MAC relativo a essa porta já que não é recebido ou enviado nenhum pacote durante o tempo que foi programado para a porta.

De referir, que quando se tem o mecanismo de *switch* sem VLANs associadas e se faz a inserção de um pacote na *dummy* da FPGA, este é retransmitido correctamente para a porta com determinado MAC destino ou para todas as portas ("*flooding*") que se encontrem configuradas com o mecanismo de *switch*, seja quando é encontrado o MAC de destino no array de MACs da *dummy*, seja quando este não for encontrado, respectivamente.

Verifica-se então que o mecanismo de *switch* e o protocolo de VLANs estão a funcionar de forma correcta, embora seja necessário efectuar algum trabalho futuro que será mencionado napróxima secção.

6.2 Trabalho Futuro

Nesta secção são referidas algumas sugestões para que se melhore a implementação realizada no âmbito do mestrado alvo desta dissertação.

A correcção da actualização do *agetime* para uma implementação com múltiplas entradas de endereço deverá ser feita de modo a que a actualização do agetime seja feita a nível de MAC e não de portas. Dever-se-á tomar como base a implementação base, já que são mínimas as alterações que devem ser feitas. Também dever-se-á complementar o uso de MACs com 48 bits, lendo os MACs de origem e de destino como uma **string** e convertendo estes valores para um **long** e um **short**, sendo que o **long** deverá ter os 32 bits menos significativos do MAC e o short os 16 bits mais significativos.

Depois disto implementado e baseando-se nas suposições feitas nesta dissertação dever-se-á programar uma FPGA de forma a interagir com os processos da API para verificar na prática que tudo implementado correctamente.

Dever-se-á continuar o estudo dos protocolos de Spanning Tree e IGMP, partindo das funções e procedimentos a programar para uma API que suporte estes protocolos no âmbito de se poder alargar as capacidades da API do *Switch* Ethernet. Estas funções e procedimentos já se encontram apresentadas no capítulo **Erro! A origem da referência não foi encontrada.**, nas secções relacionadas com cada um dos refridos protocolos.

Referências Bibliográficas

[1] S.Deering. RFC 1112, “Host extensions for IP multicasting”. Stanford University, Agosto, 1989.

[2] W.Fenner. “RFC 2236, Internet Group Management Protocol, Version 2”. Xerox PARC, Novembro 1997.

[3] IEEE. “Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks”. IEEE, 19 de Maio de 2006.

[4] LAN/MAN Standards Committee, IEEE. “IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges”. IEEE, 9 de Junho de 2004.

[5] LAN/MAN Standards Committee, IEEE. “IEEE Computer Society”. IEEE, 2005.

[6] Antônio Carlos T. Costa JR., Fernando Ney da C. Nascimento e Luís Eduardo C. Leite. “Implementação de um interface de programação de serviços multicast”. Universidade Federal do Rio Grande do Norte, 2000.

[7] Charles E. Spurgeon. “Ethernet: The definitive guide”. O’Reilly Media Inc., 2000.

[8] Linux.com: Everything Linux and Open Source. <http://www.linux.com/base/ldp/howto/Multicast-HOWTO-7.html> [Acedido a 6 de Janeiro de 2008]

[9] Celso Alberto Saibel Santos. “Como criar um segmento de mémoria compartilhada.”

< http://www.dca.ufrn.br/~adelardo/cursos/DCA409/node96.html> [Acedido a 13 de Março de 2008]

[10] Hadriel Kaplan & Robert Noseworthy. “The Ethernet Evolution From 10 Meg to 10 Gig. How it all Works!”. Atalanta, 2001.

[11] Robert Breyer & Sean Riley. “Switched, Fast and Gigabit Ethernet”. Sams, 3rd Edition, 30 de Dezembro de 1998.

[12] Datacom. “Spanning Tree Protocol”. <lways1.ifsc.usp.br/kyatera/dmswitch/resiliencia.pdf> [Acedido a 08 de Maio de 2008]

[13] Matti Tommiska and Jorma Skytt. “Dijkstra’s Shortest Path Routing Algorithm in Reconfigurable Hardware”. Helsinki University of Technology,2001.

[14] University of New Hampshire Interoperability Laboratory. “The multiple spanning tree protocol”. <ftp://ftp.iol.unh.edu/pub/bfc/UNH-IOL_BFC_Knowledgebase_MSTP.ppt> [Acedido a 12 de Julho de 2008]

[15] Intel Corporation. “L2 Bridge MicroACE Design Document”. <www.capsl.udel.edu/~fchen/projects/np/docs/manual/Pdf/ACEdesign/L2Bridge_MicroACE_Design.pdf> [Acedido a 10de Abril de 2008]

[16] Jorg Liebeherr & Magda El Zarki. “Chapter 5- LAN Switiching” de Mastering Networks: An Internet Lab Manual”. Addison-Wesley, 2003.

