Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2012

Ricardo Jorge Gonçalves Ribeiro

# Switch Ethernet Distribuído para Sistemas Embutidos

# Distributed Ethernet Switch for Embedded Systems

Ricardo Jorge
Gonçavels Ribeiro

# Switch Ethernet Distribuído para Sistemas Embutidos

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 Doutor Arnaldo Oliveira e do Doutor Paulo Pedreiras, Professores Auxiliares do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Este trabalho é financiado por Fundos Nacionais através da FCT - Fundação para a Ciência e a Tecnologia no âmbito do projecto Serv-CPS -PTDC/EEA-AUT/122362/2010.

**o júri / the jury**

| | |
|---|---|
| Presidente / President | **Prof. Doutor José Alberto Gouveia Fonseca**<br>Professor Associado, Universidade de Aveiro |
| Arguente Principal / Main Examiner | **Prof. Doutor Valter Filipe Miranda Castelão da Silva**<br>Professor Adjunto, Universidade de Aveiro |
| Orientador / Advisor | **Prof. Doutor Arnaldo Silva Rodrigues de Oliveira**<br>Professor Auxiliar, Universidade de Aveiro |
| Co-Orientador / Co-Advisor | **Prof. Doutor Paulo Bacelar Reis Pedreiras**<br>Professor Auxiliar, Universidade de Aveiro |

**agradecimentos /**
**acknowledgements**

Em primeiro lugar gostaria de agradecer ao Professor Doutor Arnaldo Oliveira e ao Professor Doutor Paulo Pedreiras. Um muito obrigado pela disponibilidade demonstrada e paciência na orientação deste projecto. Agradeço também o incentivo, as sugestões e conhecimentos científicos transmitidos que foram importantes durante estes meses de trabalho.

Gostaria de agradecer também aos meus pais, avós e irmã pelo grande apoio e preocupação ao longo de toda a minha vida. Obrigado por estarem sempre disponíveis quando precisava e nunca faltarem com nada. Obrigado pela oportunidade de estudar e adquirir competências mais sólidas para o meu futuro.

Agradeço também à minha namorada, pela força e compreensão ao longo do meu percurso académico.

Um muito obrigado aos meus amigos pelo grande espírito de camaradagem e amizade que tornaram o meu percurso académico mais harmonioso e enriquecedor.

**Palavras Chave**

Sistemas Embutidos, Sistemas de Tempo Real, Sistemas Distribuídos, Comunicações de Tempo Real, Comunicações em Ambientes Industriais, Ethernet, Rede Ethernet Comutada.

**Resumo**

O uso de sistemas embutidos está cada vez mais generalizado em diversas áreas nomeadamente na automação industrial, aviónica, automóvel e produção de energia. Estes sistemas são utilizados para controlo de processos e gestão de aplicações, como na segurança das pessoas e bens materiais, que frequentemente apresentam requisitos temporais estritos no que envolve a execução das tarefas para as quais estão destinados. É usual que muitos dos sistemas embutidos estejam inseridos numa rede distribuída, constituída assim por vários subsistemas inteligentes e autónomos que cooperam entre si e partilham uma linha de comunicação que garante a conclusão dos objectivos especificados. Vários protocolos de comunicação foram desenvolvidos de modo a proporcionarem às redes distribuídas garantias em termos de determinismo, latência e previsibilidade.

O propósito inicial da *Ethernet* era a sua utilização em redes de dados em sistemas domésticos e empresariais onde os requisitos temporais não são críticos. Actualmente esta tecnologia tem vindo a ser utilizada como uma solução em sistemas embutidos distribuídos, nomeadamente através da utilização de redes comutadas - *Switched Ethernet Networks*. De facto, a flexibilidade e velocidade desta tecnologia foram fortes motivações para que na última década tivessem sido desenvolvidos vários protocolos que permitem a sua utilização em aplicações críticas de tempo real. É no entanto necessário um ponto de equilíbrio entre desempenho, custo e fiabilidade para aplicação da *Ethernet* nestes sistemas.

Esta dissertação apresenta o desenvolvimento de uma infraestrutura que permite a utilização da *Ethernet* em sistemas embutidos de tempo real para tráfego periódico e esporádico suportado numa rede em anel. Os nodos da rede foram desenvolvidos em módulos que são inseridos no *kernel* de sistemas *Linux*. É especificada a rede desenvolvida, a estrutura interna dos nodos presentes na mesma como também os mecanismos aplicados para cumprimentos de tempo real periódico e utilização eficiente da largura de banda. Com o objectivo de validar a implementação e avaliar o seu desempenho, foram realizadas diversas experiências, cujos resultados se encontram também presentes nesta dissertação.

**Palavras Chave**

Embbed Systems, Real Time Systems, Distributed Systems, Real Time Communications, Communications in Industrial Environments, Ethernet, Switched Ethernet Networks.

**Abstract**

Nowadays, the use of embedded systems has become ubiquitious in various industrial fields including automation, avionics, automotive and energy production industries. These systems are commonly used for process control and application management that need strict time requirements in order to execute propperly and ensure people's and material safety. Usually due to structural constraints and efficient resources management many embedded systems are implemented in a distributed network composed by several smart and autonomous subsystems with different functionalities that cooperate together to achieve a common goal.

The Ethernet was initialy developed to be used in household and business data networks where the time constrainst are not critical. Currently this techonology has been largely used as a solution in embebbed distributed systems, namely by using Switched Ethernet. In fact communication speed and technology flexibility motivated the development of several protocols to be used in real time applications in the past decades. However the use of Ethernet technology in these systems requires a compromise between performance, cost and reliability.

This dissertation presents the development of a infastructure that uses Ethernet in real time embebbed systems to support periodic and sporadic traffic in a ring topology based network. The network nodes were designed in modules that are included in the kernel of the Linux system. It also specifies the network developed, the internal strucure of the nodes and all the procedures used to assure the periodic real time communications and the efficient use of the bandwith. In order to validate and evaluate the implemetation several tests were conducted and are also presented in this document.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Lista de Acrónimos

**API** *Application Programming Interface.*

**ASIC** *Application Specific Integrated Circuits.*

**BE** *best-effort.*

**BPDU** *Bridge Protocol Data Unit.*

**CAN** *Controller Area Network.*

**CD** *Collision Detection.*

**CRC** *Cyclic Redundancy Check.*

**CSMA** *Carrier Sense Multiple Access.*

**EPL** *Ethernet PowerLink.*

**FCFS** *First-Come First-Served.*

**FCS** *Frame Check Sequence.*

**FPGA** *Field Programmable Gate Array.*

**IFG** *Interframe Gap.*

**IFS** *Inter Frame Spacing.*

**KLM** *Kernel Loadable Module.*

**KMOD** *Kernel Module.*

**LAN** *Local Area Network.*

**LLC** *Logical Link Control.*

**MAC** *Medium Access Control.*

**NAPI** *New Application Programming Interface.*

**OBD** *On-Board Diagnostics*.

**OSI** *Open Systems Interconnection*.

**RC** *rate-constrained*.

**SFD** *Start of Frame Delimiter*.

**SKB** *Socket Buffer*.

**SoA** *Start of Acyclic*.

**SoC** *Cycle Start*.

**STP** *Spanning Tree Protocol*.

**TDMA** *Time Division Multiple Access*.

**TT** *time triggered*.

**UTP** *Unshielded Twisted-pair*.

**VLAN** *Virtual Local Area Network*.

# Capítulo 1

# Introdução

Nos últimos anos, o uso de sistemas embutidos para controlar e gerir diversos tipos de aplicações e equipamentos tem vindo a generalizar-se. O avanço da tecnologia e a diminuição do custo da mesma permitiu adicionar alguma inteligência nos equipamentos tendo em vista a realização de aplicações específicas. Ao contrário de sistemas de uso geral onde os equipamentos apresentam grande versatilidade e capacidade para realização de várias tarefas, os sistemas embutidos são especializados para a realização de uma tarefa ou um conjunto reduzido de aplicações definidas à partida. Usualmente os sistemas embutidos são responsáveis por receberem a informação proveniente de sensores, processam essa mesma informação e com base nas especificações definidas accionam os actuadores. Estes equipamentos são produzidos em grande escala e portanto é necessário analisar algumas características como a área, o consumo de potência e os custos de desenvolvimento por forma a encontrar-se um compromisso entre os recursos necessários e o desempenho do sistema na realização da tarefa pretendida. Frequentemente estes equipamentos estão inseridos em sistemas distribuídos, em que cada nodo da rede pode executar operações diferentes mas que cooperam entre si e partilham uma linha de comunicação tendo como objectivo concretizar uma finalidade comum.

Actualmente o paradigma dos sistemas embutidos distribuídos tem vindo a crescer em ambientes industriais, existindo já em várias aplicações nomeadamente na indústria da automação, na indústria automóvel, aviónica e aeroespacial. Uma das principais motivações para distribuição de sistemas é a partilha de recursos na rede, nomeadamente *software*, *hardware*, dados, serviços, etc. Aliada a este factor, a capacidade de existir paralelismo na execução de tarefas aumenta o desempenho destes sistemas. Os sistemas distribuídos fornecem alguma redundância na rede, pois falhas parciais de subsistemas são fáceis de detectar e podem não prejudicar gravemente o sistema. No entanto, a distribuição de sistemas implica um aumento da complexidade e assim do custo para o seu desenvolvimento. Ainda no que diz respeito aos ambientes industriais, diversos estudos foram realizados de modo a desenvolver nodos inteligentes com capacidades de comunicação e processamento ao nível das camadas mais baixas dos protocolos de comunicação.

A troca de informação em aplicações industriais necessita de requisitos temporais estritos tendo em vista o funcionamento correcto e seguro dos sistemas, ao contrário das aplicações doméstica e empresariais, em que as restrições temporais não são tão rigorosas. Em termos de qualidade de serviço, as comunicações industriais, que representam comunicações de tempo real, privilegiam os requisitos temporais, nomeadamente *deadlines*, tempos de resposta e atrasos, enquanto nas comunicações sem especificações temporais são normalmente avaliadas

métricas como *throughput* ou a justiça na selecção de tráfego. No sentido de cumprir as diferentes especificidades foram desenvolvidas tecnologias, que operam nas camadas mais baixas do protocolo de comunicação, designadas de *fieldbuses*. Os *fieldbuses* são sistemas de comunicação interligam os sensores, actuadores e controladores suportando transferências de pequenos fluxos de informações com comportamentos temporais bem definidos. As mais utilizadas hoje em dia são *Controller Area Network* (CAN), *ProfiBus*, *DeviceNet*, *WorldFip*, entre outras.

## 1.1 Enquadramento e Motivação

Ao longo das últimas décadas a tecnologia *Ethernet* evoluiu rapidamente. A elevada velocidade na troca de informação, que hoje em dia pode atingir os *Gbps*, é um dos factores que torna aliciante o uso desta tecnologia mesmo em situações para a qual não foi projectada, como as comunicações industriais. Trata-se de uma tecnologia cujo o custo de produção é reduzido e já se encontra presente em vários tipos de equipamentos, facilitando de certo modo a sua utilização. A *Ethernet* encontra-se em constante crescimento, já bastante documentada e sobre a qual estão frequentemente a ser desenvolvidos estudos. É também fácil integrar a *Ethernet* com a *Internet* abrindo portas a novas funcionalidades. Neste sentido, a sua utilização ao nível de *fieldbus* em aplicações críticas de tempo real nos sistemas industriais tem vindo a ganhar força ao longo dos anos, existindo já diversos protocolos tempo-real baseados nesta tecnologia, conforme se verá mais adiante neste documento.

As redes *Ethernet* baseadas numa topologia em anel, através da utilização de *switches*, apresentam vantagens nomeadamente ao nível da redução da cablagem na ligação entre os diferentes nodos face à topologia em árvore usualmente utilizada. A capacidade de dotar os *fieldbuses* de maior velocidade de transmissão de dados e de maior largura de banda através da *Ethernet* é motivo para o interesse no desenvolvimento de tecnologias neste sentido.

As redes comutadas baseadas na tecnologia *Ethernet* conferem algumas características de tempo real sendo, no entanto, necessário acrescentar mecanismos de gestão destas redes para as tornar o mais eficientes possível. Devido a isto, os nodos na rede apresentam necessariamente uma arquitectura mais complexa. É, portanto, necessário existirem compromissos entre alguns factores, nomeadamente desempenho, determinismo e fiabilidade para que este paradigma seja aplicável na prática.

## 1.2 Objectivos

No âmbito desta dissertação, o objectivo principal é o desenvolvimento de uma infraestrutura baseada na tecnologia *Ethernet* que seja aplicável em sistemas embutidos distribuídos de tempo real. Neste sentido, foram estipulados os seguintes procedimentos que serviram como fio condutor para a conclusão do projecto:

- Investigação sobre *Ethernet*: partindo da sua origem e evolução histórica até algumas aplicações desta tecnologia actualmente;
- Definição da topologia de rede e arquitectura interna dos nodos;
- Estudo sobre o funcionamento da *Stack* Protocolar de Rede dos sistemas Linux;

- Desenvolvimento de um nodo *switch*, através da criação de um módulo ao nível do *kernel* dos sistemas Linux;
- Definição de um protocolo de gestão dinâmica da topologia da rede e da largura de banda em uso;
- Desenvolvimento de uma aplicação ao nível do utilizador para comunicação com o nodo desenvolvido;
- Teste do funcionamento e obtenção de resultados.

## 1.3 Organização da dissertação

Esta secção tem como objectivo apresentar a estrutura segundo a qual está organizada esta dissertação, fazendo um breve resumo dos conteúdos abordados em cada capítulo:

**Capítulo 2** apresenta uma perspectiva histórica da tecnologia *Ethernet* e alguns conceitos básicos sobre a mesma. É realizada uma análise sobre topologias das redes onde esta é frequentemente utilizada. São abordados conceitos básicos sobre redes comutadas *Ethernet* - *Switched Ethernet* - em geral.

**Capítulo 3** faz uma breve referência a algumas técnicas para utilização da tecnologia *Ethernet* em aplicações de tempo real. São descritos alguns protocolos de tempo real utilizados para *Ethernet* em meios partilhados sendo dado, no entanto, maior ênfase aos protocolos aplicados às redes *Switched Ethernet*.

**Capítulo 4** apresenta uma visão geral do sistema implementado. É descrita a topologia da rede e os mecanismos de gestão da mesma. Define-se neste capítulo a estrutura interna dos nodos e o protocolo de comunicação entre eles de modo a cumprir os requisitos temporais estipulados.

**Capítulo 5** descreve como foi realizada a implementação dos conceitos apresentados no Capítulo 4 na *stack* protocolar de rede dos sistemas Linux. É explicado como é possível inserir um módulo ao nível do *kernel* Linux, a maneira de manipular os pacotes que circulam na rede, nomeadamente a recepção e envio de pacotes, e ainda como foram implementados os mecanismo de gestão da rede.

**Capítulo 6** apresenta resultados experimentais da infraestrutura desenvolvida, no que diz respeito aos mecanismos de comutação de pacotes como também a gestão da rede e sua topologia, e uma análise sobre os mesmos.

**Capítulo 7** expõe a conclusão da dissertação, nomeadamente uma análise sobre contribuições científicas deste projecto, bem como possíveis *guidelines* para trabalho futuro.

# Capítulo 2

# Conceitos Básicos sobre *Ethernet*

A tecnologia *Ethernet* é utilizada em larga escala para a conexão de computadores, impressoras e servidores em redes locais. Foi desenvolvida por Robert Metcalfe, em meados de 1973, no centro de investigação *Xerox Pablo Alto* com o objectivo de fazer a ligação entre um computador e uma impressora a laser de alta velocidade [1][2].

Inicialmente esta tecnologia possibilitava a transmissão de informação a 2.94 Mbps mas devido ao seu rápido desenvolvimento permite hoje em dia troca de informação a ritmos na ordem dos 10 Gbps. A *Ethernet* é utilizada nas duas primeiras camadas do modelo *Open Systems Interconnection* (OSI), a camada física e a camada de ligação [2].

A *Ethernet* não foi originalmente concebida para ser aplicada em ambientes onde a necessidade de cumprimento de *deadlines* seja extremamente importante. Contudo este protocolo apresenta vantagens que podem resultar em benefícios em aplicações industriais onde a comunicação em tempo-real e robusta é essencial.

Ao longo deste capítulo irá ser feita uma breve caracterização da tecnologia e a descrição da sua evolução até aos dias de hoje, referindo as topologias de rede, as normas utilizadas e os mecanismos de acesso ao meio. Serão referidas também as motivações para o uso de *Ethernet* em aplicações críticas de tempo-real, dando ênfase às redes *Switched Ethernet*.

## 2.1 Origem e Evolução da *Ethernet*

A tecnologia *Ethernet* surgiu tendo por base outro padrão que foi desenvolvido por um grupo de investigadores, com principal referência a Norman Abramson, designado de ALOHAnet, com o objectivo interligar os campus da Universidade de Hawaii existentes nas várias ilhas. O modo como os vários dispositivos têm acesso ao canal transmissão neste padrão é aleatório, isto é, qualquer terminal pode enviar informação a qualquer momento. Assim que é enviado um pacote, o emissor espera por um sinal de validação - *acknowledge* - durante um tempo pré-definido, de modo a perceber se a sua mensagem foi enviada correctamente. Caso não receba nenhuma confirmação do sucesso no envio do pacote, o emissor assume a ocorrência de uma colisão e espera um tempo aleatório antes de tentar enviar novamente a informação [1][3].

Inspirado pelos estudos desenvolvidos sobre o padrão ALOHAnet, Robert Metcalfe desenvolveu o protocolo a que mais tarde se chamou de *Ethernet*, num projecto que visava a ligação de uma impressora a um computador pessoal. Esta tecnologia inicialmente utilizava cabo coaxial como meio de transmissão e operava a uma velocidade de 2.94 Mbps [2]. O controlo de

acesso ao meio é baseado no protocolo CSMA/CD. O *Carrier Sense Multiple Access* (CSMA) consiste na análise do estado do canal de comunicação antes de tentar enviar informações. O *Collision Detection* (CD) é a designação do mecanismo que detecta colisões de informação antes dos pacotes serem totalmente transmitidos. A capacidade descrita, que será aprofundada na subsecção 2.1.3, permitiu que a tecnologia *Ethernet* tivesse um funcionamento bastante mais eficiente que a ALOHAnet.

O bom desempenho da tecnologia bem como a facilidade de utilização, fez com que fosse utilizada em grande escala para interligação de redes locais. Desta maneira, com o objectivo de facilitar a investigação e venda de equipamentos para o uso desta tecnologia, o *Institute of Electrical and Electronic Engineers* (IEEE) padronizou a *Ethernet* como a norma 802.3 [2].

Nas subsecções seguintes irá ser feita uma descrição mais aprofundada da tecnologia.

### 2.1.1 Normas Ethernet

Após o surgimento da *Ethernet* rapidamente esta tecnologia foi evoluindo nomeadamente na velocidade de transmissão e utilizada em diversos meios físicos. A tabela seguinte apresenta algumas das principais expansões que foram sendo acrescentadas à tecnologia:

| Designação | Taxa de Transmissão | Meio Físico |
|---|---|---|
| 10Base5 | 10Mbps | Cabo coaxial grosso |
| 10Base2 | 10 Mbps | Cabo coaxial fino |
| 10BaseT | 10 Mbps | Pares de cabo de cobre |
| 100BaseT (FastEthernet) | 100 Mbps | Pares de cabo de cobre |
| 1000BaseT (GigabitEthernet) | 10000 Mbs | Pares de cabo de cobre |

Tabela 2.1: Algumas expansões da norma *Ethernet*

As evoluções à norma foram sendo designadas segundo um padrão *XXBaseYY* ou *XXBroadYY*, conforme se pode constatar na tabela 2.1. O valor *XX* representa a velocidade de transmissão da tecnologia em Mbps (Mega bits por segundo) e a designação Base ou *Broad* é utilizada consoante a frequência de funcionamento esteja na banda base ou em banda larga respectivamente. Em relação a *YY* caso seja um valor, indica o comprimento máximo de meio físico para transmissão em centenas de metros, ou caso seja uma letra, representa o meio físico [4].

#### 10Base5

Este é o padrão *da Ethernet*, definido na norma 802.3 [5], desenvolvido em meados de 1983. Foi a primeira versão comercializável desta tecnologia [6]. O canal de comunicação entre dispositivos é o cabo coaxial grosso com tamanho máximo de 500m e opera a uma taxa de 10 Mbps.

#### 10Base2

Está definido na extensão 802.3a [5] de 1985 apresentando uma velocidade de transmissão de 10 Mps, usando como meio físico o cabo coaxial com extensão máxima de aproximadamente 200m (186 m). Conhecida também por *Cheapernet*, foi apresentada como uma tecnologia de

fácil instalação e com custo reduzido. Está preparada para a utilização de repetidores na rede de modo a regenerar os sinais e assim obter-se um melhor desempenho [6][7].

#### 10BaseT

Esta norma, definida em 1990 como 802.3i [5], utiliza como meio físico de comunicação dois pares de cabos de cobre - *Unshielded Twisted-pair* (UTP) - de comprimento máximo de 100m, que permite uma velocidade de transmissão de 10Mbps. O facto do comprimento dos segmentos UTP ser reduzido, é comum a utilização de *hubs*[1] para regenerar os sinais e deste modo alongar o comprimento das redes [6].

#### 100BaseT

Conhecida também como *FastEthernet*, esta norma foi definida em 1995 na extensão 802.3u [5]. A velocidade de transmissão de informação é de 100 Mbps e utiliza o mesmo meio de transmissão que a norma 10BaseT. Simultaneamente foi surgindo o conceito de *switching* e redes comutadas, despoletando o uso de comunicação em *full duplex*, isto é, comunicação em simultâneo entre dois terminais, o que com o uso do padrão *FastEthernet* proporcionou comunicações mais rápidas [6].

#### 1000BaseT

Este *standard* é utilizado nas comunicações de alta velocidade de transmissão de dados. Padronizado na norma 802.3ab [5] em meados de 1997, apresenta compatibilidade com as normas anteriores, e deste modo existe um custo reduzido na migração para este standard. Operando a 1Gbps, tem a capacidade de trabalhar em *full duplex* e *half duplex*, sendo necessário alguns ajustes para suportar esta última funcionalidade [2].

### 2.1.2 Formato das Tramas

Na tecnologia *Ethernet* a informação é encapsulada em pacotes bem estruturados, designados de tramas. Existem fundamentalmente duas estruturas mais comuns para as tramas *Ethernet*: o formato IEEE 802.3 e*Ethernet* II. As figuras 2.1 e 2.2 ilustram os campos dos formatos referidos.

- **802.3**

| Preâmbulo | SFD | DA | SA | Tamanho | LLC | Dados | FCS |
|---|---|---|---|---|---|---|---|
| 7 bytes | 1 byte | 6 bytes | 6 bytes | 2 bytes | 3 bytes | 46-1497 bytes | 4 bytes |

Figura 2.1: Trama 802.3

[1]Hub - dispositivo com várias portas com a capacidade de regeneração de sinais e repetição em todas as suas portas.

- ***Ethernet* II**

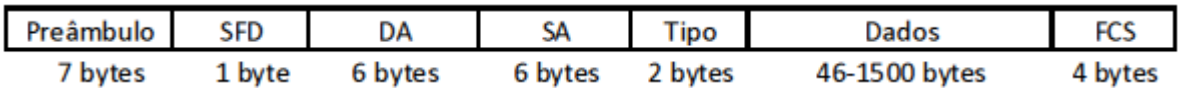



Figura 2.2: Trama *Ethernet* II

O Preâmbulo é constituído por 7 bytes com o padrão 10101010. Este campo é importante pois sinaliza a transmissão de uma nova trama, permitindo a sincronização entre o transmissor e o receptor. É enviado de seguida 1 *byte Start of Frame Delimiter* (SFD) que sinaliza o início da trama propriamente dita, com a sequência 10101011 [3].

O endereço de destino e endereço de origem, representados nas figuras 2.1 e 2.2 pelos campos DA (*Destination Adress*) e SA (*Source Adress*), são os endereços físicos dos terminais da rede. São endereços cujo tamanho é 6 *bytes*, único para cada dispositivo sendo parcialmente atribuídos pelo fabricante [3].