[17] B. Cain, Cereva Networks, S. Deering, I. Kouvelas, Cisco Systems, B. Fenner, AT&T Labs – Research, A. Thyagarajan, Ericsson. “RFC3376 - Internet Group Management Protocol, Version 3”. Outubro de 2002.

Anexo A

API estudada para o protocolo IGMP

Estudou-se uma *Framework Multicast* [6] que é implementada independentemente da infra-estrutura utilizada ou do serviço proposto.

A *framework* estudada define uma interface de propagação (API) para o desenvolvimento de aplicações que necessitam de transmissão *Multicast* em redes *Ethernet* e ATM. A interface ATM teve necessidade de implementação de um agente MARS (*Multicast Address Resolution Server*), que tem como objectivo resolver o endereçamento dos membros do grupo ou de um servidor de distribuição.

A interface estudada define um serviço de *Multicast* como sendo um conjunto de procedimentos e interfaces que permite a troca de mensagens para um grupo de participantes num ambiente de processamento e comunicação. Podendo ser usados objectos numa *thread*, *thread* dentro de um processo, processos dentro de uma estação e estações dentro de uma rede.

A implementação da arquitectura de um serviço de *Multicast* pode ser separada em duas partes: gestão do grupo e a construção de uma infra-estrutura de distribuição.

Serviço de gestão do grupo

Este serviço é responsável pela oferta de mecanismo para manter a base da informação dos grupos e componentes. Bem como fornecer as informações necessárias para a resolução de endereços de grupos utilizadas pelos mecanismo de transmissão.

A resolução de endereços é responsável pelo mapeamento de endereços de nível N em um ou mais endereços de nível N-1. Esta resolução pode ser feita de forma directa ou por intermédio de protocolos de resolução.

Se a resolução for de forma directa o protocolo da camada N traduz localmente o endereço da sua camada para a camada inferior através de uma função de mapeamento, tabela ou máscara (endereços IP sobre *Ethernet*).

Se a opção for uma resolução via protocolos, uma entidade de nível N faz a requisição a uma outra entidade para que retome os endereços N-1 correspondentes.

A não existência de endereços *Multicast* num nível inferior implica dois tipos de situações a serem resolvidas:

1. Se as informações do grupo estiverem centralizadas num serviço de resolução de endereços, a resolução dá-se por intermédio de um protocolo que pode ser utilizado requisitando desta forma a um servidor os endereços correspondentes. É de notar que o servidor não é responsável pela distribuição de mensagens, ficando esta a cargo do transmissor ou de algum servidor.
2. Se as informações estiverem centralizadas num servidor para distribuição de mensagens, a resolução pode ser feita através do mapeamento de endereços de grupo de nível N no endereço de nível N-1 do servidor, ou seja, o cliente desconhece a camada abaixo, mas o endereço do servidor que serve como endereço de *Multicast*.
3. Se a entidade contém os endereços de grupo aos quais ela pertence é feita uma requisição de resolução a todas as entidades, cabendo aos membros do grupo responder com o seu respectivo endereço de nível N-1.

Outras abordagens são possíveis através da combinação das três situações descritas anteriormente.

O serviço de gestão possui uma tabela que relaciona endereços de nível N com um ou mais endereços de nível N-1.

O bloco de programação *Group Manager* é o principal responsável do serviço de gestão de grupo, onde são definidas as seguintes operações:

- *Group Create*;
- *Group Destroy*;
- *Group Join*;
- *Group Leave*.

Quando um utilizador requisita este serviço envia uma notificação indicando o tipo de operação que deseja efectuar para um endereço que depende do modo como as informações estão armazenadas.

Quando se efectua uma operação deve-se distinguir o modo de notificação a todos os utilizadores, indicando quando se está no modo centralizado e quando se está no modo distribuído.

No modo centralizado, o endereço do servidor de resolução de endereços actualiza a sua base de dados e envia uma notificação a todos os utilizadores ou ao servidor mulitcast, enquanto no modo distribuído, as informações de grupo estão distribuídas por todos os clientes e é enviada uma notificação para o endereço de grupo *all-users* quando é efectuada alguma operação no serviço de gestão de grupo.