Na trama IEEE 802.3 existem 2 *bytes* que indicam o tamanho do campo de dados e 3 *bytes* para o *Logical Link Control* (LLC). O LLC é uma subcamada da camada de ligação presente no modelo OSI que é responsável pela comunicação entre a camada *Medium Access Control* (MAC) e as camadas superiores do mesmo modelo. Engloba 1 *byte* para identificar o serviço na estação de destino (DSAP), 1 *byte* para identificar o serviço na estação de origem da trama e 1 *byte* de controlo de fluxo. O campo Tipo, na trama *Ethernet* II indica o protocolo de nível superior que é transportado na trama [7].

O campo Dados de ambas as tramas é onde está a informação que se quer transmitir, o *payload*, que pode variar entre 46 a 1500 *bytes*. Apresenta um tamanho mínimo de 46 *bytes*, definido pelo *standard* 802.3, de modo a permitir, em caso de colisão, que todas as estações percebam a trama como errada [3]. Quando a informação que se pretende transmitir é inferior aos 46 *bytes* mínimos requeridos, são acrescentados ao campo de dados *bits* de *padding* até completar o valor mínimo necessário [3]. De notar que nas tramas 802.3 o campo LLC, que são *3 bytes*, fazem parte do *payload* pelo que, apenas podem ser usados 1497 *bytes* para a informação que se pretender enviar.

Por último, o *Frame Check Sequence* (FCS) representado pelos 4 últimos *bytes* de ambas as tramas, permite a verificação de anomalias nas mesmas. Faz uso de um polinómio que utiliza todos os bits da trama para calcular o seu valor usando um algoritmo de *Cyclic Redundancy Check* (CRC) [3]. Caso o FCS seja incorrecto a trama é considerada inválida, considerando-se que houve perda ou alteração da informação inicial.

### 2.1.3 Mecanismo de Acesso ao Meio: CSMA/CD

Numa rede onde o meio de comunicação é partilhado por vários equipamentos é necessário algum tipo de mecanismo que faça o controlo do acesso ao canal de comunicação. A norma *Ethernet* utiliza o mecanismo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*). Neste protocolo os dispositivos têm total acesso à rede em qualquer ocasião e com igual privilégio, isto é, podem tentar aceder ao meio de comunicação sempre que precisarem (*Multiple Access*) [8].

Quando um terminal quer aceder ao meio de comunicação para enviar informação necessita primeiramente de escutar se este se encontra disponível para ser utilizado (*Carrier Sense*).

Existem no entanto três variações no protocolo CSMA que podem ser utilizadas, descritas em seguida.

- **1-persistente CSMA**

Neste tipo de CSMA, o terminal vai estar constantemente a analisar o canal de comunicação até o encontrar livre e passível de ser utilizado. Quando esta situação ocorre o terminal vai enviar a sua trama imediatamente com uma probabilidade de 1 [3]. Esta variação é a que está definida na tecnologia *Ethernet*.

- **p-persistente CSMA**

Nesta variação do protocolo de escuta, quando o dispositivo detectar que o canal está livre e passível para ser utilizado, vai enviar a sua trama com uma probabilidade de p e atrasar o envio desta durante um certo intervalo de tempo com uma probabilidade 1-p [3].

- **não-persistente CSMA**

Nos protocolos não persistentes, os terminais não estão constantemente a verificar o estado do canal de comunicação. Pontualmente verificam se o canal está livre, e em caso afirmativo, começam a transmitir a sua informação. Caso o canal esteja ocupado, esperam um intervalo de tempo aleatório e depois voltam a analisar o estado do meio de comunicação [3].

Devido ao facto das comunicações não serem instantâneas, vários dispositivos podem detectar o canal de comunicação livre e iniciarem em simultâneo a transmissão de informação, ocorrendo portanto colisões. Quando os terminais estão a enviar informação, continuam a escutar os dados que circulam no meio, e quando um dispositivo detecta que a informação no canal de comunicação não é igual à que está a transmitir, a colisão é detectada (*Collision Detection*). Neste momento a estação emissora pára de transmitir a sua trama, envia um sinal *jam* para informar as restantes estações da rede que ocorreu uma colisão e espera um tempo aleatório definido por um algoritmo de recuo binário truncado até voltar a tentar retransmitir [7][1]. Em caso de ocorrerem 16 colisões consecutivas para a mesma trama, o respectivo terminal aborda a sua transmissão. Quando a transmissão é realizada com sucesso é esperado um intervalo de tempo, *Inter Frame Spacing* (IFS), de modo a que os terminais estejam prontos para receber novas tramas [9]. A figura 2.3 pretende ilustrar o funcionamento do CSMA/CD.

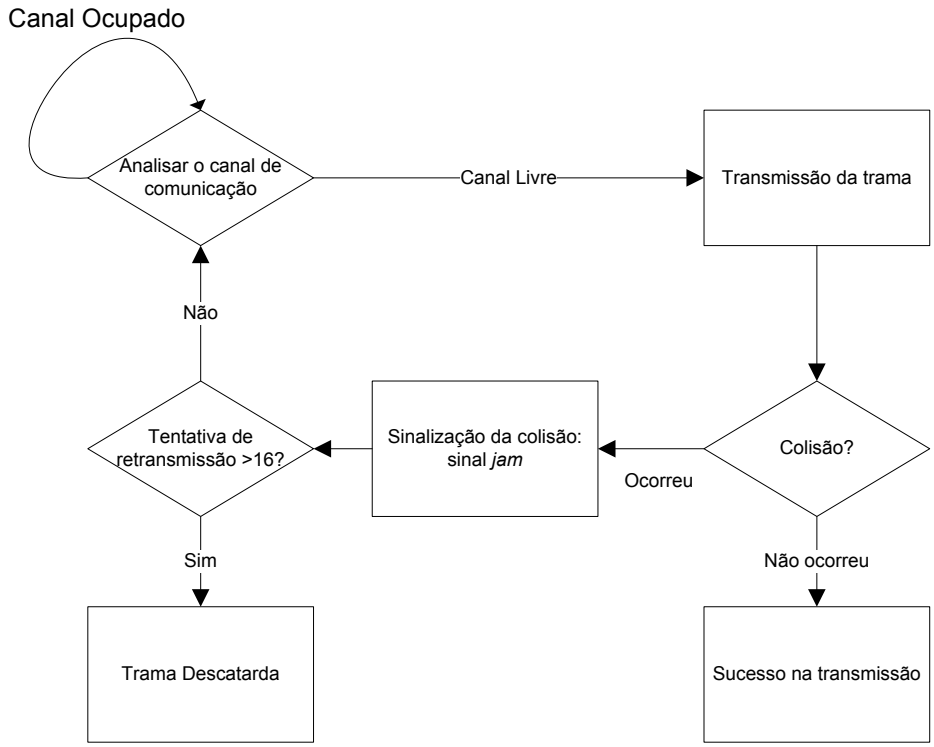



Figura 2.3: CSMA/CD

## 2.2 Topologia de Rede

Uma rede consiste num conjunto de terminais (nodos) interligados entre si, que trocam informação e partilham recursos. Dependendo das funcionalidades da rede e dos requisitos para a mesma, estes terminais podem estar ligados de diferentes maneiras. Existem quatro topologias básicas para a ligação dos nós na rede: Barramento (*bus*), Anel (*ring*), Estrela (*star*), Árvore (*tree*) e Malha (*mesh*).

### 2.2.1 Barramento

A figura 2.4 ilustra uma topologia de rede em barramento. Todos os terminais estão ligados ao mesmo meio de comunicação e a informação que, do ponto de vista lógico, circula no barramento está disponível para todos os dispositivos, sendo no entanto entregue às camadas superiores nos dispositivos à qual é dirigida. Na tecnologia *Ethernet*, como foi referido no capítulo 2.1.3, utiliza o método CSMA/CD para controlar o acesso ao meio de comunicação. Outro método, especificado na norma IEEE 802.4 - *Token Bus*, recorre a um *token* que passa de dispositivo em dispositivo, dando acesso ao canal que o possuir [10]. É uma topologia que utiliza menos cablagem comparativamente às outras referidas, possibilita uma fácil inserção de novos nós na rede e a extensão da mesma. No entanto, um problema óbvio é a perda do *token* e a sua recuperação, o que torna a rede pouco robusta. Em caso de falha do meio de comunicação toda a rede fica inoperável [8]. Necessita também de terminadores no final do barramento de modo a evitar reflexões que prejudicam o desempenho da rede.

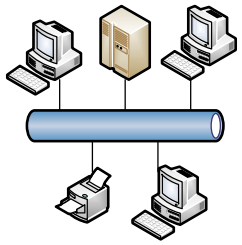


Figura 2.4: Topologia de Rede - Barramento

### 2.2.2 Anel

Na topologia em Anel os nós estão interligados num caminho fechado não existindo um nó central. A informação circula de terminal em terminal, e só é lida pelo nó à qual é destinada. Para comunicar, os dispositivos têm necessidade de requisitar um *token*, e só depois de o terem na sua posse é que podem enviar a informação para a rede. Este mecanismo ajuda a diminuir as colisões na rede e assim melhorar o seu desempenho [10]. Trata-se também de uma topologia de fácil análise e implementação. É difícil encontrar eventuais falhas na rede e caso um terminal fique inoperável toda a rede falha [10]. Uma solução para este problema é a existência de um duplo anel, para que em caso de falha da rede principal, continue a existir um caminho alternativo para a informação circular. A figura 2.5 ilustra este tipo de topologia.

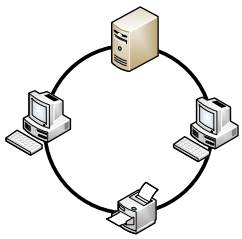


Figura 2.5: Topologia de Rede - Anel

### 2.2.3 Estrela

Numa topologia de rede em Estrela, figura 2.6, todos os terminais comunicam com um nó central. Esse nó central que pode ser um *hub* ou *switch* é responsável por receber a informação e encaminhá-la para o ponto da rede a que é destinada [10]. Caso um terminal da rede falhe é fácil detectar a sua localização e a restante rede continua a funcionar normalmente. Modificar e acrescentar nós à rede é relativamente fácil pois todos comunicam com o mesmo ponto. Apresenta uma gestão da rede centralizada. No entanto, caso o nó central falhe toda a rede fica sem funcionar. Devido à necessidade de todos os nós comunicarem com o nó central existe um maior número de ligações e portanto mais cablagem [10].

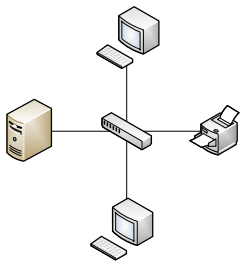


Figura 2.6: Topologia de Rede - Estrela

### 2.2.4 Árvore

Na topologia em árvore, o princípio de funcionamento é semelhante ao da topologia em estrela. Existe um nó central responsável por controlar toda a rede. Os nós vão sendo acrescentados à rede, criando ramificações, conforme é ilustrado na figura 2.7. Uma falha de um nó não prejudica o funcionamento dos nós existentes ao mesmo nível da hierarquia, apenas os que estão ligados a esse nó. Em caso de falha do nó central toda a rede fica inoperável. Sistemas baseados nesta topologia podem crescer em qualquer direcção sendo necessário ter sempre o cuidado de não se formarem *loops* na rede uma vez que degrada o funcionamento da mesma [11]. Usualmente são inseridos repetidores entre segmentos de modo a regenerar a informação que circula na rede.

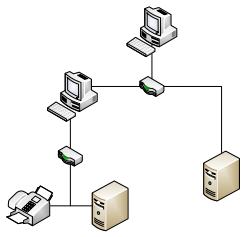


Figura 2.7: Topologia de Rede - Árvore

### 2.2.5 Malha

Numa rede organizada em Malha todos os terminais comunicam entre si. Um dos grandes problemas deste tipo de topologia é que cada vez que é necessário integrar um novo terminal, o número de ligações aumenta exponencialmente, o que para redes de grandes dimensões implica uma maior complexidade na gestão dessas redes [10]. Portanto, o elevado número de cabos para efectuar as ligações é a grande desvantagem desta topologia. Apresenta também um custo acrescido de implementação relativamente às topologias atrás referidas. No entanto, como todos os nós comunicam entre si, existe grande robustez e tolerância a falhas na rede, existindo diversos caminhos alternativos para trocar informação. A figura 2.8 ilustra uma rede em Malha com quatro terminais.

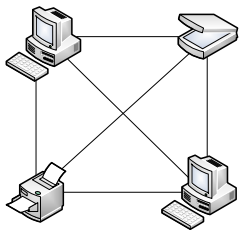


Figura 2.8: Topologia de Rede -Malha

### 2.2.6 Aplicações e Topologias Empregues

Usualmente, as redes baseadas na tecnologia *Ethernet* são configuradas em estrela. No entanto em sistemas industriais, de acordo com [12], normalmente são utilizadas topologias em anel. No mesmo *white paper* [12] é mostrado um sistema de vigilância distribuído de um edifício baseado numa topologia em anel. As câmaras de vigilância são ligadas a um *switch* de 3 portas da *Micrel* e os *switchs* ligados entre si formando o anel. Uma das vantagens já referidas deste tipo de topologia é o reduzido número de cabos utilizados nas ligações, como referido anteriormente, levando a um menor custo na sua implementação. A figura 2.9 ilustra esta rede.

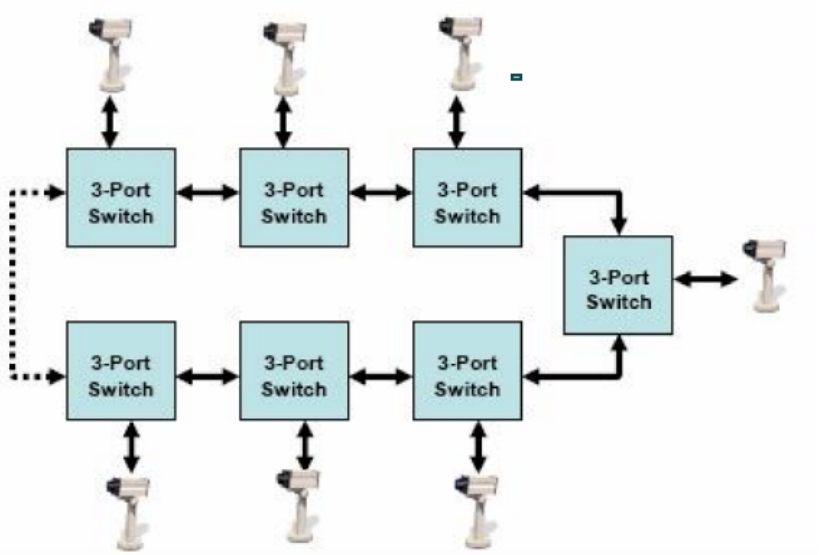



Figura 2.9: Exemplo da rede referida no *white paper* [12]

Outra das áreas onde se começou a introduzir a tecnologia *Ethernet* foi na indústria automóvel. Em [12] é dado um exemplo de uma rede baseada na fusão das topologias em anel e estrela, como ilustrado na figura 2.10. Basicamente a rede é constituída por um *gateway* que faz a ligação entre o interface de diagnósticos do automóvel (*On-Board Diagnostics* (OBD)) e outras redes existentes. Esse *gateway* liga também a uma rede *Ethernet* em anel, constituída pelo computador de bordo do automóvel, pelo sistema de navegação e pelo sistema de áudio, por exemplo [12]. A introdução de redes *Ethernet* permite a fácil ligação à Internet através

de módulos de comunicação sem fios e deste modo realizar várias aplicações, como *download* de músicas ou mapas para os sistemas de navegação.

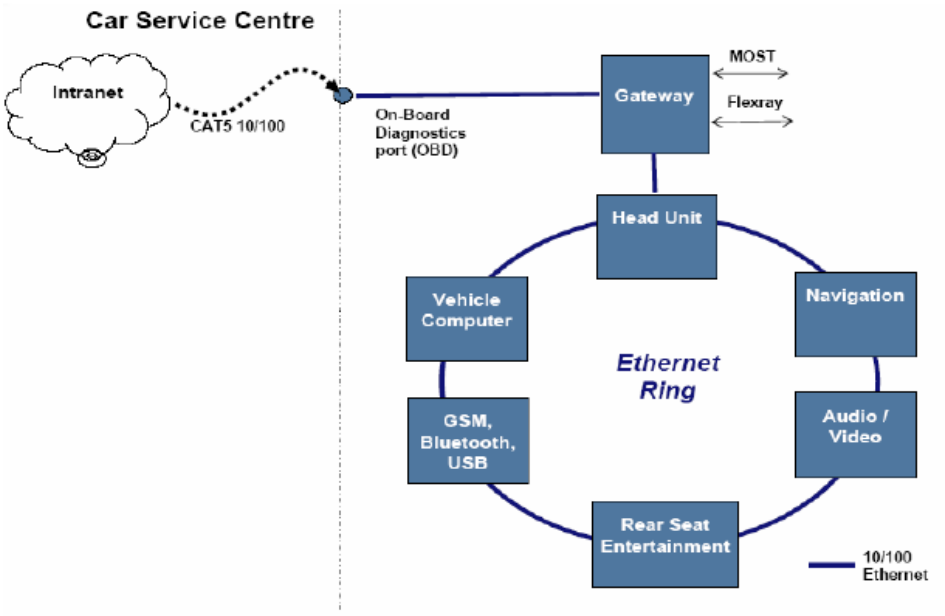



Figura 2.10: Exemplo de uma rede *Ethernet* em automóveis referida no *white paper* [12]

## 2.3 *Switched Ethernet*

Com o passar dos anos a utilização de *switches Ethernet* tem vindo a ganhar grande importância. O facto de existir apenas um domínio de colisão por cada porta do *switch* contribui para melhorar o determinismo da rede, pois o mecanismo CSMA/CD não é normalmente activado devido à inexistência de contenção. Assim possibilita-se a sua aplicação em sistemas com imposições temporais rigorosas. Os princípios de funcionamento de redes baseadas nesta filosofia foram influenciados pela tecnologia usada nos telefones [8]. Outras das potencialidades dos *switches* são o seu elevado *throughput* e a sua capacidade de isolar tráfego. Apresentam também grandes velocidades de transmissão e latência reduzida, o que é importante em aplicações que requerem algum tipo de controlo sobre o sucesso na transmissão da informação [8]. Actualmente as redes baseadas em *Switched Ethernet* estão a ser bastante utilizadas em vários domínios industriais, como automação industrial, onde é necessário o cumprimento *deadlines* e a capacidade de tolerância a falhas.

Em seguida será feita uma abordagem sobre os conceitos fundamentais relativos a *Switched Ethernet* bem como dos protocolos de gestão de rede onde esta tecnologia é utilizada.

### 2.3.1 Funcionamento do *switch*

Um *switch* é um dispositivo com várias portas que interliga pontos de uma rede, operando portanto na camada de ligação do modelo OSI. Cada dispositivo tem a si associado um endereço MAC que o identifica. A funcionalidade de um *switch* é reencaminhar uma trama que

lhe chega a uma das portas para o respectivo destino. Estes dispositivos têm uma tabela de encaminhamento que contem informação dos endereços MACs dos equipamentos ligados às suas portas [13]. Na chegada de uma trama, o *switch* tem capacidade de guardar a porta pela qual chegou e o endereço MAC do equipamento de origem na sua tabela de encaminhamento, ficando desta maneira com a noção da localização dos dispositivos existentes na rede. O passo seguinte, após um pacote ter chegado ao *switch*, é verificar se o endereço MAC do destino se encontra na sua tabela, e enviar para o segmento da rede indicado através dos mecanismos descritos em seguida [8][14].

### Mecanismo de Flooding

Se uma trama chegar ao *switch* e o endereço MAC do destino não se encontra associado a nenhuma das suas portas, o procedimento é reencaminhar o pacote para todas elas excepto para a porta pela qual chegou. Este mecanismo é designado de *flooding*. A desvantagem deste processo é o envio de pacotes para locais da rede onde não é destinado, ocupando largura de banda e diminuindo assim desempenho da rede. Normalmente, este processo ocorre no início da operação de uma rede onde as tabelas de encaminhamento não estão preenchidas. A figura 2.11 representa esse mecanismo.

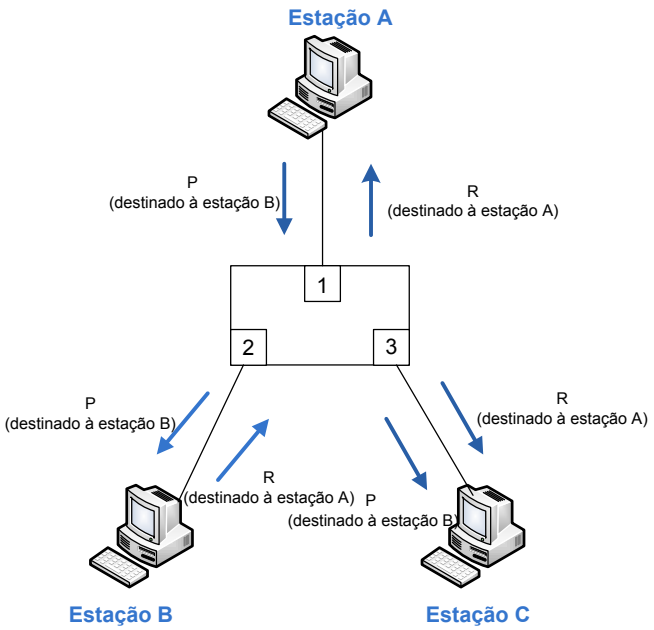



Figura 2.11: *Flooding* de pacotes

### Mecanismo de Forwarding

A figura 2.12 pretende ilustrar o mecanismo de *forwarding*. Caso esteja presente na tabela de encaminhamento, o *switch* vai reencaminhar a trama apenas para a porta que permite aceder ao destino pretendido. Este mecanismo apresenta como grande benefício o facto do pacote ser directamente encaminhado ao destino, aumentando assim o desempenho da rede. A imagem que se segue descreve de modo simples o funcionamento deste mecanismo: a estação A envia um pacote P destinado à estação B. Chegando ao *switch*, se existe na tabela de encaminhamento deste informação para chegar ao destino, é necessário apenas reencaminhar o pacote pela sua porta 2, e deste modo o pacote é entregue. O mesmo raciocínio é aplicado ao pacote R da resposta dada pela estação B à estação A. De notar que a estação C está fora da troca de mensagens porque nada lhe é destinado.

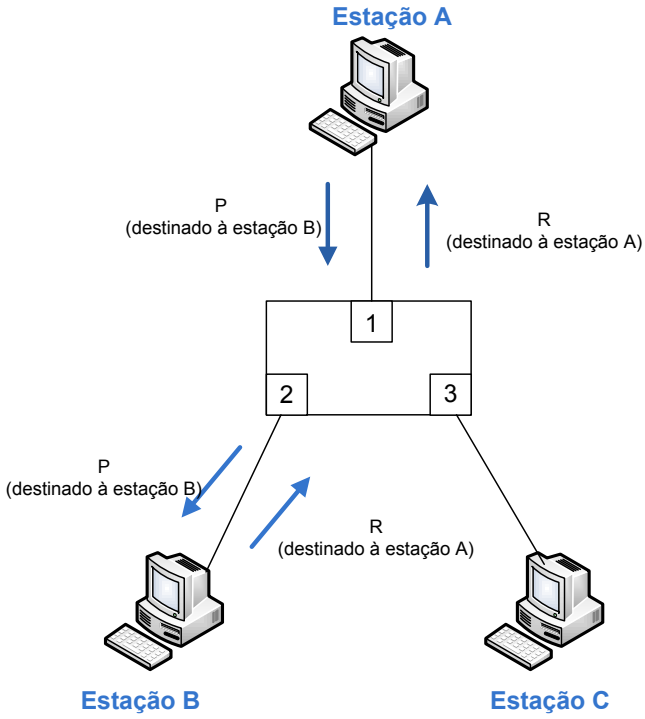



Figura 2.12: *Forwarding* de pacotes

### Tempo de *Aging*

Existe outra funcionalidade dos *switches*, designada por *Aging*, que é responsável por permitir a existência na tabela de encaminhamento de apenas equipamentos activos na rede. Quando o *switch* aprende um endereço de um dispositivo um contador é activado e cada vez que for enviada alguma trama do respectivo nodo este é reiniciado. Após um tempo pré-definido, caso não tenha sido recebida qualquer informação do dispositivo este é retirado da tabela de encaminhamento [14].