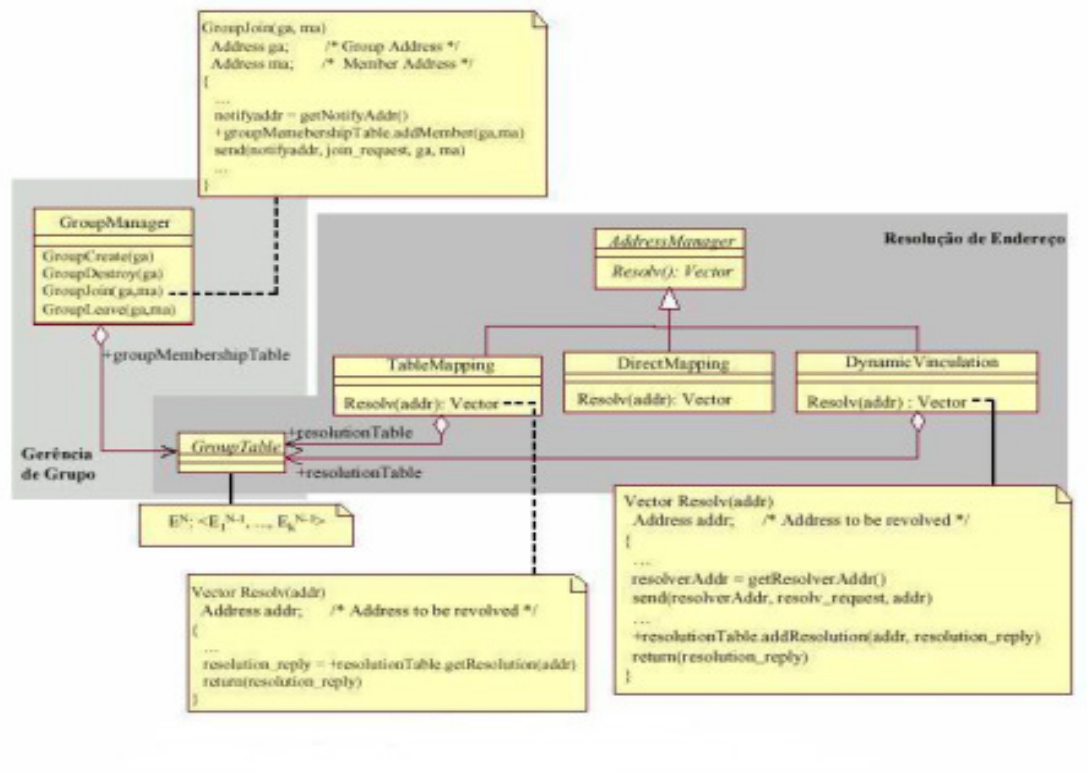

Figura Anexo A.A: Serviço de gestão de grupo

Serviço de routing

O serviço de *routing* pode ser implementado de forma centralizada ou distribuída. Sendo que no modo centralizado, todas as funções de *routing* bem como as informações de *routing* estão concentradas numa única entidade, e no modo distribuído, a entidade do sistema é responsável por manter e manipular as suas próprias informações de *routing*.

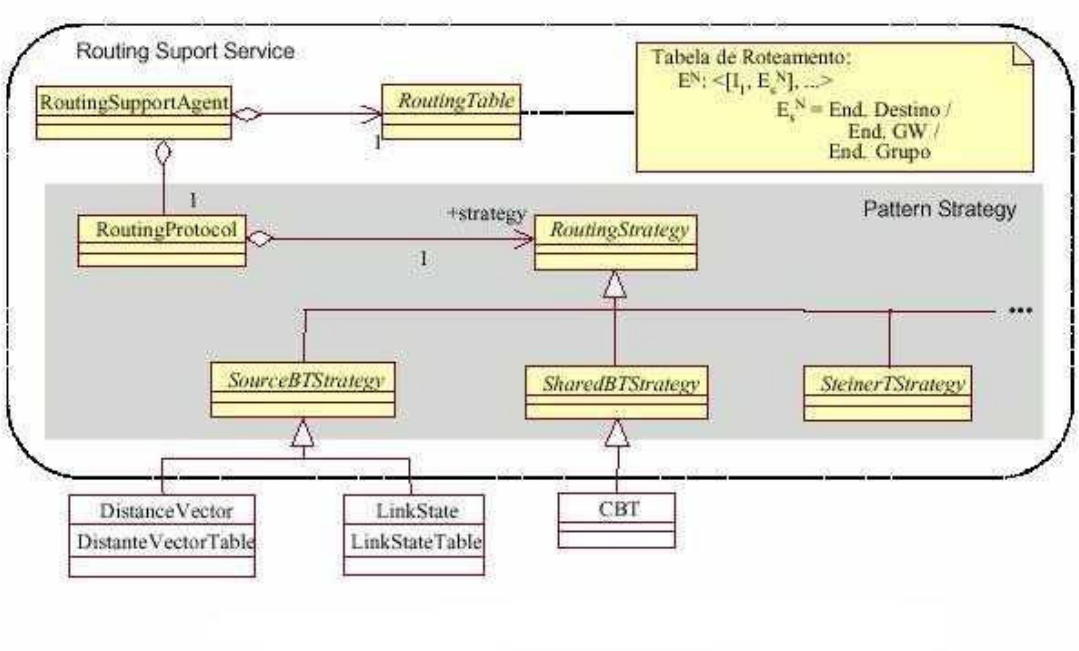

Figura Anexo A.B: *Framework* de suporte ao serviço de *router Multicast* estudado

Esta *Framework Multicast* foi desenvolvida em *Linux*, sendo uma interface destinada a aplicações Java e consistindo num pacote de classes a que chamaram *MulticastAllocationService*:

- *Group Management Server*;
- *Group Manager*;
- *Group Record*;
- *Code Message Protocol*.

A gestão do grupo ocupa-se:

- Da criação de um servidor que contém a base de informações sobre grupos – Servidor de gestão do grupo (classe *Group Management Server*);
- De uma interface de aplicação de cliente com este servidor pela classe *Group Manager*;
- O Servidor de Gestão de Grupo responsabiliza-se pela manutenção de todas as informações sobre grupos e dos seus participantes;
- O protocolo IP responsabiliza-se pela troca de mensagagens entre servidor e os utilizadores;
- Nesta interface é implementado um objecto *MeshTable*, que é uma tabela do servidor de gestão de grupo que apresenta um campo identificador de grupo (**idGroup**) que aponta para um objecto da classe *Group Record*.

Um objecto da classe *Group Record* é descrito da seguinte forma:

```
public class GroupRecord(srintg idGroup)
{
        InetAdress ipGroup;
        InetAdress ipCreator;
        String description;
…
}
```

- O campo **idGroup** identifica univocamente um grupo. Este campo tem como objectivo abstrair o utilizador da interface de endereços IP *Multicast*, fazendo com que seja mais fácil a identificação de um grupo pelo utilizador. O servidor de gestão de grupo associa um endereço IP de *Multicast* aos grupos criados.
- O campo **ipMulticast** representa o endereço IP *Multicast* associado a um determinado **idGroup**.
- **ipCreator** representa o endereço IP *Unicast* do criador do grupo. A existência deste campo deve-se já que apenas o criador do grupo é que poderá destruir o mesmo.
- **ipListGroup** é uma lista de endereços IP *Unicast* de todos os membros de um grupo.
- A string description contém uma descrição da finalidade do grupo e informações adicionais.

Um objecto **GroupCreate** é descrito da seguinte forma:

```
public static int GroupCreate(string idGroup, string description)
{
        …
        server.println(CodeMessageProtocol.Group_Create+idGroup+
        CodeMessageProtocol.Separator+description);
        …
}
```

O objecto acima descrito adiciona uma entrada na tabela para um grupo especificado pelo campo **idGroup** e adiciona-lhe uma descrição especificada pela string description.

É de referir que o campo ipListGroup permanece vazio até que seja enviada uma mensagem do tipo **GroupJoin**, sendo esta mensagem responsável pelo adicionamento de membros ao grupo criado.

Se o grupo não for criado por algum motivo é necessário que o servidor retorne uma mensagem avisando que não foi possível criar um grupo e especificando o motivo que deverá fazer parte da classe **CodeMessageProtocol**. Também deverá retornar um inteiro do tipo public static int, indicando o resultado desta operação.

O objecto **GroupDestroy** é descrito da seguinte forma:

```
public static int GroupDestroy(string idGroup)
{
…
```

server.println(CodeMessageProtocol.GroupDestroy+idGroup);

…

}

Este objecto ocupa-se da destruição de um grupo identificado por **idGroup**. O grupo poderá ser destruído pelo seu criador.

O servidor deve retornar uma mensagem caso não seja destruído o grupo.

Nesta *Framework* falta verificar os objectos que se ocupam pelo adicionamento e afastamento de utilizadores de um grupo, **GroupJoin** e **GroupLeave**, respectivamente.

O objecto **GroupJoin** é descrito da seguinte forma:

public static *MulticastSocket* GroupJoin(string idGroup)

{

…

server.println(CodeMessageProtocol.Group_Join+idGroup);

…

}

Este objecto é responsável pelo adicionamento de um membro no grupo identificado pelo **idGroup**, sendo adicionado o endereço IP *Unicast* do cliente ao ipListGroup.

Se não existir o grupo, o servidor deverá retornar uma mensagem de erro ao cliente, avisando este que o grupo identificado com essa **idGroup** não foi criado.

O objecto **GroupLeave** encontra-se descrito da seguinte forma:

public static int GroupLeave (string idGroup)

{

…

server.println(CodeMessageProtocol.Group_Leave+idGroup);

…

}

O objecto **GroupLeave** remove um membro do grupo especificado pelo **idGroup**. É de referir que o endereço IP *Unicast* do cliente é retirado da ipListGroup.

Este objecto deverá também retornar um inteiro para indicar o sucesso ou não da operação.