***Cut-through Switches***

A velocidade de reenvio de informação, quando esta chega ao *switch*, varia consoante o mecanismo de *switching* que esteja implementado. Uma trama pode ser enviada mal seja possível verificar o endereço de destino na sua chegada, mecanismo designado de *cut-through*. Desta maneira é possível reencaminhar o pacote para o destino rapidamente e diminuindo a latência entre as portas de recepção e transmissão, visto que a trama não é guardada na sua totalidade [8][14]. Este processo tem o inconveniente de não verificar a existência de erros na trama e deste modo enviar para a rede dados desnecessários prejudicando o seu desempenho. De forma a tentar minimizar este impacto alguns *switches* verificam a integridade da trama e, caso seja detectada uma elevada quantidade de erros, este bloqueia a porta pela qual está a enviar ou então muda o mecanismo para *store-and-forward*, que será referido de seguida. Um *switch* com esta capacidade de alterar o seu funcionamento entre *store-and-forward* e *cut-through*. designa-se de *Adaptative Switchs* [8][2][4]. A figura 2.13 ilustra este mecanismo.

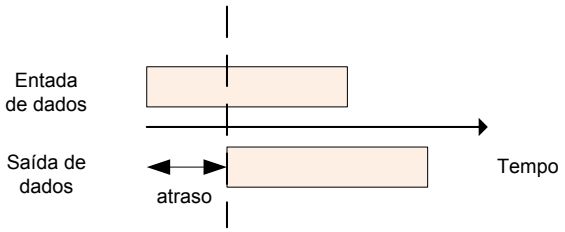



Figura 2.13: Mecanismo *Cut-through*

***Store-and-forward Switches***

Um processo que garante uma maior fidelidade na transmissão da informação é designado de *store-and-forward*, em que o *switch* espera pela recepção da totalidade da *frame* antes de a reencaminhar para o destino. Consequentemente a troca de informação tem mais latência mas existe um maior controlo de erros conseguindo-se, deste modo, optimizar a largura de banda. A figura 2.14 ilustra o mecanismo descrito.

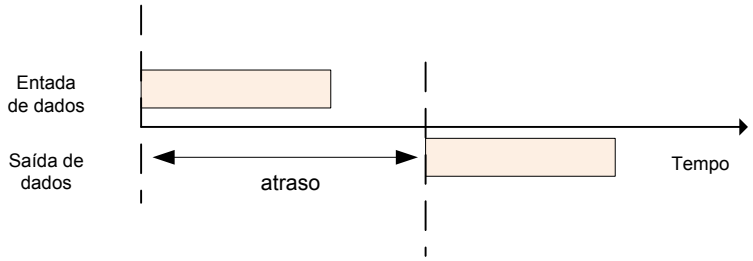



Figura 2.14: Mecanismo *Store-and-forward*

### 2.3.2 *Hardware switching vs Software switching*

Existem duas abordagens principais no projecto de um *switch*: *software switching* e *hardware switching*. Numa abordagem por *software* o desempenho do *switch* depende de alguns

factores como o número de portas por microprocessador, a frequência a que os dados são processados e a qualidade do código de programação na avaliação do seu desempenho. Em comparação com a abordagem por *hardware*, apresenta menor taxa de transmissão de dados tendo no entanto a grande vantagem de permitir realizar actualizações no *switch* através de alterações a nível de *software* [8]. Usualmente *software switching* é utilizado quando o processamento da informação não pode ser realizado ao nível do *hardware*.

Outra abordagem na projecção de um *switch* é via *hardware*, desenvolvido em circuitos integrados designados de *Application Specific Integrated Circuits* (ASIC). A grande vantagem deste dispositivo é a capacidade de transmitir informação a grandes velocidades nas redes. No entanto, a necessidade de alterações no dispositivo implica a produção de um novo circuito integrado. Geralmente são gerados num processo que apresenta duas etapas: a primeira é a sua implementação atrás de uma *Field Programmable Gate Array* (FPGA), isto é, um dispositivo que permite a geração de *hardware* reconfigurável e desde modo optimizar o seu funcionamento, para depois numa segunda etapa ser implementado num ASICs. A sua produção em massa torna estes dispositivos baratos e bastante aliciantes para o mercado [8].

### 2.3.3 Protocolo *Spanning Tree*

A qualidade de uma rede de telecomunicações é avaliada, entre outros parâmetros, pela sua capacidade de configuração automática para recuperar de situações anómalas de modo autónomo. É portanto necessário, caso exista avarias em pontos da rede, que esta possua a redundância suficiente para se adaptar à nova situação e arranjar um caminho alternativo para trocar informação, e portanto, manter a rede activa.

Tendo em vista proporcionar a uma rede a capacidade de recuperar de falhas, foi desenvolvido o Protocolo *Spanning Tree*, mecanismo especificado na norma 802.1D do IEEE. Na análise de funcionamento seguinte, irá usar-se o termo *bridge* em vez de *switch* para manter coerência com a bibliografia pesquisada.

#### Funcionamento

Considere a título exemplificativo uma rede simples como a representada na figura 2.15 constituída por três *bridges* a interligar três *Local Area Network*s (LANs). É necessário configurar a rede tendo em vista a optimização do seu desempenho. O *Spanning Tree Protocol* (STP) foi desenvolvido com o objectivo de resolver esta problemática, tentando organizar a rede definindo um caminho activo único entre dois quaisquer nós, adicionando alguma robustez à rede para que em caso de falha de algum caminho seja capaz de se re-organizar novamente.

Inicialmente é necessário definir a *bridge* que ficará como o primeiro nó da rede, a *bridge* raíz. Esta *bridge*, que será o início da *spanning tree*, pode ser definida da seguinte maneira: o administrador de uma rede pode escolher qual a *bridge* que pretende para ser o primeiro nó, bastando configurar os dois *bytes* mais significativos do *bridge* ID (identificador da *bridge*: 2 *bytes* de configuração de prioridade + 6 *bytes* do endereço MAC da *bridge*) ou então a *bridge* escolhida é a que tiver menor endereço MAC de todas as que estão na rede [7].

Na parte inicial da configuração da rede todas as *bridges* assumem ser a raíz e enviam para todas as suas portas mensagens de configuração com o seu endereço, *Bridge Protocol Data Unit* (BPDU). Cada *bridge* irá comparar as mensagens que recebem com o seu endereço, irão verificar qual é o endereço mais baixo e considerar que essa é a *bridge* raiz. No final, quando a rede convergir, a *bridge* raíz será reconhecida por todas as outras *bridges* [7][15]. As restantes

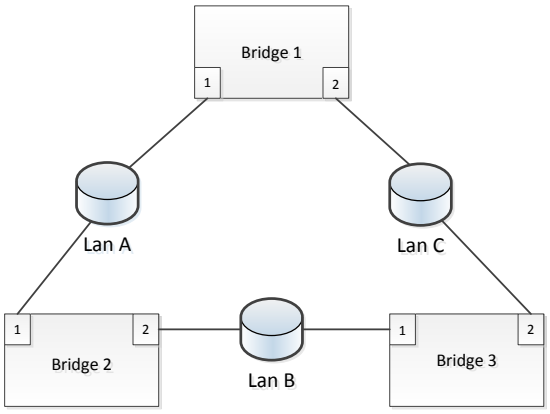



Figura 2.15: Exemplo de uma rede a configurar

*bridges* irão definir qual o caminho mais curto para a *bridge* raíz usando um algoritmo de Bellaman-Ford assíncrono e distribuído [7]. Em caso de existirem vários caminhos mínimos para a raíz é necessário arranjar um processo de modo a escolher apenas um. Todas as *bridges* têm definido um custo do caminho a percorrer para a raíz, e portanto, o primeiro critério de desempate é a que tiver um custo menor. Caso o custo para a raíz seja igual em ambas as *bridges*, o critério seguinte a ser verificado é o *bridge* ID (identificador da *bridge*) sendo seleccionada a que tiver menor valor. Se os critérios atrás referidos forem idênticos, o caminho mínimo para a raíz irá ser definido pelo menor *port* ID (identificador da porta) [7].

Existem conceitos que são fundamentais na definição do protocolo *spanning tree*. As portas raízes existentes nas *bridges* são responsáveis pela comunicação com a *bridge* raíz. As portas designadas são aquelas que numa LAN são responsáveis pela troca de informação com *bridge* raíz. As restantes portas que não se enquadram em nenhuma designação anterior são bloqueadas [7]. Segundo este protocolo as portas capazes de receber e processar dados ou mensagens de configuração estão num estado designado de *forwarding* - caso das portas raízes e designadas. As portas bloqueadas que não estão habilitadas a processar pacotes de dados, recebem e processam mensagens de configuração - estado *blocking*. Quando existe necessidades de reconfigurar a rede por algum motivo as portas bloqueadas passam para o estado *listening*, esperando um determinado tempo, designado de *forward delay*. Durante esse tempo, caso receba uma mensagem de configuração, BPDU, esta porta passa para o estado *learning*, caso contrário é bloqueada. No estado *learning* a porta está activa no processo de definição do protocolo *spanning tree* podendo passar para o estado *forwarding* ou *blocking*. Existe ainda outro estado definido neste protocolo, o estado *disable*, no qual a porta não participa no desenvolvimento do algoritmo [15].

A figura 2.16 ilustra um exemplo simples de uma rede com três *bridges* em que o STP convergiu. Assumiu-se que o custo para a raíz é igual em ambos os percursos. Analisando a imagem podemos verificar que a *bridge* 1 apresenta menor *bridge* ID que as restantes, logo, após a troca das mensagens de configuração a rede vai assumi-la como *bridge* raíz. Do ponto de vista da LAN A e C, as portas que permitem a comunicação com a *bridge* 1 (raíz) são a 1 e 2 respectivamente e portanto são as portas designadas dessas LANs. As portas que das *brigdes* 2 e 3 permitem a troca de informação com a *brigde* raíz são a 1 e 2 respectivamente pelo que são nomeadas de portas raíz. Das duas portas que estão ligadas à LAN C, apenas

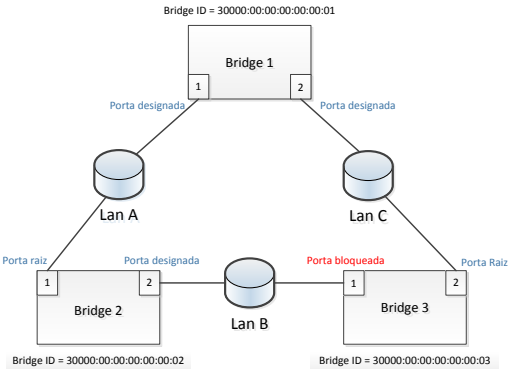



Figura 2.16: Protocolo *Spanning Tree*

uma pode ser designada, e como assumimos que o custo para a raíz é igual é necessário ir analisar o *brigde* ID das *brigdes* associadas a esta LAN. Deste modo, a *bridge* 2 apresenta um menor *bridge* ID pelo que a sua porta 2 é designada para a LAN C. A porta 1 da *brigde* 3 é bloqueada, e desde modo o STP convergiu.

### 2.3.4 *Virtual Local Area Networks* - VLANs

Os *switches* têm a possibilidade de gerar redes virtuais independentes dentro de uma rede. Inicialmente o grande objectivo das *Virtual Local Area Network*s (VLANs) era diminuir o domínio de colisões de uma rede Ethernet aumentando desta maneira o seu desempenho. Com o aparecimento dos *switches*, que apresentam vários domínios de colisão, a utilização de VLANs com o propósito inicial caiu em desuso. Apesar disso as VLANs nas redes com *switches* são utilizadas na tentativa de diminuir os domínios *broadcast*. Apresentam a grande vantagem de permitir a configuração de uma rede sem a necessidade de acrescentar ou alterar dispositivos na rede. Aliada a isso, este protocolo pode ser utilizado para fornecer segurança a uma determinada rede utilizando, por exemplo, uma rede virtual de modo a proteger informações importantes [16]. A utilização de VLAN está padronizada na norma 802.1Q do IEEE [16][7].

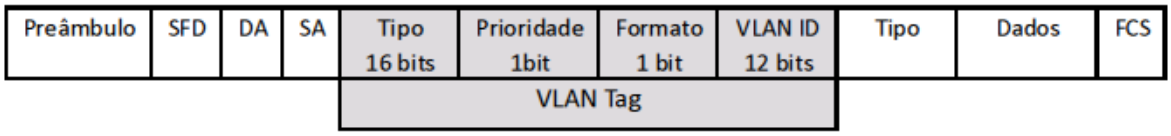



Figura 2.17: *tag* VLAN inseridas nas tramas

A figura 2.17 ilustra a *tag* inserida nas tramas *Ethernet* de acordo com o protocolo 802.1Q do IEEE.

Dentro do campo *Tag Vlan* existem 2 *bytes* que identificam o protocolo que está a ser utilizado. Caso seja a norma 802.1Q este campo tem o valor 0x8100.

O campo *Prioridade*, constituído por 3 *bits*, permite indicar a prioridade da informação inserida na trama.

De seguida existe 1 *bit* que indica se a trama se encontra no formato canónico (*bit* a 1) ou não (*bit* a 0). Este campo permite verificar se a trama é compatível com a norma Token *Ring*.

O campo VLAN ID, de 12 *bits*, permite identificar a VLAN da respectiva trama. Podem estar definidos 4094 identificadores diferentes, pois os valores 0x000 e 0xFFF reservados.

### 2.3.5 *Auto - Negotiation*

Esta funcionalidade está definida na norma 802.3u que caracteriza a tecnologia *FastEthernet*. Permite que os dispositivos que operam as taxas de transmissão diferentes numa ligação *Ethernet* reconheçam e sincronizem a velocidade máxima de transmissão de informação de forma transparente para o utilizador. Em redes *Switched Ethernet* este processo é bastante vantajoso pois não seria prático um utilizador configurar cada porta dos diferentes *switches* de modo a que a comunicação fosse viável.

Para todas as tecnologias Ethernet que usam como meio físico os cabos de cobre esta funcionalidade está disponível. No entanto, nas comunicações ópticas apenas a tecnologia *Gigabit Ethernet* tem definido o mecanismo de *auto-negotiation* [17].

### 2.3.6 *Link Aggregation*

O *Link Aggregation* é uma funcionalidade mais recente dos *switchs* e está definida na norma IEEE 802.3ad [18]. Em termos simples o *Link Aggregation* permite agregar várias ligações físicas entre dispositivos em uma só ligação lógica de maior largura de banda. Uma vez que utilizada a capacidade de todas as ligações da rede permite aumentar o seu desempenho e a sua fiabilidade.

De acordo com [18] esta funcionalidade permite aumentar a disponibilidade e capacidade das ligações e acrescenta ainda alguma tolerância a falhas na rede. É um protocolo não intrusivo o que significa que é aplicável ao *hardware* existente.

## 2.4 Sumário

Este capítulo teve o objectivo de apresentar a tecnologia *Ethernet*. Foi abordada a sua perspectiva histórica, desde a origem até aos dias actuais. Apresentou-se as características desta tecnologia, nomeadamente a evolução da velocidade de transmissão de dados, o formato das tramas *standard* que são utilizadas e a política de acesso ao meio de comunicação, o CSMA/CD. Em seguida foram ilustradas algumas topologias de rede onde normalmente a *Ethernet* é utilizada e algumas aplicações já existentes. Por fim, focou-se um pouco as redes comutadas baseadas em *Ethernet* e algumas funcionalidades associadas como o STP e *Link Aggregation*. Até este ponto foram abordados os conceitos básicos sobre *Ethernet* de modo a perceber o funcionamento desta tecnologia e permitir uma transição suave para as secções seguintes.

# Capítulo 3

# Ethernet para aplicações de tempo-real

A tecnologia *Ethernet* não foi originalmente concebida para ser aplicada em ambientes onde a necessidade de cumprimento de *deadlines* é extremamente importante. O mecanismo de acesso ao meio desta tecnologia, o CSMA/CD, apresenta características não determinísticas que não são desejáveis em situações críticas. Contudo este protocolo apresenta inúmeras vantagens que podem gerar benefícios em aplicações industriais onde a comunicação em tempo-real é essencial. De acordo com [1], o facto da *Ethernet* ser barata, devido à sua produção em massa, aliada à rápida evolução da velocidade a que a informação circula na rede, são fortes atractivos para a utilização desta tecnologia em aplicações industriais. Outros factores são também a facilidade de integrar a Internet sobre *Ethernet* bem como outros protocolos de comunicação. Trata-se de uma tecnologia bastante actual e bem especificada, existente em grande parte dos equipamentos e portanto, sem grandes problemas de compatibilidade [1].

No entanto, como já foi referido, a tecnologia *Ethernet* apresenta um conjunto de características que por definição não são próprias para comunicações de tempo-real. Alguns factores como tempo de resposta, previsiblidade, uso eficiente de largura de banda e cumprimento de *deadlines* são fundamentais para aplicações críticas de tempo-real. Devido a isto foram desenvolvidos vários estudos de modo a tornar possível a aplicação da tecnologia *Ethernet* nessas situações.

## 3.1 Técnicas para usar *Ethernet* em aplicações críticas

Muito trabalho foi desenvolvido de modo a oferecer condições à tecnologia *Ethernet* para o seu uso em aplicações de tempo real. De modo a que a *Ethernet* tenha um comportamento temporal determinístico é necessário diminuir ou mesmo remover o número de colisões de pacotes ou pelo menos encontrar um mecanismo para controlar essas colisões com alguma previsibilidade. A utilização de *switches Ethernet* permite, como foi referido, diminuir o número de colisões e portanto acrescentar algum determinismo às redes. No entanto, o uso de *switches* não resolve todos os requisitos necessários para aplicações de tempo real.

Uma abordagem que permite dotar a *Ethernet* com um comportamento temporal previsível consiste na modificação do seu mecanismo de acesso ao meio. A grande desvantagem deste método é ser bastante intrusivo, isto é, com a modificação do MAC *standard* da *Ethernet* deixa de haver compatibilidade com equipamentos já existentes no mercado.

Um método menos intrusivo consiste em adicionar protocolos de controlo *sobre Ethernet* e deste modo possibilita-se o uso de equipamentos *standard* já existentes. Esta abordagem acrescenta no entanto alguma complexidade de implementação. Alguns protocolos desenvolvidos serão referidos de seguida.

Uma solução é apresentada pelo *Windows Protocol*. Nesta abordagem existe um intervalo de tempo acordado entre todos os nós da rede, designado por *window*, intervalo este que é sincronizado sempre que ocorre uma transmissão com sucesso [1]. As mensagens a transmitir são enviadas para o canal de comunicação e, caso exista mais que uma mensagem numa *window* ocorre uma colisão. Sempre que esta situação ocorre o tamanho da janela temporal vai sendo reduzido até isolar apenas uma mensagem a enviar. Quando a transmissão é realizada com sucesso, a *windows* fica livre para uma nova transmissão e o intervalo temporal desta permanece igual [1]. Em caso de não existirem mensagens no canal de comunicação o tamanho da janela vai aumentando até atingir o valor máximo. Novos nós podem ser acrescentados à rede, que podem apresentar uma janela temporal diferente da estabelecida, o que irá gerar um número maior de colisões até os nós ficarem novamente sincronizados.

Outra abordagem consiste no uso de uma interface sobre a camada *Ethernet* que irá gerir e controlar o tráfego que circula na rede. Essa interface, designada por *traffic smoother*, tem como objectivo limitar a velocidade de transmissão de cada nó da rede de modo a evitar o excesso de informação na mesma. Esta solução é designada de *Traffic Shaping*. A ideia consiste em fazer passar toda a informação que não apresenta requisitos temporais, e portanto sem necessidade de transmissão em tempo real, pelo *traffic smoother* de modo a ter uma gestão sobre a mesma e não prejudicar o desempenho de aplicações críticas. O tráfego que requer o cumprimento de requisitos temporais normalmente apresenta mecanismos próprios de controlo de fluxos pelo que não é gerido pelo *traffic smoother*. É portanto conferida à informação que requer transmissão em tempo-real maior prioridade que a restante, e o desempenho do sistema é melhorado pois a interferência de tráfego sem requisitos temporais na rede é limitada [1].

Atribuir a um nó na rede a capacidade de gestão da mesma, é uma solução para acrescentar algum determinismo e previsibildade às redes. Trata-se portanto de uma abordagem *master/slave* em que o nó *master* é responsável pelo controlo e atribuição do canal de comunicação aos restantes nós da rede, os *slaves*. As restrições temporais são então garantidas pelo nó central, *master*, que faz uso de algoritmos de escalonamento em tempo-real. Como toda a gestão é feita pelo nó central, circula na rede para além de informação que se pretende transmitir, mensagens de controlo geradas pelo *master*, o que pode levar a sobrecarga na rede. É portanto necessário maior largura de banda ou mecanismos para gerir o uso da mesma, com o objectivo de melhorar o desempenho da rede. Existem protocolos que utilizam este mecanismo de controlo entre os quais *Ethernet PowerLink* [19] e *Flexible Time-Triggered* sobre *Switched Ethernet* (FFT-SE) [1][20].

O *Token Passing* é um conceito para gestão do acesso ao canal de comunicação. É baseado na existência de um *token* que circula de nó em nó na rede. O dispositivo que tiver o *token* na sua posse pode transmitir a sua informação para o canal de comunicação. O *token* é libertado quando a informação é transmitida ou quando a janela temporal estabelecida é esgotada, passando em seguida para outro nó da rede. Como este mecanismo só possibilita o acesso ao canal de comunicação ao dispositivo que possuir o *token*, as colisões de pacotes no canal são evitadas.

Um exemplo de um protocolo baseado na ideia de *Token Passing* é o *RETHER* [21], onde o mecanismo de acesso ao meio é o CSMA/CD característico da tecnologia *Ethernet* para tráfego sem requisitos temporais restritos, alterando para o mecanismo de *token-bus* de modo

transparente na chegada de tráfego de tempo-real [1][21].

## 3.2 *Ethernet PowerLink*

*Ethernet PowerLink* (EPL) é um protocolo que confere à tecnologia *Ethernet* características de tempo real. Trata-se de um protocolo não intrusivo, o que significa que pode ser implementado em dispositivos *Ethernet standard* [19]. Foi inicialmente desenvolvido para a tecnologia *Ethernet* em meios partilhados oferecendo uma maior velocidade de transmissão e maior precisão, nomeadamente *jitter* na ordem de 1µs [22].

Neste protocolo a gestão da rede é baseada em *master-slave*, onde o *master* (também designado na bibliografia por *Managing Node*) implementa um mecanismo de *Time Division Multiple Access* (TDMA), controlando assim as janelas temporais, que permitem a cada dispositivo *slave* (também designados de *Controlled Nodes*) aceder ao canal de comunicação para transmitir informação. Deste modo as colisões de pacotes são evitadas, existindo determinismo e controlo da informação que circula na rede. Em cada ciclo de transmissão, o protocolo EPL apresenta mecanismos que suportam tráfego periódico e não periódico. A figura 3.1 pretende ilustrar a estrutura do ciclo de transmissão do protocolo EPL.

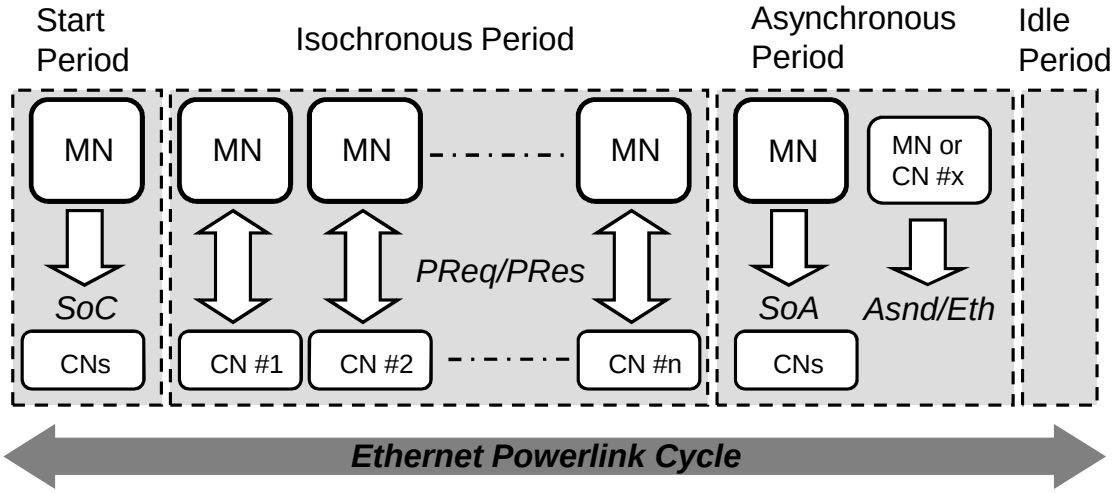



Figura 3.1: *Ethernet Powerlink* - estrutural da janela temporal [22]

Analisando a figura 3.1 , pode verificar-se que num ciclo EPL existem quatro fases distintas.

A primeira fase, *Start Period*, indica o início de uma nova janela temporal. O nó*Master* envia para todos os *Slaves* da rede uma mensagem de controlo *Cycle Start* (SoC) que indica o início de um novo ciclo e permite a sincronização entre os nós. A fase seguinte é a que diz respeito à troca de tráfego periódico na rede, *Isochronous Period*. O *Master* envia para cada *Slave* uma mensagem de*Pool Request*. Os *Slaves* respondem com uma mensagem *Pool Response* não só para o *Master* mas para todos os nós da rede, isto é, em *broadcast*. Quando o *master* receber a mensagem de *Pool Response*, espera um intervalo de tempo antes de comunicar com outro *slave*. As comunicações periódicas podem ser contínuas, isto é, repetidas

em todos os ciclos temporais como podem também ser multiplexadas no tempo e apenas serem realizadas a cada $n$ ciclos [22]. A fase da comunicação aperiódica é sinalizada pelo *Master* através do envio de uma mensagem *Start of Acyclic* (SoA) para todos os nós da rede. Neste período apenas é concedido o direito de comunicação a um nó: o *Master* envia uma mensagem SoA para o respectivo *Slave* que responde com a mensagem assíncrona que pretende transmitir. É possível que o*Master* receba vários pedidos para comunicação por parte dos *slaves*, e mediante alguma política de escalonamento escolhida, conceda o acesso para comunicação a um deles [22]. Na última fase que termina um ciclo temporal no protocolo EPL, *Idle Period*, não existe qualquer tipo de comunicação na rede. Esta acção realizada pelo *Master* permite o início com precisão e sem interferências de um novo ciclo.

## 3.3 *Switched Ethernet*

Na secção 2.3 foram descritas algumas das funcionalidades do *switch*. O facto de apenas existir um domínio de colisão por cada porta do *switch* não garante o correcto funcionamento em aplicações de tempo real. Normalmente, quando as tramas chegam ao *switch* são guardadas numa fila e geridas segundo um critério *First-Come First-Served* (FCFS). Assim, mensagens com maior prioridade e com *deadlines* a cumprir serão processadas do mesmo modo que mensagens de menor prioridade, o que não é viável para aplicações críticas. O uso de filas de espera com diferentes prioridades, é uma solução que permite a distinção entre tráfego com características de tempo real e o restante [1].

Tendo em vista melhorar o desempenho das redes comutadas, como as de *Switched Ethernet*, várias soluções foram sendo desenvolvidas ao longo dos anos, como é o caso da PROFInet e TTEthernet, onde foram concedidos ao *switch* mecanismos de controlo e gestão de tráfego.

### 3.3.1 *PROFInet*

O *standard PROFInet*, desenvolvido pela *PROFIBUS* e *PROFInet Internacional*, consiste no uso da tecnologia *PROFIBUS* sobre *Ethernet*. Trata-se de uma tecnologia capaz de ser integrada em sistemas existentes e uma solução viável para comunicações críticas de tempo real. Os requisitos necessários na indústria da automação são salvaguardados no uso da tecnologia *PROFInet* pois foi possível adaptar e incorporar nesta as experiências e anos de estudo do *PROFIBUS* [23]. A tecnologia *PROFInet* apresenta três classes que permitem identificar o tipo de tráfego e deste modo obter melhores desempenhos [24]:

- TCP/IP para tráfego sem requisitos temporais críticos, como por exemplo mensagens de configuração;
- Tráfego de tempo real (RT), onde se incluí, como exemplo, o processamento de dados em sistemas industriais;
- Tráfego isócrono de tempo real (IRT) para situações críticas que requerem tratamento personalizado, como por exemplo para controlo de movimentos

A imagem 3.2 representa a arquitectura da pilha protocolar de rede da *PROFInet*.

Existem fundamentalmente duas versões, *PROFInet CBA (Component Based Automation)* e *PROFInet IO (Input/Output)*. *PROFInet CBA* é caracterizado pela definição de simples módulos distribuídos dotados com alguma inteligência formando uma rede funcional.

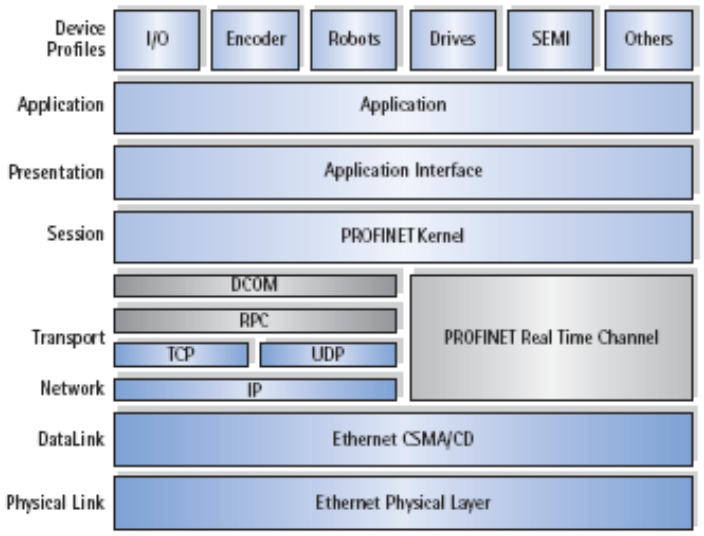



Figura 3.2: PROFInet - Arquitectura da Rede [25]

Esta versão do *PROFInet* fornece também um canal de comunicação baseado em TCP/IP para troca de informações sem requisitos temporais e disponibiliza um canal de comunicação para tráfego com características de tempo real. A versão *PROFInet IO* suporta comunicações com requisitos RT e IRT. Segue uma filosofia produtor/consumidor, onde os vários dispositivos enviam a informação a ser processada para consumidor (usualmente um *Programmable Logic Controller*-PLC) [23].

Esta tecnologia usa os *switches* como componentes de rede pois, como foi referido anteriormente, estes dispositivos não só diminuem e/ou anulam as colisões na rede o que permite um uso eficiente da largura de banda como também são capazes de regenerar as mensagens e reenviá-las de um modo selectivo [23]. Comunicações RT, requerem usualmente tempos de resposta na ordem dos 5 a 10$ms$ enquanto as comunicações IRT são mais críticas apresentando tempos de resposta de cerca de 1$ms$ e um *jitter* de 1µs [23].

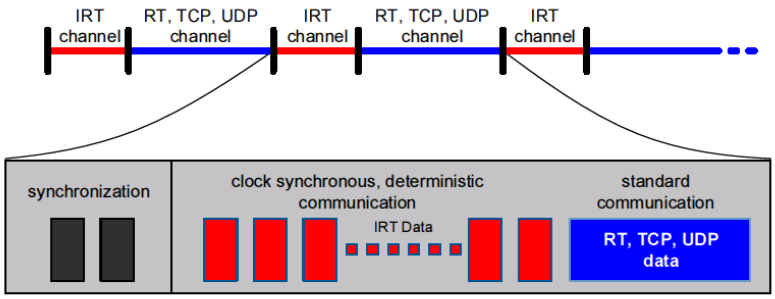



Figura 3.3: *PROFInet* - Janela temporal de comunicação [26]

Na figura 3.3 está representada a estrutura do ciclo de comunicação na tecnologia *PROFInet*. Podemos verificar que existe uma janela temporal bem definida para a comunicação IRT onde os *deadlines* são de maior importância, e outra janela para o tráfego de tempo real

não tão crítico e para tráfego sem requisitos temporais. Existe uma fase de sincronização dos dispositivos na rede. Em seguida são enviadas as informações IRT onde o intervalo temporal atribuído às aplicações para o envio dos dados é gerido por um algoritmo de escalonamento definido *à priori*. Depois, é enviado o restante tráfego em que o tamanho da janela temporal depende das mensagens a enviar em cada intervalo de tempo. A política de escalonamento é seleccionada mediante as especificações necessárias para a rede, tendo em conta a topologia da mesma, o tipo de informação que circula e características dos dispositivos presentes.

A tecnologia *PROFInet* utiliza *Ethernet* e TCP/IP para comunicação, e portanto é de fácil integração em sistemas já existentes. O valor 0x8892 inserido no campo Tipo das tramas *Ethernet* representa a *PROFInet*. No *payload* são inseridos os cabeçalhos IP e TCP, de tamanho mínimo de 24 *bytes* para ambos, juntamente com os dados a enviar. Existe um campo opcional a inserir nas tramas que define a prioridade da mesma.

### 3.3.2 *TTEthernet*

O protocolo *Time Triggered Ethernet* utiliza *Ethernet standard* com o objectivo de tornar possível a sua utilização em aplicações críticas de tempo real acrescentando algum determinismo, tolerância a falhas e propriedades de tempo real associadas ao paradigma *time triggered*. Foi desenvolvido pela *TTTech Computertechnik AG* em parceria com a Universidade de Viena [27]. Trata-se de um protocolo transparente em termos de sincronização, pois possibilita, no mesmo meio de comunicação, a existência de outro tipo de tráfego. O *TTEthernet* suporta todos os meios físicos da norma IEEE 802.3 para redes *Switched Ethernet*. [28]. Esta tecnologia faz uma divisão do tráfego em três classes: tráfego *time triggered* (TT), tráfego *rate-constrained* (RC) e tráfego *best-effort* (BE).

A informação TT representa o tráfego de tempo real de maior prioridade na rede, e portanto sobrepõem-se ao restante. É baseado no paradigma das comunicações *time triggered* onde existe necessidade de grande rigor no cumprimento de *deadlines* e precisão temporal, tendo em vista o bom funcionamento do sistema.

O tráfego RC, baseado no paradigma *event-triggered* é utilizado para informação de tempo real que não requer um determinismo e rigor temporal tão preciso como o tráfego TT. Apresentam uma garantia temporal e atrasos limitados. Não existe sincronização no envio deste tipo de mensagens pelo que podem ser enviadas em simultâneo informações para o mesmo ponto da rede, sendo necessário guardá-las numa fila para serem processadas, levando ao aumento do *jitter* de transmissão [27].

Em relação ao tráfego BE, é utilizado para envio de mensagens de baixa prioridade sem garantias de que a transmissão tenha sido realizada ou definição de atraso temporal. É portanto menos prioritário que o tráfego TT e RC, e é enviado sempre que exista largura de banda disponível.

A figura 3.4 ilustra a arquitectura da camada de rede da tecnologia *TTEthernet*. Os diferentes tipos de tráfego são enviados através do canal de comunicação segundo uma política de TDMA sendo os pacotes de maior prioridade enviados sempre que é necessário e os restantes sempre que exista largura de banda disponível, como referido.

Devido ao facto de poderem co-existir vários tipos de tráfego nesta tecnologia, existe a possibilidade de degradação da *performance* da rede quando o tráfego de maior prioridade ficar bloqueado por tráfego de menor prioridade. Segundo [27], existem alguns métodos que são utilizados pela tecnologia *TTEthernet* por forma a minimizar esta situação, que serão referidos de seguida.

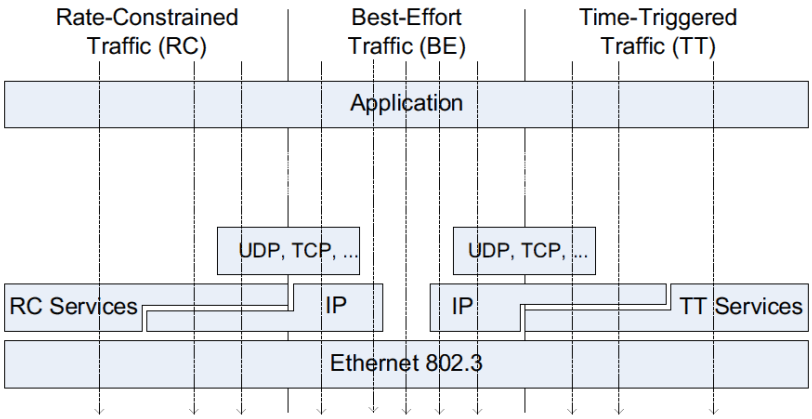



Figura 3.4: *TTEthernet* - arquitectura da rede [27]

Um desses métodos é designado de *preemption*. Nesta abordagem, quando uma mensagem de maior prioridade está pronta a ser transmitida e caso exista uma mensagem de prioridade menor a ser enviada esta é interrompida, via *jam sequence*. Em seguida, é esperado um tempo pré-definido para o envio da mensagem mais prioritária. Este método permite um menor atraso no envio das mensagens mais prioritárias, no entanto degrada a largura de banda, pois a mensagem que foi interrompida tem de ser re-transmitida.

Outra abordagem é designada por *timely block*. Em termos gerais, o *switch* tem conhecimento de quando uma mensagem de maior prioridade vai chegar e para onde vai ser transmitida. Neste método, o *switch* não vai transmitir qualquer tipo de mensagens de menor prioridade durante este intervalo, garantindo assim um atraso mínimo no envio da tráfego prioritário. No entanto, como existem períodos de tempo em que a rede não vai transferir informação, a largura de banda não é utilizada eficiente.

Uma outra situação consiste em atrasar o envio da mensagem de maior prioridade até a mensagem de menor prioridade ser completamente transmitida - designada de *shuffling*. Nesta abordagem, não existem intervalos de tempo sem informação relevante no canal de comunicação pelo que a largura de banda é utilizada eficientemente. Apesar disso, como as mensagens prioritárias podem não ser enviadas de imediato, vão sofrer atraso suplementar e *jitter*.

# Capítulo 4

# Arquitectura da Rede e Comportamento do *Switch*

Como foi referido no capítulo que introduz este documento, é objectivo deste projecto o desenvolvimento de uma infraestrutura que permita a aplicação da tecnologia *Ethernet* em sistemas de tempo real. Nesta secção serão apresentadas as ideias desenvolvidas para a estrutura e comportamento desta rede do ponto de vista agnóstico da sua implementação, fazendo-se uma análise qualitativa dos métodos utilizados.

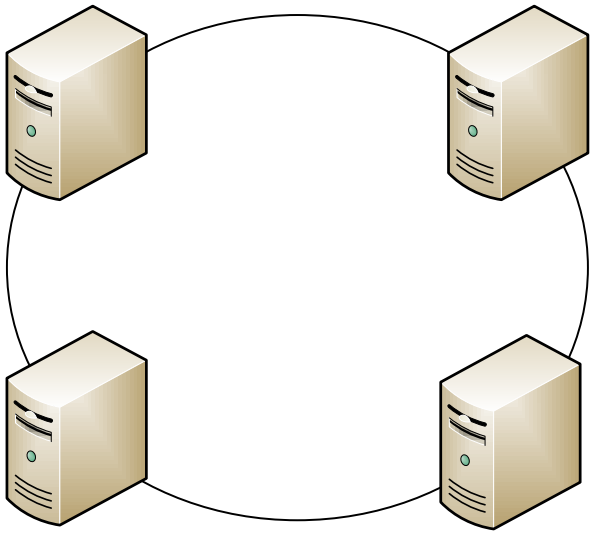


Figura 4.1: Protótipo da rede

Tendo em consideração os objectivos propostos, foi estipulado que a rede seria desenvolvida segundo uma topologia em anel, conforme a da figura 4.1, tirando partido das vantagens analisadas na secção 2.2, nomeadamente em termos de simplicidade no desenvolvimento e número reduzido de ligações. Sendo a rede montada segundo um anel físico, os nodos presentes apresentam apenas duas interfaces de rede, que representam entrada e saída de informação, e deste modo constata-se facilmente um menor número de ligações a cada nodo. Em contrapartida, não existem ligações dedicadas para cada nodo, e portanto estes são responsáveis por

retirar da rede a informação que a eles é destinada, implicando desta forma um aumento da complexidade da estrutura interna dos nodos. A arquitectura interna dos nodos está ilustrada na figura 4.2.

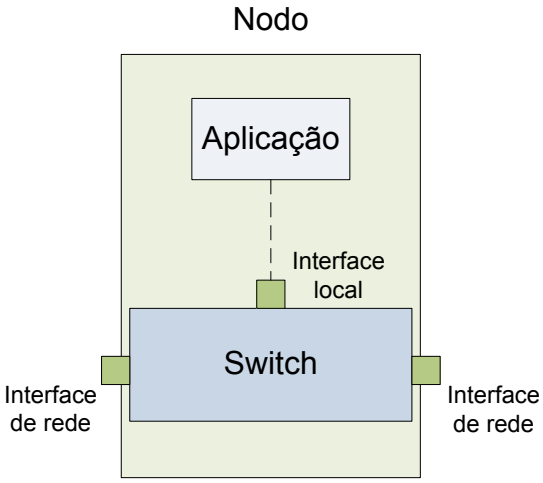



Figura 4.2: Visão alto nível da arquitectura dos nodos na rede

Cada nodo na rede será baseado num *switch* que reencaminhará as tramas que a ele não são destinadas e que será dotado com algumas funcionalidades que permitem a gestão e controlo da rede, como se verá mais à frente neste documento. Na rede desenvolvida cada nodo tem as funcionalidades necessárias para realizar a comutação de pacotes *Ethernet*, e portanto o *switch* é distribuído. Este *switch* apresenta uma interface local que reencaminha os pacotes destinados ao nodo para a respectiva aplicação. Dependendo da tecnologia sobre a qual será desenvolvido o nodo, várias especificações poderão ser aplicadas tendo em conta a especificidade e requisitos computacionais desejados. A título de exemplo, numa implementação do nodo em FPGA poderá ser facilmente realizável uma comutação de pacotes baseada nos paradigmas *cut through* ou *store and forward* referidos na secção 2.3. Neste tipo de implementações por *hardware* existem fundamentalmente vantagens a nível da capacidade de processamento. Numa abordagem por *software* a comutação baseada em *cut through* não é realizável e portanto outras considerações têm de ser tomadas, tirando partido dos recursos computacionais existentes e da qualidade da programação utilizada, de modo atingir os objectivos desejados.

## 4.1 Visão Global do Sistema

Como foi referido anteriormente, o sistema foi projectado segundo uma topologia em anel. A ferramenta desenvolvida tem o intuito de:

- Identificar a topologia;
- Reservar recursos na rede para tráfego periódico e esporádico de tempo-real;
- Implementar mecanismos para controlo de admissão de tráfego.
- Implementar mecanismos para gestão dos recursos da rede;

No arranque do sistema é efectuado um levantamento da topologia da rede. Com base nessa informação, são reservados e inicializados os recursos necessários para gerir a rede.

As aplicações presentes no sistema, quando querem enviar tráfego de tempo-real para a rede, têm que fazer um pedido. Os pedidos realizados são sujeitos a controlo de admissão sendo aceite apenas quando há recursos disponíveis. Os nodos na rede só enviam o tráfego que tenha sido admitido. De modo a respeitar este requisito, cada nodo faz um controlo do tráfego por si enviado.

Foi definida uma *Application Programming Interface* (API) que permite uma abstracção dos detalhes do sistema, e deste modo, uma interacção simples e fácil com o utilizador.

Os mecanismos de gestão da rede, que serão descritos em seguida, foram desenvolvidos tendo em vista numa primeira fase, o controlo de tráfego periódico e esporádico de tempo-real e gestão de topologia.

## 4.2 Formato Genérico das Tramas na Rede

Foi necessário estipular um protocolo para as mensagens de configuração da rede como também para as mensagens de dados. Após análise dos protocolos *Ethernet* registados e que podem ser encontrados em [29] foi definido que o protocolo para as tramas *Ethernet* que circulam na infraestrutura seria 0x88B5 uma vez que está definido para uso público e para investigação. Os diferentes tipos de mensagens a circular na infraestrutura serão depois distinguidos através de sub-tipos presentes no *payload* das tramas. A figura 4.3 ilustra a estrutura destas tramas. De notar que as tramas para configuração e controlo da rede são enviadas com o tamanho máximo de uma trama *Ethernet*, isto é, 1500 *bytes* no campo de dados. Foi definido desta maneira para facilitar a implementação, uma vez que as mensagens de configuração que foram definidas não apresentam igual estrutura, sendo umas de maior tamanho que outras.

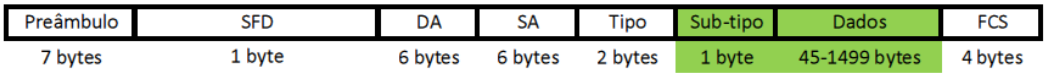



Figura 4.3: Estruturas das tramas que circulam na rede

Existem vários tipos de mensagens de configuração que podem ser enviadas na rede dependendo da finalidade. A tabela 4.1 identifica os diferentes sub-tipos considerados na infraestrutura desenvolvida. Para cada um destes sub-tipos, a estrutura presente no *payload* é diferente, como poderá ser constatado mais à frente neste documento.

| Sub-tipos | Valor (hexadecimal) |
|---|---|
| Levantamento da topologia | 0x01 |
| Reserva de Mensagem de Dados | 0x02 |
| Resposta à reserva | 0x03 |
| Dados tempo-real | 0x04 |
| Eliminar reserva | 0x05 |

Tabela 4.1: Vários sub-tipos de mensagens

As tramas para levantamento da topologia são enviadas quando a rede é inicializada

e permitem o conhecimento de quantos nodos existem na rede, a sua localização e a sua identificação. As tramas de reserva de recursos e resposta a essa reserva são trocadas quando existe a necessidade de um nodo enviar uma mensagem na rede, definindo-se assim controlo de admissão que se pretende. Esta situação é ilustrada na figura 4.4 em que o Nodo A requisita recursos para o envio de dados para a rede e o nodo B, baseado na política de controlo de admissão implementada, responde se é possível.

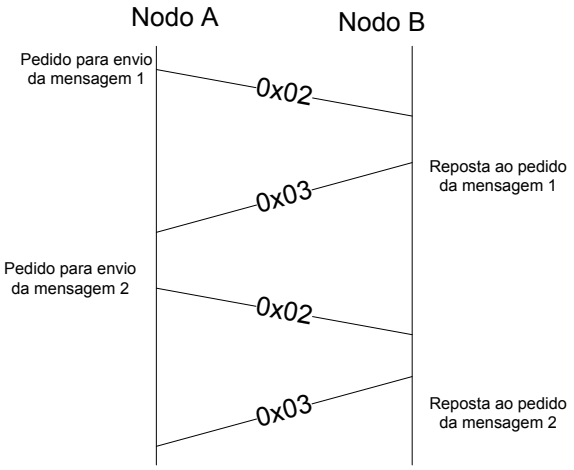



Figura 4.4: Fluxo das mensagens de pedido e resposta à reserva de recursos

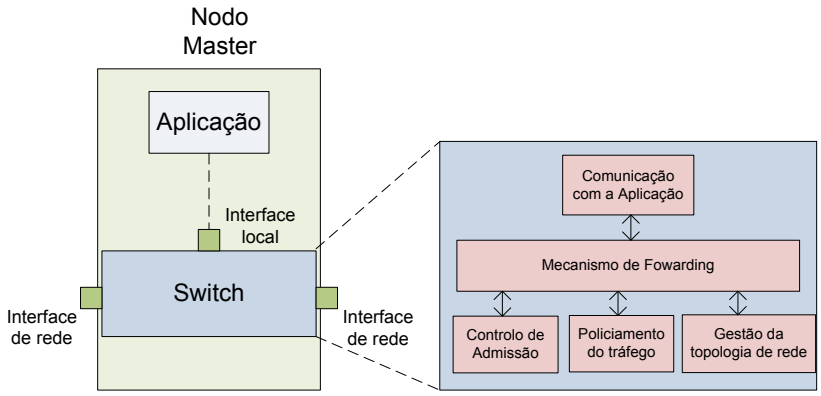
A informação de tempo real que os nodos enviam para a rede é encapsulada em tramas bem definidas como se verá mais adiante neste documento. Quando é necessário libertar recursos que estão a ser utilizados são trocadas mensagens para eliminar as reservas efectuadas pelos nodos.

## 4.3 Arquitectura dos Nodos

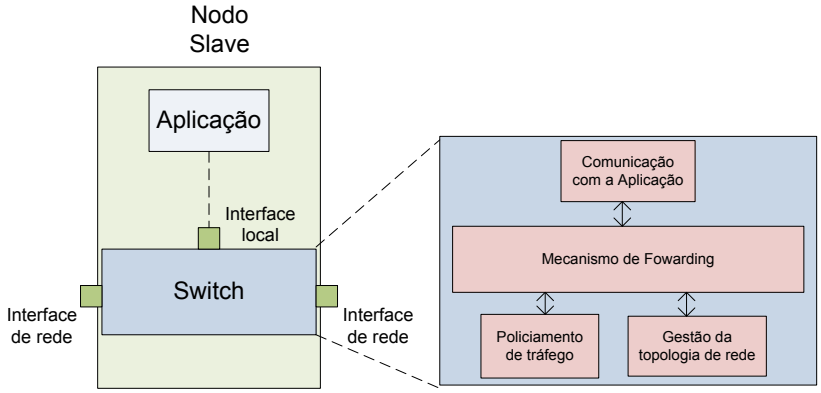
A rede é constituída por um nodo *Master* e por nodos *Slaves* de forma a existir um melhor controlo do tráfego na rede. A figura 4.5 tem o objectivo de ilustrar a arquitectura dos nodos.

Através da análise da figura pode constatar-se que arquitectura dos nodos é semelhante. Tanto o nodo *Master* como os nodos *Slaves* apresentam um bloco capaz de realizar a comutação dos pacotes com os requisitos pré-determinados. Aliado a esta situação estes nodos apresentam também mecanismos que permitem a comunicação de modo transparente com a aplicação à qual são destinados.

O nodo *Master* tendo em vista uma melhor gestão da rede e do tráfego que nela circula foi dotado com funcionalidades/privilégios adicionais. Este nodo é estático e é pré-definido antes da inicialização da rede. Apresenta um bloco responsável pelo controlo de admissão implementado que, baseado nas condições da rede e dos recursos disponíveis, irá aceitar ou rejeitar os pedidos para reserva de recursos realizados pelos *Slaves*. O nodo *Master* é também responsável por gerar mensagens que permitem uma gestão da topologia, mensagens estas devidamente recebidas e processadas pelos *Slaves*. O funcionamento dos nodos *Master* e *Slaves* será detalhado mais à frente neste documento.




(a) Arquitectura do nodo *Master*




(b) Arquitectura do nodo *Slave*

Figura 4.5: Arquitectura dos Nodos

## 4.4 Estruturas de Dados para Gestão da Topologia e da Rede

Ao longo desta secção serão descritas todas as estruturas que foram implementadas no desenvolvimento do projecto. Estas estruturas foram desenvolvidas com o objectivo de tornar mais fácil e directo o controlo da gestão de tráfego da rede.

### 4.4.1 Estrutura da Tabela de Mensagens do *Master*

A tabela onde estão registadas todas as mensagens a circular na rede é criada quando o nodo *Master* é colocado em funcionamento. Cada linha desta tabela representa uma mensagem na rede.

| Id. da Msg. | Id. Source | Id. Destination | Period (ms) | Length(*bytes*) | Valid |
|---|---|---|---|---|---|
| 4 *bytes* | 1 *byte* | 1 *byte* | 4 *bytes* | 4 *bytes* | 1 *byte* |

Figura 4.6: Linha da tabela do *Master* para controlo das Mensagens na Rede

As informações necessárias para o registo da mensagem são as ilustradas na figura 4.6. Os cinco primeiros campos dizem respeito às características específicas de cada mensagem:

- O primeiro campo representa o identificador da mensagem. Trata-se de uma variável que pode ser representada em 4 *bytes*;
- De seguida estão representados os identificadores do nodo que pretende enviar a mensagem e do nodo para o qual esta é destinada, respecticamente. São variáveis de 1 *byte* cada uma;
- O campo *period*, representa a periodicidade com que o *Slave* pretende enviar a sua mensagem e está definida em *ms*;
- O tamanho do campo de dados, em *bytes*, é especificado na variável *length*.

O campo *valid*, representa se a entrada da tabela está ocupada ou não. Quando a tabela é criada, todas as entradas têm aquele campo definido com o valor zero. Sempre que uma mensagem é registada, o valor é colocado a um. Isto permite de modo fácil a pesquisa de uma nova entrada na tabela, bastando procurar uma linha da tabela cujo campo *valid* seja zero. Quando uma mensagem é para ser removida da rede, é suficiente procurar o identificador dela na tabela do *Master* e, na respectiva entrada, mudar o valor do campo *valid* desta de um para zero.

### 4.4.2 Estrutura da Tabela de Gestão da Largura de Banda do *Master*

Na secção 4.5 será descrito o procedimento de como o *Master* tem conhecimento de quantos nodos existem na rede e os seus identificadores, através do envio de uma mensagem por uma das suas interfaces e depois a chegada dessa mesma mensagem pela outra. Como os nodos *Slaves* escrevem os seus identificadores pela ordem à qual a mensagem passa, sabe-se perfeitamente qual a disposição dos nodos na rede. A informação que essa mensagem contém, pode ser organizada na estrutura de uma tabela de modo a desambiguar a rede.

A tabela de gestão da largura de banda foi criada com um tamanho fixo, e cada linha da tabela é uma estrutura como a que está ilustrada na figura 4.7.

| Id. Nodo | Id. Nodo Prev | Id. Nodo Next | Bandwith Prev | Bandwith Next |
|---|---|---|---|---|
| 1 *byte* | 1 *byte* | 1 *byte* | 4 *bytes* | 4 *bytes* |

Figura 4.7: Estrutura de cada elemento da lista

Pela análise da imagem, constata-se que para cada nodo na rede existe a referência do identificador do nodo que lhe precede como também do nodo que lhe sucede na infraestrutura em anel desenvolvida. Existe ainda, para cada nodo, a informação da largura de banda que está a ser utilizada na ligação com o nodo seguinte e na ligação com o nodo anterior, representados pelas variáveis *Bandwith Next* e *Bandwith Prev*, respectivamente. Estas duas variáveis representam a largura de banda das ligações em *Kbps* e, no contexto deste projecto, cada ligação tem 100 *Mbps* de largura banda máxima disponível. Como se constata também existe uma certa redundância na tabela, uma vez que como existe apenas uma ligação entre dois nodos consecutivos, a largura de banda seguinte de um nodo é igual à largura de banda anterior do outro. Esta tabela, torna mais clara a estrutura da rede e a ligação de precedência entre os nodos. É através dela, que quando um *Slave* faz o pedido para o envio de uma mensagem é feita a análise entre os nodos de destino e de origem para perceber o número de nodos intermédios. Esta tabela tem a informação de toda a largura de banda a ser consumida

em cada ligação, e precisa de ser actualizada, apenas nas ligações correspondentes, sempre que uma mensagem é aceite e sempre que uma mensagem é retirada da rede.

### 4.4.3 Estrutura da Tabela de Mensagens do *Slave*

Os *Slaves* têm também registadas as mensagens que a ele estão associadas. A estrutura das suas tabelas é semelhante à utilizada no *Master* ilustrada na figura 4.6. No entanto, foram acrescentados alguns campos de modo a criar mecanismos para uma gestão mais eficiente da rede, campos estes representados na figura 4.8.

| Producer/Consumer | Timestamp | Interface |
|---|---|---|
| 1 *byte* | time_t | 1 *byte* |

Figura 4.8: Campos extra da tabela de mensagens nos *Slaves*

Um dos campos adicionados está representado por *Producer/Consumer*. Este campo permite indicar se o *Slave* é produtor, com o valor um, ou consumidor, com o valor zero, da mensagem. O outro campo é representado pela variável *TimeStamp* que, dependendo se o *Slave* está a produzir ou a consumir a mensagem, indica o tempo da última transmissão ou última recepção, respectivamente. Com estas duas variáveis, é possível aplicar mecanismos de gestão da rede, pois é fácil de detectar se um *Slave* está ou não a produzir ou a consumir uma mensagem e, assim, assinalar anomalias na rede.

O campo *TimeStamp* é também útil para verificar se a mensagem está a cumprir a periodicidade definida. Quando a mensagem é para ser enviada, é necessário comparar o tempo da última transmissão com o actual e verificar se se encontra coerente com o período estipulado.

Por último, o campo *Interface* indica a interface de rede pela qual o *Slave* irá enviar a informação.

No envio de uma mensagem, é portanto necessário confirmar se todas as características desta estão válidas e de acordo com o estipulado na respectiva tabela. Em caso de alguma anomalia, a mensagem não é enviada.

### 4.4.4 Estrutura das Mensagens que circulam na rede

Para que todas as tabelas sejam devidamente preenchidas, é necessário trocar mensagens de configuração e controlo. Como foi referido na secção 4.2, as diferentes mensagens na rede são distinguidas através de um sub-tipo, que é colocado no primeiro *byte* referente ao *payload*. Dependendo do tipo de mensagens de controlo, os restantes bytes do *payload* têm significados diferentes. As mensagens definidas têm como destino o endereço *broadcast* e portanto, quando um pacote chega a um nodo é necessário analisar o sub-tipo das tramas e, como base nesse valor, tomar decisões.

Estas estruturas são preenchidas com as respectivas informações e depois encapsuladas no campo de dados das tramas *Ethernet*. Para cada sub-tipo de mensagens que circulam na rede, as suas variáveis e os respectivos significados são representados de seguida.

#### Pedido para reserva de mensagem de dados

No pedido para o envio de dados por parte de um *Slave*, é enviada uma trama *Ethernet* com o *payload* estruturado como mostra a figura 4.9.

| 0x02 | Id. Msg | Id. Source | Id. Destination | Period (ms) | Length(*bytes*) |
|---|---|---|---|---|---|
| 1 *byte* | 4 *byte* | 1 *bytes* | 1 *byte* | 4 *bytes* | 4 *bytes* |

Figura 4.9: Estrutura do *payload* da trama de pedido para envio de dados

Na trama, estão claramente identificados a origem e o destino da informação como também a periodicidade, o tamanho e o identificador desta. O nodo *Master*, ao receber uma trama segundo o protocolo 0x88B5 e cujo o primeiro byte do *payload* é o valor 0x02, sabe que se trata de um pedido para o registo de uma mensagem. Deste modo, o *Master* baseado nesta trama, irá fazer as respectivas verificações e responder ao *Slave* com a decisão tomada.

### Resposta à reserva para o envio de mensagem de dados

A figura 4.10 ilustra o formato das tramas *Ethernet* que o nodo *Master* utiliza para responder aos pedidos para envio de dados feitos pelos *Slaves*.

| 0x03. | Id. Msg | Id. Source | Id. Destination | Period (ms) | Length(*bytes*) | Aceppted | Interface |
|---|---|---|---|---|---|---|---|
| 1 *byte* | 4 *bytes* | 1 *byte* | 1 *byte* | 4 *bytes* | 4 *bytes* | 1 *byte* | 1 *byte* |

Figura 4.10: Estrutura do *payload* da trama de resposta ao pedido para envio de dados

Estas tramas, identificadas pelo sub-tipo 0x03, têm todas as características das mensagens que se prentende enviar e ainda dois campos extra. O campo *Aceppted* é a resposta que o *Master* envia ao Slave: em caso de ter o valor um, significa que a mensagem foi aceite e registada, e em caso de ser o valor zero, significa que a mensagem foi rejeitada. O campo *Interface* só tem significado se o pedido foi aceite, uma vez que tem o objectivo de indicar ao *Slave* por qual das suas interfaces deve enviar a informação. A rede foi montada, para que, aquando do envio da mensagem de reconhecimento da topologia, esta chegasse ao *Slave* por uma interface conhecida e fosse reencaminhada pela outra. Desta forma, o nodo *Master* tem conhecimento das interfaces que o *Slave* deve usar, mediante a decisão tomada, para o envio da informação.

### Remoção de uma mensagem

Quando algum nodo quer parar o envio de uma mensagem, as várias tabelas, tanto no respectivo nodo *Slave* como no nodo *Master* têm de ser actualizadas. Neste sentido, o *Slave* envia uma mensagem ao *Master* com a estrutura ilustrada na figura 4.11.

| 0x05 | Id. Source | Id. Msg | Interface |
|---|---|---|---|
| 1 *byte* | 1 *byte* | 4 *bytes* | 1 *byte* |

Figura 4.11: Estrutura da trama para remoção de uma mensagem

Este sub-tipo de tramas são identificados com o valor 0x05 no primeiro *byte* do *payload*. Os campos seguintes são, respectivamente o identificador do nodo origem da informação, o identificador de mensagem, que é único para cada mensagem existente na rede e a interface por onde a mensagem está a ser enviada. O *Slave* procura a mensagem na sua tabela, através do identificador desta, e de seguida muda o valor do campo *valid* para zero, indicando assim que a respectiva linha da tabela não tem informação útil. O nodo *Master* ao receber

a respectiva trama para remoção executa um procedimento semelhante na sua tabela de mensagens. Necessita ainda de actualizar a largura de banda, na respectiva tabela que gere a largura de banda, diminuindo o valor desta nas ligações por onde a mensagem iria passar.

Assim, a mensagem deixa de estar registada na rede, e não poderá ser mais enviada até novo registo.

#### Envio de dados de tempo real

A informação que se pretende enviar para a rede deve estar encapsulada numa estrutura que permita um acesso fácil e directo. A estrutura das mensagens de dados que circulam na rede está ilustrada na figura 4.12.

| 0x04 | Id.Source | Id. Destination | Data |
|---|---|---|---|
| 1 *byte* | 1 *byte* | 1*byte* | 43 a 1497 *bytes* |

Figura 4.12: Estrutura da trama de dados de tempo real

Quando chega uma trama cujo o primeiro *byte* do *payload* é 0x04, os nodos sabem que se trata de uma mensagem de dados. Em seguida, o procedimento é verificar se o identificador de destino da informação coincide com o identificador do respectivo nodo. Em caso afirmativo, a mensagem é recebida e retirada da rede. Em caso contrário, a mensagem não chegou ao destino e portanto é reencaminhada. Como já foi referido, as mensagens de dados são enviadas com destino *broadcast*, e portanto cabe ao respectivo nodo, através desta análise à trama, perceber se é para ele ou não.

Pela análise da figura 4.12, a seguir ao identificador do nodo origem, está explicitado o identificador do nodo de destino e só depois estão os dados propriamente ditos. Devido à estrutura da trama de dados e à necessidade desta existir para ajudar a desambiguar a rede, só podem ser apenas enviados por cada trama de dados, no máximo, 1497 *bytes* de informação útil a transmitir, pois os três *bytes* iniciais do *payload* são para controlo.

## 4.5 Gestão da topologia da rede

Cada nodo na rede é representado por um identificador de 1 *byte* que o caracteriza. É necessário ter um controlo de quantos nodos existem e a sua localização na rede. Neste sentido, e como foi referido na secção 4.3, o nodo *Master* ou Monitor apresenta mais funcionalidades que os restantes *Slaves*.

Este nodo *Master*, quando é inserido na rede, envia uma trama *Ethernet* segundo o protocolo estabelecido, com o objectivo de perceber quantos nodos estão na rede e a sua localização relativa. A figura 4.13 ilustra a estrutura da trama de reconhecimento da topologia. Cada um dos *Slaves* ao receber esta trama, incrementa a variável que indica o número de nodos na rede e adiciona ao conteúdo desta o seu identificador. Em seguida, deve reencaminhá-la pela porta oposta à qual a recebeu. Quando o nodo *Master* voltar a receber a trama (pois trata-se de uma rede em anel, e portanto um circuito fechado), sabe ao certo o número de nodos na rede, o identificador de cada um e a sua posição face aos restantes. Neste ponto, é possível criar uma tabela que permite definir e gerir as ligações entre cada um dos nodos.

De modo a ilustrar melhor o método utilizado, considere o exemplo ilustrado na figura 4.14.

| Sub-tipo (0x01) | Número de Nodos | Id. do Nodo 1 | Id. Nodo 2 | ... |
|---|---|---|---|---|
| 1 byte | 1 byte | 1 byte | 1 byte | 1byte |

Figura 4.13: Estrutura da trama enviada pelo *Master* para reconhecimento da Topologia

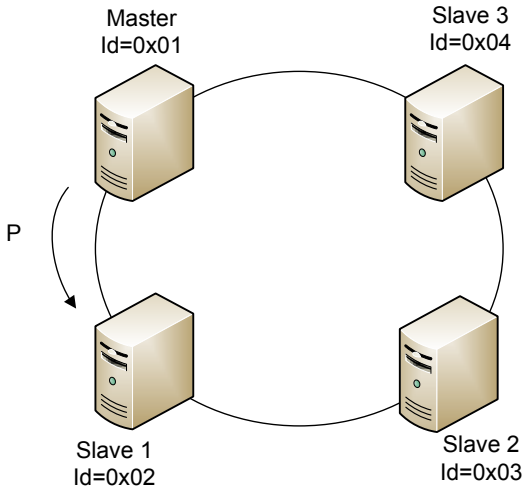



Figura 4.14: Exemplo do método para reconhecimento da topologia

O nodo *Master* envia uma trama P para reconhecimento da topologia, conforme ilustra a figura, cujo conteúdo inicial é mostrado na figura 4.15.

| Sub-tipo (0x01) | 1 | 0x01 |
|---|---|---|

Figura 4.15: Trama enviada pelo *Master* de acordo com o exemplo

Ao receber a trama P, o *Slave 1* sabe que é de reconhecimento da topologia, através do sub-tipo da trama, e que naquele momento existe um nodo na rede, o *Master* com o Id. 0x01. De seguida o *Slave 1*, actualiza a trama P, aumentando o número de nodos na rede e acrescentando o seu Id. ao conteúdo, e envia-a para o *Slave 2*. A trama enviada *pelo Slave 1*, neste momento tem a estrutura ilustrada na figura 4.16.

| Sub-tipo (0x01) | 2 | 0x01 | 0x02 |
|---|---|---|---|

Figura 4.16: Trama enviada pelo *Slave 1* ao *Slave 2* de acordo com o exemplo

O raciocínio é semelhante para o preenchimento da trama P nos *Slaves* 2 e 3, respectivamente. Tratando-se de uma topologia em anel, esta trama vai voltar ao nodo *Master* na porta oposta à de saída. Neste ponto, o conteúdo final da trama P é o apresentado na figura 4.17.

| Sub-tipo (0x01) | 4 | 0x01 | 0x02 | 0x03 | 0x04 |
|---|---|---|---|---|---|

Figura 4.17: Trama P no regresso ao nodo *Master* de acordo com o exemplo

Quando o *Master* recebe novamente a trama P tem conhecimento do número de nodos na rede, dos seus identificadores e das suas posições relativas na rede.

Na fase de arranque do sistema é necessário que os *Slaves* estejam já inseridos na rede para que, aquando da activação do nodo *Master*, estejam disponíveis para responder à mensagem que permite ao *Master* ter um conhecimento da rede. Caso este não receba a mensagem para reconhecimento da topologia, é porque existe alguma anomalia na rede, nomeadamente poderá haver algum *Slave* inactivo ou alguma ligação poderá ter sido cortada. Através desta mensagem os *Slaves* têm também conhecimento do identificador do *Master*.

## 4.6 Gestão de Tráfego na rede

### 4.6.1 Nodo *Master*

Por forma a tornar fiável o funcionamento da rede e assegurar os requisitos temporais requiridos para a mesma, é necessário existir controlo do tráfego nela existente. Neste sentido, outra das funcionalidades do *Master* é o controlo do tráfego na rede através da definição de uma política de controlo de admissão. A ideia é que toda a informação de tempo real que circula na rede, esteja registada no nodo *Master* de modo a existir um controlo constante dos recursos que estão a ser utilizados. Neste sentido, foi desenvolvida uma tabela no *Master*, com um tamanho máximo de mensagem onde estão registadas todas as informações que estão na rede e as suas características. A estrutura detalhada desta tabela foi apresentada na secção 4.4.1.

Quando um *Slave* pretende enviar dados para a rede necessita de efectuar uma reserva junto do *Master*, enviando para este uma mensagem. Esta mensagem enviada necessita de ter informações como, os identificadores do nodo que pretende enviar os dados e do nodo de destino destes, o tamanho total da mensagem e a periodicidade com que esta vai ser enviada. O próprio *Slave* atribui um identificador à mensagem, de modo a representá-la e ajudar a desambiguar de outras com possíveis características semelhantes. Com base nesta trama, o *Master* vai analisar a disponibilidade actual da rede e responder ao *Slave* se a sua mensagem pode ser ou não enviada. O diagrama da figura 4.18 ilustra a análise realizada pelo *Master*.

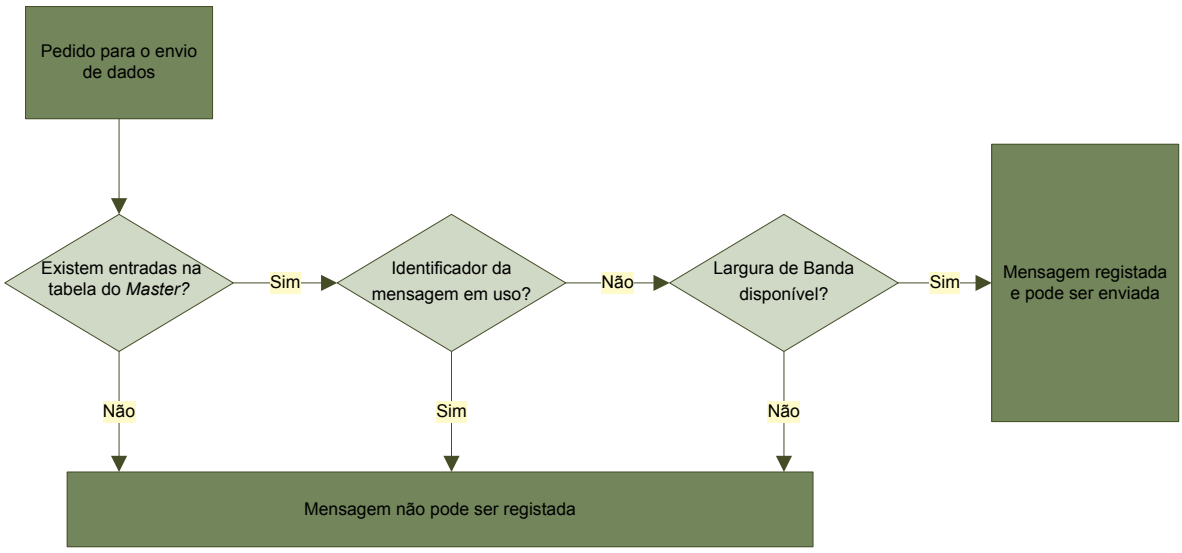



Figura 4.18: Diagrama de Fluxo alto nível da gestão de mensagens no *Master*

Na chegada de um novo pedido para envio de informação na rede, é necessário verificar primeiro se a tabela onde o *Master* regista todas as mensagens tem entradas disponíveis. Em caso de não existir espaço na tabela, é respondido ao *Slave* que não é possível enviar a informação que pretende. Caso seja possível registar novas mensagens é necessário averiguar se o identificador da mensagem não se encontra já registado. Como foi referido atrás, cada *Slave* caracteriza a sua mensagem com um identificador. Se a mensagem que se pretende enviar tem um identificador já em uso por outra presente na tabela do *Master*, então esta não pode ser registada e portanto não pode ser enviada pelo *Slave*.

Nesta fase, e passadas as verificações já realizadas, torna-se necessário saber se existe largura de banda suficiente para permitir que a mensagem seja enviada sem deteriorar o desempenho da rede. Quando foi feito o levantamento da topologia e do número de nodos existentes na rede, o *Master* tem conhecimento do número de ligações existentes e disposição dos nodos. Desta maneira, é possível manter uma tabela com a largura de banda a ser utilizada na ligação entre dois nodos consecutivos. Esta tabela através da qual é gerida a largura de banda nas ligações da rede foi detalhada na secção 4.4.2. Tratando-se de uma rede em anel, existem dois caminhos pelos quais a mensagem pode seguir até chegar ao destino. O *Master* calcula a largura de banda necessária para o envio da mensagem em pedido e vai verificar, nos dois caminhos possíveis, se existe largura de banda disponível. Desta análise podem derivar vários resultados:

- Não existe largura de banda em nenhum dos dois caminhos possíveis e, portanto, a mensagem não pode ser registada no *Master* nem enviada pela *Slave*.
- Existe largura de banda suficiente para o envio da mensagem do *Slave* em pelo menos um dos dois caminhos possíveis:
  - Se existir largura de banda em ambos os caminhos, o *Master* irá escolher aquele que minimiza o número de nodos intermédios;
  - Se apenas existe largura de banda num sentido, o *Master* irá, obviamente, escolher o disponível.

Passadas todas estas verificações, e caso o pedido do *Slave* para o envio da mensagem com as características estipuladas tenha sido aceite, as seguintes operações são executadas:

1. As características da mensagem são registadas na tabela de mensagens do *Master*;
2. É actualizada a largura de banda nas ligações pelas quais a mensagem irá passar, na tabela do *Master* que gere a largura de banda;
3. É enviada, pelo *Master*, uma resposta ao *Slave* a informar que a mensagem foi aceite e a indicar o caminho pelo qual os dados devem ser enviados.

Em caso contrário, isto é, se o pedido do *Slave* não for aceite, este receberá uma mensagem de resposta a indicar essa informação.

#### Exemplo do registo de mensagens nas tabelas do *Master*

A titulo de exemplo é ilustrado um hipotético cenário em que já existe uma mensagem T a circular na rede do *Slave 2* para o *Slave 1*, conforme ilustra a figura 4.19. Em seguida é

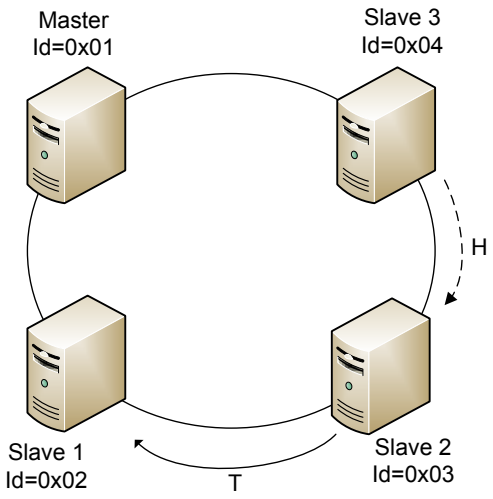



Figura 4.19: Exemplo de uma mensagem de dados a ser enviada do *Slave 2* para o *Slave 1*

feito um pedido por parte do *Slave 3* para o envio de uma mensagem H. Foram consideradas que todas as ligações entre os nodos são iguais e com largura de banda máxima de 100 *Mbps*.

Na respectiva tabela do *Master* para o registo das mensagens, tendo em conta a estrutura desenvolvida e apresentada em 4.4.1, encontra-se a informação mostrada na tabela 4.2.

| Id. da Msg. | Id. Source | Id. Destination | Period (ms) | Length(*bytes*) | Valid |
|---|---|---|---|---|---|
| 20 | 0x03 | 0x02 | 1 | 1000 | 1 |

Tabela 4.2: Tabela de mensagens do *Master* no exemplo em análise

De acordo com a tabela, a mensagem T é caracterizada pelo identificador 20, está a ser enviada do *Slave 2* para o *Slave 1* com um período de 1 *ms* e o seu campo de dados ocupa 1000 *bytes*. Tendo em consideração a estrutura de uma trama *Ethernet* apresentada em 2.1.2 juntamente com os 12 *bytes* mínimos referentes ao *Interframe Gap* (IFG), a largura de banda consumida por esta mensagem é de 8034 *Kbps*. A tabela que gere a largura de banda em uso no *Master* está preenchida como ilustra a tabela 4.3.

| Id. Nodo | Id. Nodo Prev | Id. Nodo Next | Bandwith Prev | Bandwith Next |
|---|---|---|---|---|
| 0x01 | 0x04 | 0x02 | 0 | 0 |
| 0x02 | 0x01 | 0x03 | 0 | 8034 |
| 0x03 | 0x02 | 0x04 | 8034 | 0 |
| 0x04 | 0x03 | 0x01 | 0 | 0 |

Tabela 4.3: Tabela da largura de banda do *Master* no exemplo em análise

Supondo agora que o *Slave 3* pretende enviar uma mensagem H cujas características são as seguintes:

- Id. da mensagem é o 30;
- Destino da mensagem é o *Slave 2* (0x03);

- A mensagem tem um período de 2 *ms* e o campo de dados tem 1500 *bytes*

O *Slave 3* envia uma mensagem ao *Master* com estes requisitos a pedir permissão de envio. O *Master*, tendo capacidade para registar novas mensagens, vai verificar que na sua tabela de mensagens não existe nenhuma entrada cuja mensagem tenha o identificador 30 e, portanto, esta validação é verificada.

Neste fase é necessário, analisar a largura de banda das ligações em ambos os sentidos do anel. A mensagem H vai ocupar na rede, caso seja aceite, uma largura de banda de 6152 *Kbps*. Analisando a tabela, nos dois caminhos disponíveis, a largura de banda que a mensagem H irá consumir na rede não ultrapassa a máxima das ligações. Deste modo a mensagem H é aceite, e portanto registada. O *Slave 3* é informado, pelo *Master*, desta decisão como também da interface pela qual deve enviar a mensagem que, neste caso, como existe largura de banda nos dois caminhos disponíveis, é escolhido o que minimiza o número de nodos intermédios (*Slave 3* está directamente ligado ao *Slave 2*).

No final, as tabelas ficam preenchidas como se ilustra nas tabelas 4.4 e 4.5.

| Id. da Msg. | Id. Source | Id. Destination | Period (ms) | Length(*bytes*) | Valid |
|---|---|---|---|---|---|
| 20 | 0x03 | 0x02 | 1 | 1000 | 1 |
| 30 | 0x04 | 0x03 | 2 | 1500 | 1 |

Tabela 4.4: Tabela de mensagens do *Master* no final do exemplo em análise

| Id. Nodo | Id. Nodo Prev | Id. Nodo Next | Bandwith Prev | Bandwith Next |
|---|---|---|---|---|
| 0x01 | 0x04 | 0x02 | 0 | 0 |
| 0x02 | 0x01 | 0x03 | 0 | 8034 |
| 0x03 | 0x02 | 0x04 | 8034 | 6152 |
| 0x04 | 0x03 | 0x01 | 6152 | 0 |

Tabela 4.5: Tabela da largura de banda do *Master* no final do exemplo em análise

Este simples exemplo ilustra o mecanismo de controlo de admissão descrito.

### 4.6.2 Nodos *Slaves*

Do ponto de vista dos *Slaves* é necessário ter também um controlo de mensagens de modo a que não sejam enviadas para a rede informações que não estejam registadas. Os *Slaves* antes de enviarem mensagens para a rede fazem o requisito de reserva de recursos que será sujeito a um controlo de admissão pelo *Master*, conforme já analisado. Com base na resposta recebida, os *Slaves* registam ou não as mensagens que podem enviar para a rede. Este mecanismo está ilustrado na figura 4.20. Neste sentido, foi desenvolvida uma tabela, para cada *Slave*, onde estão registadas todas as mensagens que podem ser enviadas por este. Esta tabela tem tamanho limitado, e portanto, um *Slave* antes de fazer o pedido junto do *Master* tem verificar se tem entradas livres na sua tabela, para que em caso de ser aceite, possa registar a mensagem.

A estrutura da tabela é semelhante à existente no *Master*, conforme foi analisado na secção 4.4.3.

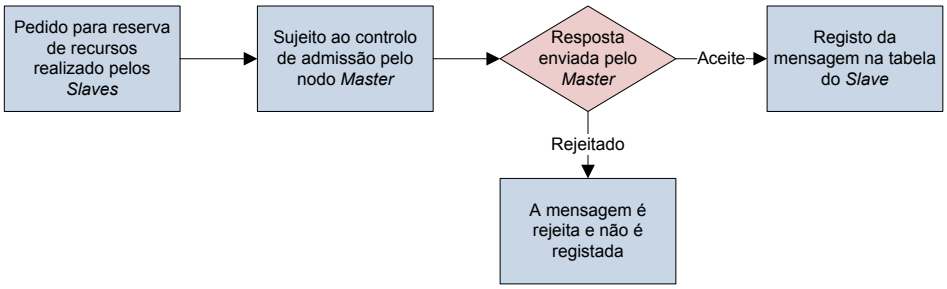



Figura 4.20: Procedimento dos *Slaves* face ao controlo de admissão realizado pelo nodo *Master*

Antes de um *Slave* enviar a informação que pretende, tem que analisar a sua tabela e fazer algumas verificações de modo a não comprometer o desempenho da rede:

1. É necessário verificar, através do identificador, se a mensagem se encontra registada na sua tabela, e portanto se foi aceite por parte do *Master*;

2. Caso a mensagem esteja registada, então é preciso averiguar se a informação que vai ser enviada não excede o tamanho registado na tabela;

3. De seguida vai se comparar o instante de última transmissão com o actual e perceber se os requisitos temporais estão a ser cumpridos.

Em caso de falha de alguma destas verificações, a mensagem não pode ser enviada e, deste modo, salvaguarda-se o correcto funcionamento da rede.

Quando um *Slave* pretende parar o envio de uma mensagem, é removida a entrada dessa mensagem da sua tabela. Simultaneamente, é enviada um pacote ao *Master* a informar deste procedimento, para que sejam também actualizadas as suas tabelas: tanto a tabela de gestão de mensagens como a tabela de gestão da largura de banda.

### 4.6.3 Envio de dados

Nas secções anteriores foram descritos os métodos utilizados na gestão de rede. Como foi constatado, antes de enviar informação para a rede é necessário um processo de comunicação entre o *Master* e os *Slaves*. Este controlo de admissão é baseado em pacotes com destino *broadcast*, sendo necessário mecanismos de processamento nos nodos para analisar as tramas. A única informação possuída pelos *Slaves* é a lista de identificação dos nodos presentes na rede.

O envio de dados para rede é também realizado com o destino *broadcast*, cabendo ao nodo para onde a informação é destinada retirá-la. Aquando da recepção de cada mensagem, os nodos verificam se o seu identificador coincide com o destino presente na trama de modo a perceberem se lhes é destinada. Em caso afirmativo os dados são guardados localmente. Caso contrário encaminham a mensagem para a interface de rede contrária à da recepção da mesma. Desta forma, apenas a comunicação *unicast* é suportada. A comunicação de dados em *multicast* e *broadcast* será alvo de trabalho futuro. A estrutura da mensagem de dados foi especificada na secção 4.4.4.

## 4.7 Sumário

Ao longo deste capítulo foi explicado o princípio de funcionamento da infraestrutura que se pretende desenvolver. Detalhou-se o controlo de admissão para o registo de mensagens realizado entre os nodos *Master* e *Slaves* como também a política de controlo e de gestão dos respectivos nodos antes de utilizarem os recursos da rede. As estruturas utilizadas na gestão da rede, nomeadamente os vários tipos de mensagens de controlo e de dados como também das tabelas presentes nos nodos *Master* e *Slave*, foram expostas e detalhadas.

# Capítulo 5

# Implementação em *Software* do *Switch Ethernet*

Devido à disponibilidade imediata de recursos e à flexibilidade, optou-se pela implementação do sistema apresentado no Capitulo 4 em *software* com recursos a computadores embutidos com Linux. Neste sentido, foram utilizados computadores embutidos com o sistema operativo Linux e gerados pequenos módulos programáveis ao nível do *kernel*, onde serão implementadas as especificidades requeridas, nomeadamente mecanismos de gestão de topologia da rede e encaminhamento de pacotes na rede.

Ao longo desta secção irá ser feita uma breve descrição sobre a estrutura e funcionamento da pilha protocolar da camada de rede nos sistemas Linux, sobre o modo como a informação circula na *stack* de rede a abordagem utilizada na captura e processamento dos pacotes. Irá também ser descrita e ilustrada a rede desenvolvida e as estruturas implementadas responsáveis pela gestão da mesma.

## 5.1 *Stack* de rede dos Sistemas Linux

A arquitectura de rede dos sistemas Linux pode ser segmentada em três partes: o espaço do utilizador (*user space*), o espaço do *kernel* (*kernel space*) e meio físico de comunicação, conforme ilustrado na figura 5.1.

O *user space* refere-se ao nível da aplicação onde a informação é destinada ou originária. Define uma interface simples de comunicação e interacção com o espaço destinado ao *kernel*. O *kernel space* é onde o sistema operativo Linux executa e proporciona os serviços essenciais. Neste espaço estão implementadas as chamadas ao sistema (*system calls*) através das quais aplicações ao nível do utilizador podem aceder aos recursos existentes no *kernel*. No *kernel space* existe também uma camada responsável pela definição dos protocolos de rede disponíveis, nomeadamente TCP, UDP e IP. Neste espaço estão também definidos os *device drivers*, cujo objectivo é controlar e gerir os dispositivos físicos para a comunicação. Entre estas principais camadas existem interfaces onde estão definidas um conjunto de funções que permitem o acesso e a comunicação entre estas de modo fácil e transparente. Finalmente o terceiro segmento trata-se do dispositivo físico propriamente dito que permite a ligação física das redes.

A estrutura descrita e ilustrada em 5.1 é utilizada pelos sistemas Linux por forma a implementar o modelo da Internet segundo o modelo OSI.

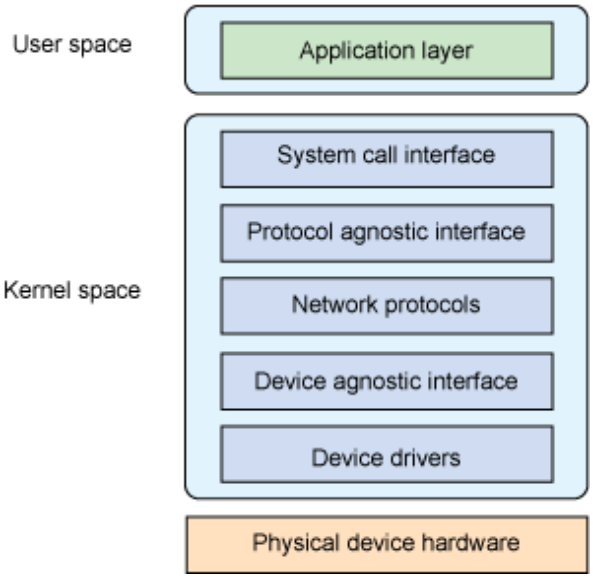



Figura 5.1: Arquitectura da *stack* de rede dos Sistemas Linux [30]

## 5.2 A estrutura *Socket Buffer* (SBK)

A informação circula na *stack* de cada nodo encapsulada numa estrutura designada por *Socket Buffer* (SKB). Sempre que um pacote é recebido pela interface de rede ou é gerado por uma aplicação ao nível do utilizador é criado um *socket buffer*. A estrutura do SKB está ilustrada na figura 5.2.

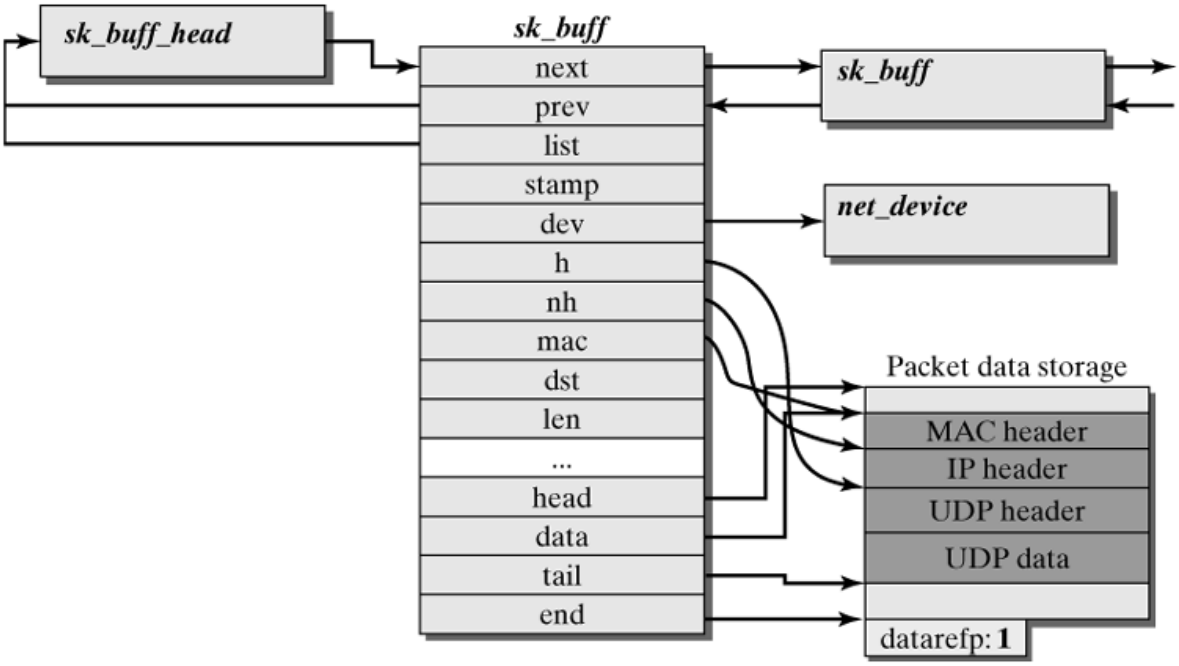



Figura 5.2: Estrutura do *socket buffer* [31]

Os pacotes estão inseridos em filas pelo que na estrutura do SKB existem ponteiros, *prev* e *next*, que indicam o SKB prévio e o SKB seguinte, respectivamente. Outro campo importante na estrutura SKB é o campo *dev* que representa a interface de rede pela qual o pacote chegou ou pela qual será enviado. Neste campo estão definidas informações importantes sobre a interface de rede, como por exemplo o seu nome e o seu endereço MAC.

Os cabeçalhos para as diferentes camadas da *stack* de rede, nomeadamente para a camada de transporte, para a camada de rede e para a camada MAC, estão devidamente inseridos na estrutura SKB e podem ser acedidos através dos ponteiros h (camada de transporte), nh (camada de rede) e mac (camada MAC) conforme ilusta a figura 5.2. A informação propriamente dita está inserida no campo *data*, existindo ponteiros que identificam claramente os limites deste campo. Informações como o tempo em que o pacote foi recebido ou enviado pela interface de rede, ou tamanho do campo de dados do pacote podem ser facilmente identificados na estrutura SKB, ilustrados na figura 5.2 pelos campos *stamp* e *len* respectivamente. Existem alguns campos que não estão ilustrados na figura, como por exemplo o campo *protocol* que indica o protocolo da trama inserida no SKB.

Em termos simples, os campos atrás referidos são os principais da estrutura do *socket buffer*. Tratando-se de uma estrutura importante no que refere à gestão e controlo de informação que circula na rede, estão definidas algumas funções para acesso e manipulação dos SKBs [31].

## 5.3 Caminho dos Pacotes pela *Stack* de Rede

A figura 5.3 ilustra as três últimas sub-camadas do segmento referente ao *kernel space* analisadas anteriormente. Como foi referido, a ligação entre os *device drivers* e a camada de rede é feita através de uma interface (na figura 5.3 designada de *device agnostic interface*) que apresenta funções e métodos que possibilitam uma comunicação fácil e transparente entre estas camadas.

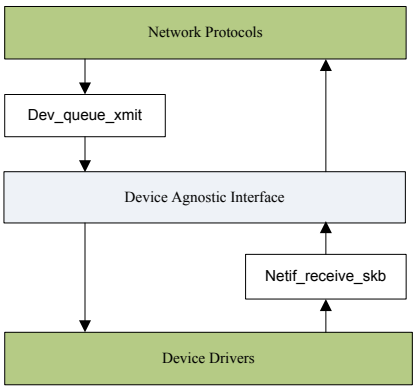



Figura 5.3: Interface entre os *device drivers* e a camada de rede

No contexto desta dissertação, foi necessário perceber como as mensagens são enviadas pela interface de rede e recebidas pela mesma, para mais tarde ser possível acrescentar *software* que permita ter controlo sobre o fluxo de informação na rede.

Existem fundamentalmente duas funções na camada *device agnostic interface* responsáveis por gerir o fluxo de informação: *dev_queue_xmit* para a transmissão e *netif_rx* ou *netif_receive_skb* para a recepção [30].

Quando é necessário enviar uma informação pela interface de rede, o SKB respectivo é colocado numa fila de transmissão ao nível do *device driver* através da função *dev_queue_xmit*. Ao nível do *device driver* responsável pela interface de rede é utilizada *hard_start_xmit* para realizar a transmissão propriamente dita.

Em relação à recepção, quando a trama chega à interface de rede, o SKB respectivo é recebido e gerido através da função *netif_rx* ou *netif_receive_skb* na *New Application Programming Interface* (NAPI). A NAPI foi implementada nas versões 2.5/2.6 dos *kernels* Linux e contribuiu para a diminuição da latência na comunicação, aumentando assim o desempenho das redes. Dependendo do protocolo que as tramas possuem, as funções *netif_rx* ou *netif_receive_skb* vão reencaminhar os pacotes para as respectivas funções de atendimento das camadas superiores - *Protocol Handlers* - da *stack* de rede.

É possível desenvolver e acrescentar dinamicamente um novo *Protocol Handler* aos já existentes através da implementação de um módulo ao nível do *kernel*, situação esta que foi realizada na prática, e que será abordada mais adiante, de modo a gerir e controlar os tráfego que chega às interfaces de rede do sistema computacional.

## 5.4 *Switching* utilizando um módulo no *kernel*

### 5.4.1 *Kernel Modules*

Identificadas as estruturas principais presentes na *stack* de rede, foi desenvolvido um módulo que pode ser inserido e removido dinamicamente num kernel em execução - *Kernel Loadable Module*s (KLMs) ou *Kernel Module*s (KMODs). Deste modo é possível ir acrescentando funcionalidades ao *kernel* sem a necessidade de serem inseridas no arranque do sistema. Existem módulos persistentes que estão já inseridos na imagem do *kernel* e que são carregados quando o sistema inicia.

Um KMOD tem duas funcionalidades obrigatórias: uma função de inicialização do módulo e uma função para remoção do mesmo. A maneira tradicional de definir estas funções é designá-las de init_module() e cleanup_module(), que são reconhecidas automaticamente como as funções de inicialização e de remoção dos módulos, respectivamente. Actualmente, existem mais forma para definir destas funções, onde numa das quais é possível designá-las com um nome arbitrário e utilizar as macros module_init(nome da função de inicialização) e module_exit(nome da função de remoção) para especificar a sua finalidade [32]. Dependendo da funcionalidade desejada para este módulo, podem ser desenvolvidas e acrescentadas novas funções no KMOD para concretizar algum objectivo específico. A extensão dos ficheiros que representam o KMOD é *.ko* nas versões 2.6 dos *kernels* linux, no entanto nas versões antigas a extensão utilizada era *.o*.

Existem alguns comandos importantes para gerir e utilizar o KMOD. O comando *lsmod* permite verificar quais os módulos inseridos, o tamanho deles e o número dos módulos dependentes deste. Para inserir um módulo no *kernel* é utilizado o comando *insmod*, que não analisa as dependências do módulo inserido com o restantes, isto é, se KMOD que queremos inserir necessita que outros módulos sejam carregados em primeiro lugar para funcionar. O comando *modprobe* tem a mesma funcionalidade que o *insmod* mas já analisa as dependências necessárias para o bom funcionamento do KMOD. Para remover módulos inseridos é utilizado o comando *rmmod* e de seguida o nome do módulo a remover, no entanto só são removidos KMOD que não estão em uso ou que não estão a ser utilizados por outros KMOD. A utilização do comando *modprob - r* também permite a remoção do KMOD. Para sabermos informações

sobre o KMOD, como por exemplo o seu autor ou a licença (caso sejam explicitados no código fonte do KMOD), é utilizado o comando *modinfo*. Estas são algumas ferramentas que permitem interagir com o KMOD, perceber se foi devidamente inserido e retirar algumas informações relevantes sobre o mesmo.

Na compilação de KMOD são utilizados vários ficheiros da imagem *kernel*. Frequentemente, algumas distribuições Linux podem ter sido recebido *updates* não *standard*, o que pode causar problemas na compilação do KMOD. Outra questão é que muitas vezes os cabeçalhos para aceder aos ficheiros da imagem do *kernel* estão incompletos e pode não ser possível aceder aos mesmos. Para evitar estes problemas, é normalmente recomendado, inclusive pelo autor de [32], compilar e construir um novo *kernel*. Neste sentido, para desenvolver o trabalho desejado, foi compilado um novo *kernel* para correr no sistema operativo Fedora 14. Pode ser consultado no apêndice A um pequeno tutorial com todos os procedimentos realizados para compilar o *kernel* Linux, sobre o qual foram desenvolvidos os KMODs no contexto desta dissertação.

### 5.4.2 *Protocol Handler* através de um KMOD

Na secção 5.3 referiu-se de modo muito simples o que era um *Protocol Handler*. Como foi abordado, as tramas quando chegam à interface de rede são processadas de maneira diferente, consoante o protocolo que transportam. Para os diferentes tipos protocolares, estão já definidas várias funções que gerem e controlam o caminho da trama até chegar ao destino. Em termos de exemplo, caso chegue uma trama cujo protocolo é *Ethernet 802.3*, a função responsável é a ipx_rcv já definida [33].

É possível adicionar e remover um *Protocol Handler* dinamicamente atraves de KMOD com um conjunto de especificações pré-definidas. Estas especificações são definidas numa estrutura do tipo *packet_type*, explicitada de seguida (retirada do ficheiro *netdevice.h*).

```
struct packet_type {
__be16                  type;
struct net_device       *dev;
int                     (*func) (struct sk_buff *,
                        struct net_device *,
                        struct packet_type *,
                        struct net_device *);
struct sk_buff  *(*gso_segment)(struct sk_buff *skb, int features);
int                     (*gso_send_check)(struct sk_buff *skb);
struct sk_buff  **(*gro_receive)(struct sk_buff **head,struct sk_buff *skb);
int                     (*gro_complete)(struct sk_buff *skb);
void                    *af_packet_priv;
struct list_head        list;
};
```

O campo *type* desta estrutura define o protocolo das tramas sobre o qual o *Protocol Handler* será chamado. No ficheiro *if_ether.h* presente em *include/linux* estão definidas já alguns valores, reconhecidos pelo kernel, que representam os protocolos. Por exemplo, para receber pacotes independentemente do seu protocolo, o campo *type* deve ser preenchido com o valor definido em ETH_P_ALL existente no ficheiro mencionado. Neste caso qualquer tipo de pacote será recebido pelo *Protocol Handler* inserido. É necessário converter este valor para *network short* de modo a ser inserido correctamente na estrutura.

O campo *dev* representa um ponteiro para a interface de rede onde o *Protocol Handler* irá

funcionar, isto é, só serão monitorizados pacotes na interface definida em *dev*. Caso este campo esteja preenchido com NULL, todas as interfaces de rede presentes irão ser monitorizadas pelo *protocol handler* definido.

Sempre que chegar à interface definida em *dev* um pacote cujo protocolo é igual ao definido em *type*, será invocada a função definida no campo *func* da estrutura *packet_type* que realizará o trabalho desejado pelo programador.

Os campos atrás detalhados são os principais para desenvolver um *Protocol Handler*. Estando definida a estrutura *packet_type* é necessário registá-la através da função dev_add_pack cujo o argumento é um ponteiro para estrutura criada. Normalmente, a inicialização e o registo do *Protocol Handler* são realizados na função de inicialização do KMOD por forma a ser inserido assim que o módulo é iniciado. Para remover o *Protocol Handler* é utilizada função dev_remove_packet cujo o argumento é o mesmo da função de registo, e normalmente é invocada na função do remoção do módulo quando este é retirado. É ainda invocada neste processo a função *synchronize_net* para sincronizar a rede e evitar que sejam retidas ligações à estrutura *packet_type* removida.

### 5.4.3 *Switch* - Implementação do Nodo

Os conceitos abordados nas secções anteriores, nomeadamente em 5.4.1 e 5.4.2, foram utilizados no desenvolvimento de um nodo ao nível do *kernel* Linux que reencaminhará os pacotes consoante especificações definidas. A figura 5.4 ilustra o modelo do *switch* implementado.

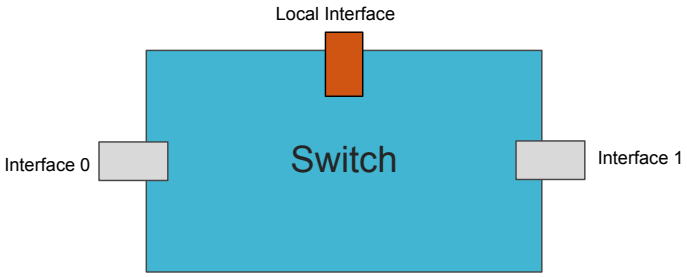



Figura 5.4: Bloco de alto nível do *Switch*

Como pode ser constatado, trata-se de um *switch* com duas interfaces de rede e uma interface local. Os pacotes chegam às interfaces de rede, e caso sejam destinados ao dispositivo ficam no respectivo nodo e continuam a progredir na *stack* de rede, através da sua interface local até atingirem o seu objectivo. Em caso contrário, se os pacotes não têm como destino o *switch* são reencaminhados pela interface de rede oposta à de chegada (pois neste caso são apenas duas). É portanto necessário desenvolver uma unidade de processamento, que implementará os princípios descritos no Capitulo 4 .

Foi definido um *Protocol Handler* capaz de monitorizar todas as interfaces de rede presentes no sistema computacional e captar todos os pacotes independentemente do protocolo que é transportado.

Foi criada uma estrutura estática do tipo *packet_type* e definida da seguinte maneira:

```
static struct packet_type hook;

void handler_add_config()
{
        hook.type=htons(ETH_P_ALL);
        hook.dev=NULL;
        hook.func= hook_func;
        dev_add_pack(&hook);
}
```

A função *handler_add_config* com a especificações definidas é invocada quando o KMOD é inserido. Deste modo, sempre que chegar um pacote a uma das interfaces de rede, este será recebido e tratado pela função *hook_func*, que recebe como parâmetro um ponteiro para uma estrutura do tipo SKB. O funcionamento da função *hook_func* esta ilustrado na figura 5.5.

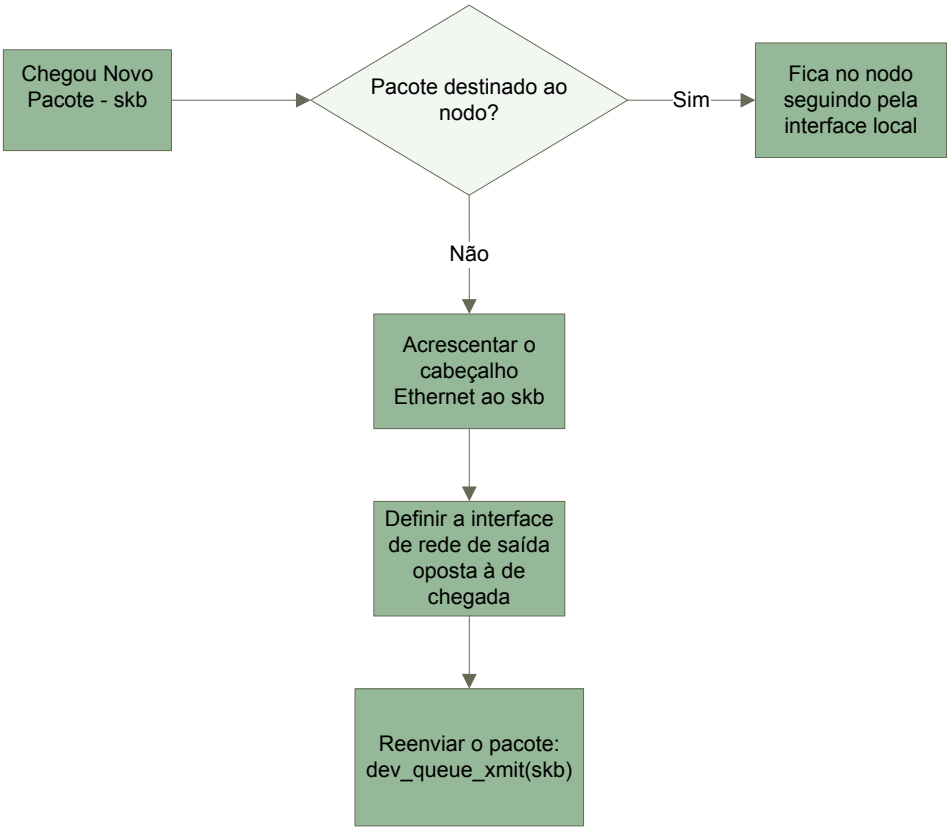



Figura 5.5: *Forwarding* de pacotes implementado

Quando chega um pacote a uma interface de rede cujo protocolo seja 0x88B5, é necessário analisar o primeiro *byte* do *payload* que representa o sub-tipo da trama. Mediante a análise deste valor são realizados os seguintes procedimentos:

- A mensagem de reconhecimento de topologia é gerada pelo *Master* e destinada a este, e portanto, é reencaminhada pelos *Slaves* após a alteração do seu conteúdo conforme, detalhado no Capítulo 4;

- Do mesmo modo, mensagens de reserva de recursos para o envio de informação como também mensagem de remoção de reservas, são apenas processadas pelo nodo *Master* e portanto reencaminhadas pelos restantes nodos;
- Em relação às mensagens de resposta ao pedido de reserva, não é suficiente apenas analisar o sub-tipo da trama. Quando a trama chega ao nodo é necessário comparar o identificador deste com identificador do nodo que fez o pedido. Em caso de correspondência, a trama é recolhida, em caso contrário é reencaminhada.
- Procedimento semelhante é realizado para as mensagens de dados, neste caso sendo comparado o identificador do nodo com o identificador do destino da informação.

Sempre que uma trama é reencaminhada, é necessário realizar alguns procedimentos. Quando um pacote é passado ao nível superior pelo *device driver*, são realizadas algumas operações, em que numa das quais o cabeçalho *Ethernet* é retirado do campo de dados do SKB respectivo. Devido a isto, é necessário acrescentar novamente o cabeçalho da trama *Ethernet* à informação no SKB e actualizar o respectivo tamanho do campo de dados. De seguida é necessário definir a interface de rede de saída, actualizando o campo *dev* do SKB referido na secção 5.1, com a interface desejada. Neste ponto, a estrutura do SKB está redefinida e pronta para ser enviada correctamente para a rede.

## 5.5 Comunicação com o *User Space*

Nesta fase, todas as estruturas e mecanismos para gestão e controlo estão implementados através de um módulo no *kernel*. Tendo em vista uma utilização directa e simples por parte do utilizador, foram acrescentados ainda mecanismos ao KMOD desenvolvido, para que do espaço de utilizador seja possível aceder a essas funcionalidades.

A interface de comunicação padrão entre o espaço do utilizador e o espaço do *kernel* é a utilização de ficheiros, conforme ilustrado pela figura 5.6. É possível registar no KMOD estruturas, designadas de *Character Devices*, que são representadas por ficheiros e que permitem adicionar funções para escrita e leitura de dados. Esta estrutura permite também o uso de uma função designada de *ioctl* (*Input Output Control*), que permite enviar ou receber dados de controlo entre o *kernel* e processos a correr no espaço do utilizador.

No módulo do *kernel* é necessário definir uma estrutura do tipo *file_operations* e preenchê-la com os endereços das funções com a funcionalidade desejada. Entre outras funcionalidades que esta estrutura permite definir, foram utilizadas no contexto deste trabalho apenas as funções *write*, *read* e *ioclt*, ilustradas de seguida.

```
struct file_operations
{
  ssize_t (*read) (struct file *, char *, size_t , loff_t *);
  ssize_t (*write) (struct file *, const char *, size_t , loff_t *);
  int (*ioctl) (struct inode *, struct file *, unsigned int ,   unsigned long);

...
 };
```

A função *write* tem o objectivo de escrever dados em variáveis do *kernel* a partir do espaço do utilizador. O objectivo da função *read* é a leitura do conteúdo de uma variável no *kernel* no espaço do utilizador. Em relação à função *ioctl*, apresenta funcionalidades de leitura e

escrita, dependendo da maneira de como é invocada, sendo no entanto como já foi referido, utilizada para controlo de processos.

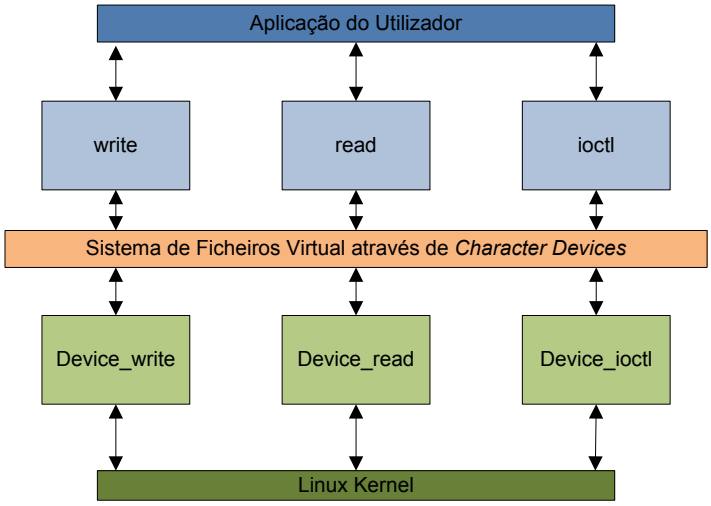



Figura 5.6: Interface entre o Utilizador e o *Kernel* Linux

É portanto necessário implementar as funções e registá-las conforme é explicitado em seguida.

```
static struct file_operations fops

void config_file(void)

{
        fops.write=device_write;
        fops.real=device_read;
        fops.ioctl=device_ioctl;

        if(register_chrdev(Major_Number,Device_Name,&fops)<0)
        printk("Error, canno't register Character Device");

};
```

A função *config_file* é invocada no arranque do KMOD. O registo de uma estrutura *Character Device* é realizado através da função *register_chardev*. O primeiro parâmetro desta função é o *Major Number*, e tem como objectivo identificar o *driver*. Deste modo, no acesso a um ficheiro especial, o *kernel* vai procurar o respectivo *Major Number* associado e aceder às respectivas funções definidas. Em seguida, é definido o nome do ficheiro que fará a interface entre o *kernel* e o espaço de utilizador. Por fim, a variável *fops* representa a estrutura *file_operations* onde são especificados os endereços para as funções que podem ser invocadas.

No espaço do utilizador, quando for invocada a função *write* associada ao ficheiro especial *Device_Name*, irá ser chamada a função *device_write* ao nível do *kernel*. A função *device_write*, no contexto deste projecto, tem o objectivo de receber os dados previamente do utilizador, fazer as verificações e validações necessários de acordo com as informações registadas nas tabelas de mensagens, e enviar informação para a rede. O utilizador tem de ter o cuidado de enviar os dados de acordo com a estrutura para as mensagens de dados, conforme definido na secção 4.4.4

Da mesma forma, a função *device_read* é invocada no *kernel* quando for chamada a função *read* pelo utilizador associada ao ficheiro especial pré-definido. O objectivo desta função é a leitura dos dados de uma trama *Ethernet*, quando esta chega ao destino, enviando-os deste modo para o utilizador. É sempre necessário ter em atenção como são recolhidos os dados, respeitando as estruturas definidas pois proporcionam uma interface simples e directa no acesso à respectiva informação.

Finalmente, a função *ioctl* utilizada no espaço do utilizador associado ao ficheiro especial definido, invoca a função *device_ioctl* no *kernel*. É utilizada neste espaço para trocar mensagens de controlo, nomeadamente para fazer pedidos de registos de mensagens e para remoção de mensagens da rede. Esta função é invocada utilizando três parâmetros: o descritor do ficheiro utilizado, um número da *ioctl* que permite definir vários tipos de funcionalidades, e por fim os dados a enviar. Este número da *ioctl* é criado através do uso de macros, onde são tidos em conta o *Major Number* definido, o comando a ser executado e a estrutura da informação que se pretende enviar [32]. Exemplos de macros utilizadas são a _IOW, para o envio de informação do utilizador para o *kernel* e _IOR, para receber informação do *kernel* no espaço do utilizador. Neste sentido, foram definidas duas variáveis, que têm de estar inseridas tanto no programa do utilizador como no KMOD, usadas na invocação da função *ioctl* para fazer o pedido de envio de dados e para remoção de mensagens na rede, conforme exemplificado de seguida.

```
#define REQUEST_TO_SEND_MSG _IOW(Major Number,0, struct Request_frame*)
#define STOP_MSG _IOW(Major Number,1, int)
```

Os parâmetros enviados pelo *ioctl*, invocada ao nível do utilizador, no pedido de envio de uma nova mensagens têm estar mais uma vez de acordo com a estrutura estabelecida, de modo a facilitar o envio da informação e a sua validade, evitando assim um envio incorrecto dos parâmetros. Para remoção, é só enviado o identificador da mensagem, e caso esta esteja registada nas tabelas, os respectivos campos da estrutura pré-definida são depois preenchidos ao nível do *kernel*. O *Character Device* é removido através do uso da função *unregister_chardev*, chamada quando o KMOD é retirado.

## 5.6 Sumário

Neste capítulo foi realizada uma breve abordagem à pilha protocolar de rede dos sistemas Linux. Conforme foi referido, as tramas circulam na rede encapsuladas na estrutura de um *socket buffer* que permite uma melhor abstracção da informação ao longo da *stack* de rede. É possível criar módulos ao nível do *kernel* dos sistemas Linux e adicionar mecanismos para manipulação dos pacotes que chegam às interfaces de rede, nomeadamente através da definição de *Protocol Handlers*.

Através da implementação de um KMOD onde foi definido um *Protocol Handler* para capturar tramas *Ethernet*, foi desenvolvido um *switch* em *software* com os princípios de funcionamento pretendidos e explicados no Capítulo 4.

É possível comunicar do espaço do utilizador com o *kernel* através de interfaces definidas por ficheiros. Neste capítulo fez-se uma breve abordagem sobre o mecanismo implementado para proporcionar o acesso às funcionalidades do *switch* implementado no KMOD através de uma aplicação ao nível do utilizador.

# Capítulo 6

# Resultados Experimentais

Este capítulo tem o objectivo de apresentar os resultados experimentais da implementação realizada. São tiradas algumas conclusões em primeiro lugar sobre o funcionamento do mecanismo de *forwarding* dos pacotes e em seguida sobre o processo de gestão topologia e da largura de banda em uso da rede implementada.

## 6.1 Reencaminhamento de Pacotes

Após a implementação do *switch*, analisada no Capítulo 5, foram desenvolvidos diversos testes com o objectivo de validar o funcionamento do nodo bem como ter algumas noções sobre o seu desempenho temporal.

Nas experiências reportadas neste capítulo foram usados nodos Linux com as seguintes características :

- Processador Intel(R) Pentium 4, com uma frequência 2.40 $GHz$;
- Umas das interfaces de rede é uma *Realtek Semicondutor Ethernet* 8139, com suporte a *Fast Ethernet* e funcionamento a *Half/Full duplex*;
- A outra interface de rede é da família IntelCorporation 82557, também com suporte a *Fast Ethernet* e funcionamento a *Half/Full Duplex*.

Todas as ligações estão configuradas para funcionar em modo *full-duplex*, têm o mecanismo de *auto-negotiation* activado e estão a operar a 100 $Mbps$.

Inicialmente foi desenvolvido um módulo capaz de realizar a funcionalidade miníma de um *switch* que é reenviar pacotes segundo um critério estabelecido. Foi possível verificar, após vários testes intensivos, que o KMOD desenvolvido recebia e era capaz de gerir pacotes de vários tipos, entre os quais pacotes *Ethernet*, IP e/ou ARP. Nomeadamente, foi possível transferir ficheiros entre dois computadores interligados pelo nodo desenvolvido através da criação de uma sessão ssh.

De modo a analisar o tempo de *turn-around* dos pacotes na rede, aliado ao nodo desenvolvido responsável pelo *switch* foi desenvolvido outro KMOD com os procedimentos semelhantes, cuja função é reenviar de volta o pacote recebido (designado como *echo* do pacote). Assim é possível ter uma noção da forma como se comporta a rede em termos temporais e a influência do número de *switches* inseridos na mesma. Utilizou-se um computador para

monitorizar os pacotes enviados e recebidos. Com auxílio do *PackEth* [34], *software* grátis para gerar pacotes, é possível criar uma trama *Ethernet* e enviá-la por uma interface definida de modo simples e intuitivo. Por forma a monitorizar os pacotes enviados e recebidos, foi utilizado o *Wireshark* [35] que se trata de um analisador de redes.

### 6.1.1 Echo Directo


Figura 6.1: *Echo* Directo entre Computador e Módulo

Numa primeira fase, foram enviados pacotes directamente para o nodo responsável pelo *echo*, conforme está ilustrado na figura 6.1. Devido a limitações físicas das interfaces de rede, os pacotes foram enviados com um espaçamento entre eles de 3 $ms$, pois deste modo foi possível minimizar, e até mesmo anular, a perda de pacotes.

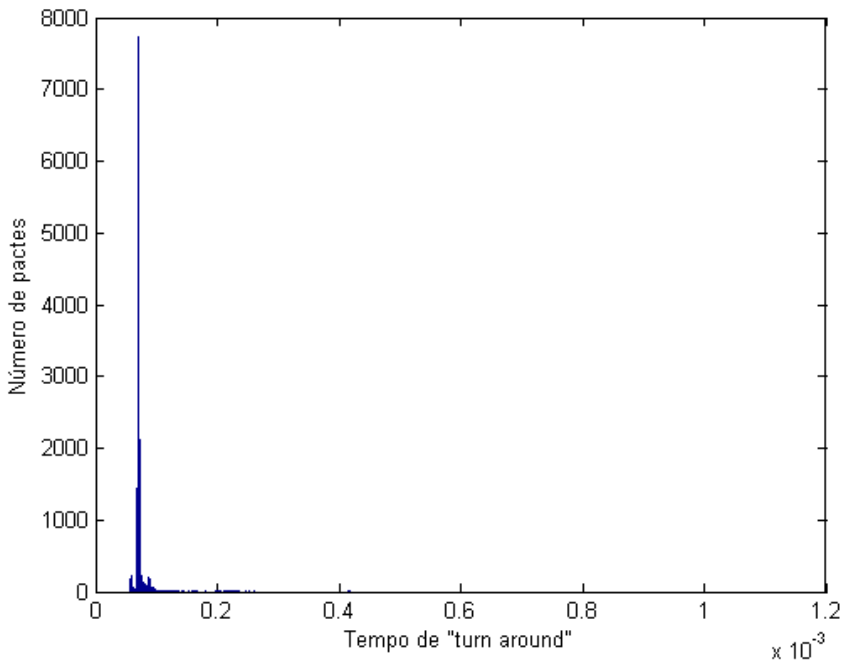



Figura 6.2: Histograma *Echo* Directo entre Computador e Módulo

Foram enviados 20000 pacotes *Ethernet* de tamanho 64 *bytes* (incluindo o cabeçalho). Com auxílio do *Wireshark* foram captadas e guardados em ficheiros diferentes as tramas

enviadas e recebidas, que depois foram tratadas usando o *Matlab* (através do código presente no apêndice B). Deste modo foi possível calcular o tempo de *turn-around* das tramas e calcular alguns valores de interesse. A figura 6.2 ilustra o histograma, na situação representada pela figura 6.1, que se refere à situação de ter um computador a enviar pacotes e outro a reenviá-los de volta. Este estudo foi realizado várias vezes e os resultados obtidos foram sempre semelhantes.

| Mínimo | Máximo | Média | Desvio Padrão |
|---|---|---|---|
| 5,700e-5 | 41,19e-5 | 7,256e-5 | 1,168e-5 |

Tabela 6.1: Tempo de *turn-around*, em segundos, no caso sem nodos intermédios - *Echo* Directo

Podemos analisar pelo histograma representado na figura 6.2 que existe uma distribuição normal do tempo de *turn around* dos pacotes. É visível uma média temporal bem definida, na ordem das dezenas de µs (72,56µs) com um desvio padrão bastante reduzido. Através deste teste é possível estudar e ter uma noção sobre a duração da comunicação.

Os dados estatísticos obtidos, referentes à figura 6.2, estão repetidos na tabela 6.1, para uma análise mais clara.

### 6.1.2 Echo com Nodos *Switch* Intermédios

De seguida foram sendo acrescentados nodos *switch* intermédios e testes semelhantes foram realizados (enviados pacotes de 64 *bytes* espaçados de 3 *ms*), conforme ilustra a figura 6.3 para o caso de um nodo intermédio.

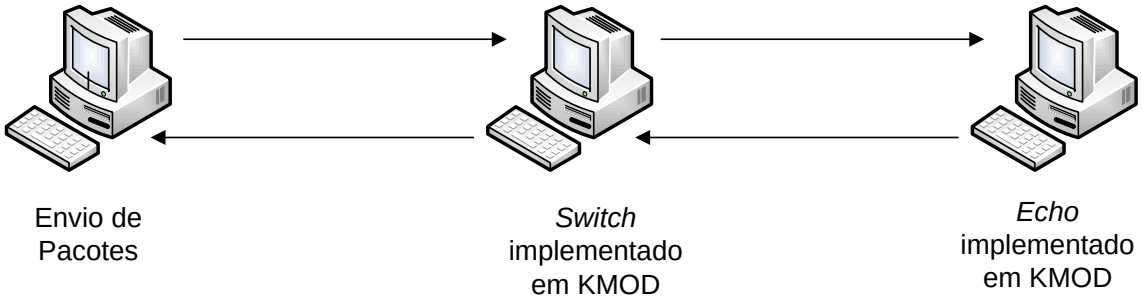


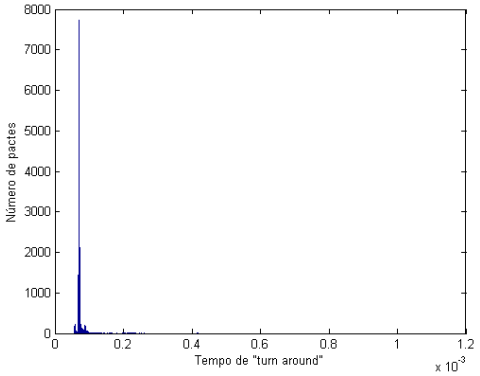

Figura 6.3: *Echo* com um *Switch* Intermédio

É de esperar que as distribuições sejam semelhantes à ilustrada em 6.2 deslocada um pouco para a direita, pois como os pacotes passam por nodos extra, o caminho a percorrerem na rede é maior e portanto a duração de ida e volta será também maior. Na figura 6.4 estão representadas as distribuições para os casos analisados e na tabela 6.2 são expostos, de modo mais claro, alguns valores estatísticos importantes para as mesmas situações.

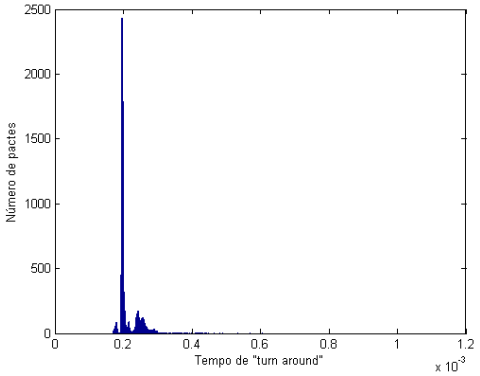
Podemos então verificar, através da análise da figura 6.4 em conjunto com a tabela 6.2, um aumento do tempo médio de *turn around* dos pacotes, como era previsto, bem como dos valores mínimos e máximos. O desvio padrão vai tendo um ligeiro aumento à medida que vão sendo acrescentados nodos à rede, mantendo no entanto a mesma ordem de grandeza. Como poder ser constado pelo histograma ilustrado na figura 6.4(b) é notória a existência de picos temporais mais salientes face ao valor médio, comparativamente com o histograma da figura

| Número de Nodos Intermédios | Mínimo | Máximo | Média | Desvio Padrão |
|---|---|---|---|---|
| 0 | 5,70e-5 | 41,90e-5 | 7,256e-5 | 1,168e-5 |
| 1 | 16,90e-5 | 60,60e-5 | 21,04e-5 | 2,811e-5 |
| 2 | 30,05e-5 | 76,50e-5 | 37,39e-5 | 4,628e-5 |
| 3 | 44,80e-5 | 100,0e-5 | 56,13e-5 | 4,246e-5 |

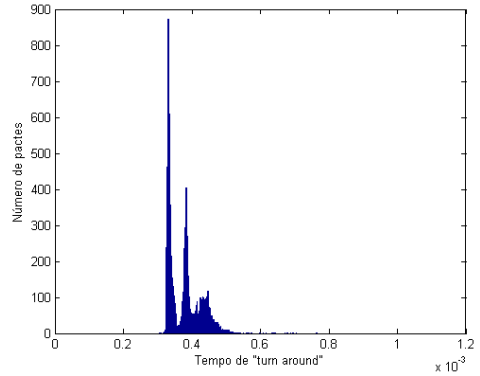
Tabela 6.2: Valores estatísticos, em segundos, do *turn-around* para diversos nodos intermédios ilustradas na figura 6.4



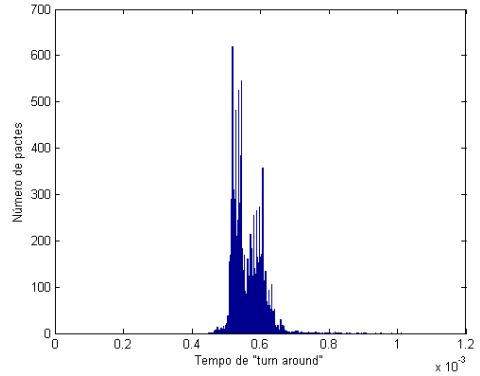

(a) Histograma *Echo* Directo entre Computador e Módulo




(b) Histograma *Echo* com um Nodo *Switch* intermédio




(c) Histograma *Echo* com dois Nodos *Switch* intermédios




(d) Histograma *Echo* com três Nodos *Switch* intermédios

Figura 6.4: Várias distribuições relativas ao tempo de *turn-around* com vários nodos *Switch* Intermédios onde foram enviados 20000 pacotes de 64 *bytes* espaçados de 3 $ms$.

6.4(a). Estes picos vão ficando mais relevantes e distintos à medida que vai adicionando nodos intermédios, como compravam as figuras 6.4(c) e 6.4(d). Verificou-se através realização de vários medições, que com o aumento do intervalo de tempo entre o envio de pacotes, estes picos continuavam presentes mas no entanto já com menor amplitude. Nos testes realizados os atrasos foram sendo variados entre 5 a 10 $ms$. O envio muito rápido de pacotes para a rede pode saturar a mesma, fazendo com que alguns pacotes se atrasem. Como à medida que são inseridos mais nodos intermédios o caminho a percorrer pelos pacotes é maior, mais

tráfego circula na rede e portanto o tempo de ida e volta dos pacotes pode desviar-se do valor médio mais vezes. Como foi referido, o sistema foi implementado em computadores com o sistema operativo Linux. Nestes sistemas computacionais existem processos que são executados periodicamente e de modo concorrente o que pode originar uma maior variação do atraso dos pacotes, no entanto esta hipótese não foi devidamente validada em termos práticos.

Com os valores médios do tempo de ida e volta dos pacotes, foi realizada uma regressão linear (figura 6.5) e calculada a correlação desta recta com os valores medidos. O valor da correlação obtido foi de 0.997, o que indica para o efeito dos testes realizados numa rede em que o número máximo de nodos presentes foram cinco - um nodo a enviar pacotes, outro nodo a reenviá-los de volta e três nodos intermédios - o tempo de ida e volta dos pacotes cresce de uma maneira aproximadamente linear.

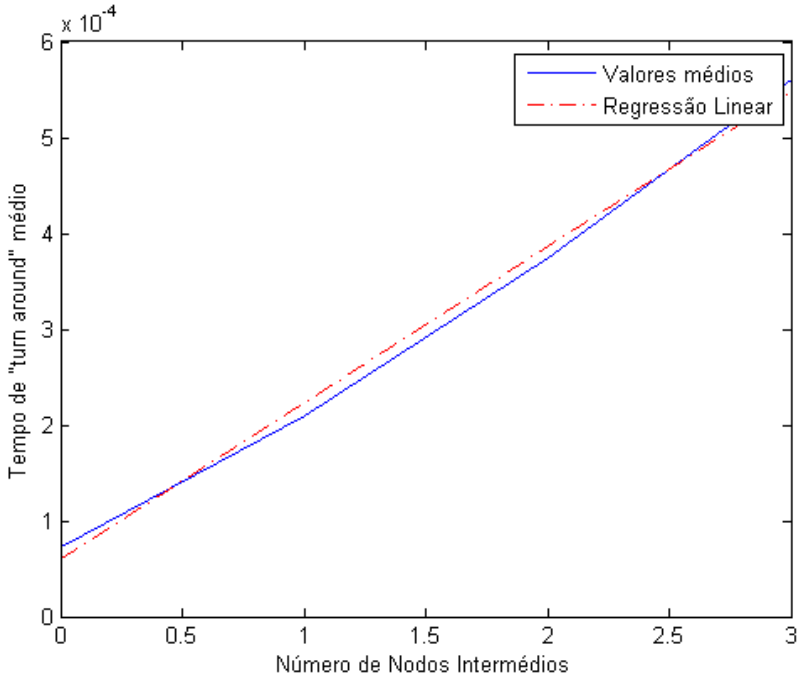



Figura 6.5: Comparação do tempo de "turn around"médio com a regressão linear

Foi também analisada a influência do tempo *de turn around* com o tamanho das tramas enviadas. Como foi referido, nos casos acima analisadas, foram enviadas tramas *Ethernet* cujo tamanho era de 64 *bytes* (14 *bytes* do cabeçalho *Ethernet* mais 50 *bytes* do *payload*). Testes semelhantes foram realizados com tramas de 1038 *bytes* sendo portanto 14 bytes para o cabeçalho *Ethernet* e os restantes para os dados. A formato dos histogramas bem como o comportamento dos tempo de *turn around* dos pacotes é em tudo semelhante aos casos analisados anteriormente. No entanto, e conforme esperado, os pacotes demoram mais tempo a serem enviados porque têm um tamanho superior, e portanto há mais informação para ser transmitida, o que consequentemente aumenta os tempos de ida e volta dos pacotes. Em termos de exemplo a tabela 6.3 mostra os tempo de *turn around* de pacotes na situação onde existem dois nodos intermédios. Como se pode constatar a ordem de grandeza é semelhante nos dois casos ilustrados, mas no entanto as tramas de maior dimensão apresentam tempos maiores.

| Tamanho das tramas (em *bytes* ) | Mínimo | Máximo | Média | Desvio Padrão |
|---|---|---|---|---|
| 64 | 3,050e-4 | 7,650e-4 | 3,739e-4 | 4,628e-5 |
| 1038 | 8,200e-4 | 13,000e-4 | 8,870e-4 | 3,640e-5 |

Tabela 6.3: Valores estatísticos, em segundos, com dois nodos intermédios e tramas com diferentes tamanhos

Foram realizados mais testes alterando outros parâmetros, nomeadamente enviando 30000, 40000 e 50000 pacotes nas experiências realizadas, sendo os resultados obtidos na linha do que foi demonstrado.

O trabalho realizado teve o objectivo de validar o nodo *switch* desenvolvido em termos funcionais, o que foi conseguido. Permitiu também ajudar a conhecer a ordem de grandeza do tempo de envio dos pacotes pela rede, neste caso para uma rede constituída por cinco nodos (caso limite implementado). No seguimento da linha desta dissertação, o passo seguinte foi elaborar uma rede, baseada em nodos *switch* desenvolvidos, numa topologia em anel e acrescentar mecanismos de gestão e controlo que permitam a sua utilização de aplicações periódicas e esporádicas de tempo real.

## 6.2 Mecanismo de Gestão da Rede

Com o objectivo de analisar e de validar os métodos de gestão e controlo da rede descritos ao longo do Capítulo 4, foi projectada uma rede com quatro nodos, sendo um deles o *Master* e os restantes *Slaves*, como ilustrado na figura 6.6.

Os *Slaves* são inseridos em primeiro lugar na rede sendo depois colocado o *Master* que envia a mensagem para reconhecimento da topologia. Foram realizados vários testes com o objectivo de confirmar se esta mensagem é gerada correctamente e se os procedimentos executados pelos respectivos nodos estão de acordo com o esperado. Neste sentido, pôde-se constatar que:

- Cada vez que o nodo *Master* é inserido na rede, é gerada e enviada correctamente a mensagem para reconhecimento da topologia;
- Se os *Slaves* estiverem já presentes na rede, recebem a mensagem, acrescentam o seu identificador e de seguida reenviam-na pela interface oposta à da recepção;
- O nodo *Master* ao voltar a receber a mensagem de reconhecimento de topologia cria a tabela para controlar a largura de banda de cada ligação entre os nodos na rede;
- Caso os *Slaves* não estejam inseridos, não é recebida nenhuma informação no *Master* sobre a rede e portanto nada acontece;

Após a validação da capacidade de levantamento dos nodos na rede é possível enviar tráfego. Como foi referido, para um *Slave* enviar informação para um ponto da rede necessita de fazer um pedido que será sujeito a um controlo de admissão por parte do nodo *Master*.

No sentido de validar o controlo de admissão por parte do *Master* foram realizados vários pedidos por parte dos *Slaves*. As mensagens para o pedido de reserva de recursos são geradas correctamente com todos os campos relevantes, explicitados no Capítulo 4, bem definidos. A tabela onde estão registadas as informações que circulam na rede foi criada para apenas

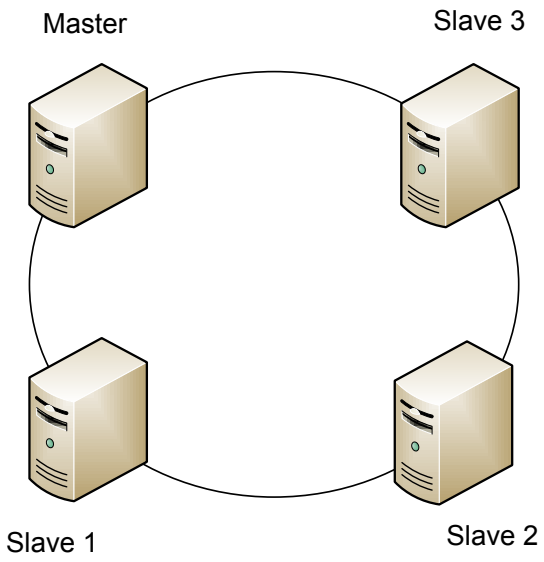



Figura 6.6: Rede desenvolvida com quadro nodos

aceitar 100 mensagens, mesmo que exista ainda largura de banda disponível para registar mais. Para confirmar esta situação a rede foi saturada com 100 pedidos para reserva de recursos, sendo estes todos aceites e registados na tabela. Em termos concretos, recorrendo à figura 6.6, o *Slave 3* requisitou ao *Master* o envio de 100 mensagens para o *Slave 2* onde cada uma iria utilizar $500Kbps$. Como se pode constar a largura de banda utilizada no total seria de 50 $Mbps$, o que não excede os 100 $Mbps$ das ligações. Em seguida, o *Slave 3* tentou registar uma nova mensagem, mas como a tabela já não tem entradas disponíveis, este pedido foi rejeitado.

Analisada esta primeira verificação realizada pelo nodo *Master*, mais testes foram realizados para verificar o funcionamento do controlo de admissão deste nodo. Como foi referido no Capítulo 4, as mensagens têm um identificador único que as representa, e portanto não existem mensagens com o mesmo identificador na rede. Recorrendo à figura 6.6 o *Slave 2* registou uma mensagem cujo identificador era o valor 1. De seguida, o *Slave 3* requisitou a reserva de recursos para uma mensagem com o mesmo identificador. O nodo *Master*, apesar de existirem recursos disponíveis na rede, rejeitou o pedido pois o identificador 1 já se encontrava em uso. Deste modo comprovou-se um dos requisitos impostos pelo nodo *Master* na reserva de recursos.

De seguida foram realizados vários testes de modo a perceber se a largura de banda ocupada por cada mensagem é correctamente analisada e se o caminho pelo qual as mensagens vão circular na rede foi escolhido de acordo com as especificações definidas no Capítulo 4. Relembrar que as ligações consideradas têm uma largura de banda máxima de 100 $Mbps$. Os testes para validar o controlo de admissão foram realizados como no exemplo exposto de seguida. O *Slave 3* registou junto do *Master* 10 mensagens, cada uma a necessitar de 20 $Mbps$, a enviar para o *Slave 2*. Tendo como referência a figura 6.7 que ilustra a experiência realizada, as primeiras 5 mensagens foram aceites e ficou registado que seriam enviadas através da ligação *c*. As restantes 5 mensagens foram também aceites e que seriam utilizadas as ligações *d-a-b* para as mensagens chegarem ao destino. O *Slave 3* tentou ainda registar mais

uma mensagem que iria necessitar de 20 *Mbps*, mas foi rejeitada pelo *Master*. Estes testes permitiram verificar que:

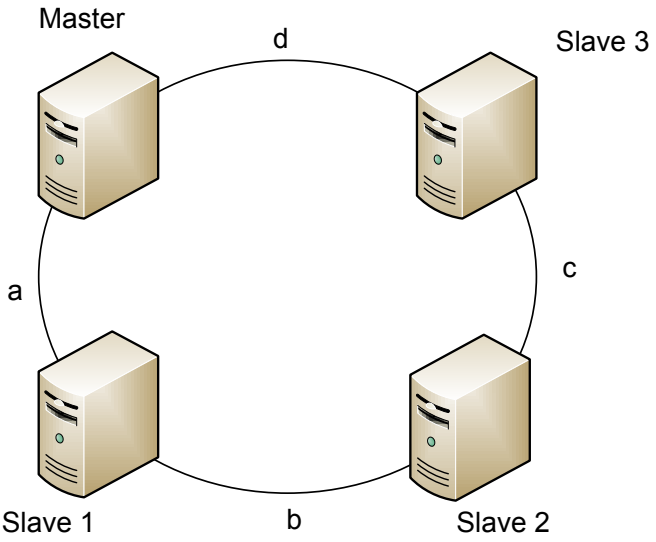



Figura 6.7: Rede desenvolvida com quadro nodos

- A largura de banda nas duas direcções existentes entre a origem e o destino é correctamente analisada:
  - Sempre que existe largura de banda suficiente em ambas as ligações, o pedido é aceite e regista-se a ligação que minimiza o número de nodos;
  - Foi verificado também que sempre que uma ligação se encontra saturada, a informação é enviada pela ligação contrária;
  - Quando não existe largura de banda suficiente em ambos os sentidos do anel para acomodar a mensagem, o pedido é rejeitado.
- Após o controlo de admissão, as respostas que o *Master* envia aos *Slaves* espelham correctamente a decisão tomada.

Nesta fase, foi verificada a capacidade de pedido para reserva de recursos por parte dos *Slaves* e uma análise correcta por parte do *Master* sobre a disponibilidade actual da rede na resposta enviada. Os pedidos aceites são devidamente registados na tabela do *Master* e a largura de banda que será utilizada é devidamente actualizada nas respectivas ligações entre os nodos.

Os *Slaves* em caso da resposta do *Master* ser afirmativa registam na sua tabela as características da mensagem que vão enviar. Como foi referido no Capitulo 4, os *Slaves* fazem um policiamento do tráfego por eles enviado com o objectivo de confirmar se os requisitos acordados para a mensagem são cumpridos. Quando se pretende enviar uma mensagem para a rede é necessário confirmar se o seu identificador se encontra na tabela. Verificou-se que sempre que se tentava enviar informação para a rede que não estava registada no respectivo *Slave* esta era bloqueada e portanto não enviada, sendo o utilizador alertado para esta situação.

Outra das características analisadas pelos *Slaves* antes de enviarem a informação é se o tamanho do campo de dados coincide com o acordado. Neste sentido, foram registadas algumas mensagens e tentou-se de seguida enviar para a rede dados com cujo tamanho do campo de dados não coincide com o que foi estipulado. Em termos mais concretos, recorrendo à rede ilustrada na figura 6.7, o *Slave 3* reservou recursos no *Master* para o envio de uma mensagem com as seguintes características:

- O identificador da mensagem é 10;
- A periodicidade é de 1 $ms$;
- O campo de dados é composto por 1000 *bytes*.

Em seguida, tentou-se enviar a mensagem registada com o identificador 10 com 1500 *bytes* de dados, sendo rejeitada. Foi experimentado também enviar informação na mesma mensagem mas agora com 500 *bytes*, situação esta que foi validada. Com a realização de testes semelhantes ao exemplificado foi possível verificar e validar parte do policiamento realizado pelos *Slaves* no envio de pacotes para a rede. Verificou-se que, nos casos em que as mensagens tinham um campo de dados superior ao registado, não eram enviadas. Deste modo, o uso de recursos por parte de cada mensagem é limitada à reserva efectuada. Sempre que o tamanho das mensagens não excedia o estabelecido eram enviadas, sendo no entanto o utilizador alertado para esta situação.

Antes de enviar as informações, os *Slaves* precisam também de controlar se as mensagens estão a ser enviadas de acordo com a periodicidade pré-estabelecida. Para verificar esta situação, foram realizadas várias experiências onde se variava a periodicidade com que se enviava a informação. Nesse sentido, e recorrendo ao exemplo anterior e às características de mensagem com identificador 10 registada pelo *Slave 3*, tentou-se enviar a informação associada a essa mensagem com um período de 0.5 $ms$, situação esta que não foi permitida. Tentou-se também enviar a mensagem com identificador 10 com uma periodicidade de 2 $ms$, que neste caso foi enviada. Experiências semelhantes foram realizadas e permitiram verificar que sempre que se tentava enviar informação referente a uma mensagem registada com período inferior ao estipulado esta era impedida. As mensagens cujo o período era superior ao pré-definido eram enviadas, alertando o utilizador para o atraso respectivo.

A libertação de recursos referentes a mensagens enviadas pelos *Slaves* foi também devidamente testada e validada. A tabela do *Slave*, caso a mensagem esteja lá registada, é devidamente actualizada removendo-se a entrada para a respectiva mensagem e de seguida é enviada uma mensagem ao *Master* para actualizar as suas tabelas. Neste nodo, toda a largura de banda é actualizada correctamente nas respectivas ligações e a mensagem é removida da rede.

Em jeito de conclusão, a rede foi saturada com as diversas mensagens de controlo e informação tendo em vista a análise e a validação do comportamento desta. Os vários testes permitiram verificar o correcto funcionamento tanto do controlo de admissão por parte do nodo *Master* como também da política de policiamento do tráfego enviado pelos *Slaves*.

# Capítulo 7

# Conclusões e Trabalho Futuro

Neste capítulo é realizada a conclusão da dissertação, enquadrando os resultados alcançados com os objectivos pretendidos. É feito um comentário sobre possíveis características que poderão ser inseridas no sistema bem como outros ambientes de desenvolvimento a utilizar em trabalhos futuros.

## 7.1 Conclusões

O objectivo deste projecto era o desenvolvimento de uma infraestrutura baseada na tecnologia *Ethernet* para ser utilizada em sistemas embutidos distribuídos de tempo-real. Neste sentido, foram desenvolvidos nodos dotados de algumas características necessárias para atingir o objectivo proposto:

- Foi projectada uma rede baseada na topologia em anel segundo o paradigma *Switched Ethernet* e cada nodo da rede integra um *switch* de dois portos;
- Utilizaram-se computadores com o sistema operativo Linux no desenvolvimento dos nodos;
- Os nodos foram desenvolvidos em módulos dinâmicos que podem ser inseridos no *kernel* Linux;
- O mecanismo de reencaminhamento de pacotes foi implementado com sucesso nos nodos;
- Foram desenvolvidos mecanismos de gestão de topologia;
- Foram implementados processos responsáveis por registar e controlar tráfego periódico e esporádico de tempo real;
- Realizaram-se diversas experiências com o objectivo de verificar e validar o correcto funcionamento dos métodos implementados;
- Foi criada um interface para comunicação entre o utilizador e o módulo executado no *kernel*.

Apesar de não suportar todo o tipo de tráfego que normalmente existe nas redes de hoje em dia, foi concebida uma infraestrutura baseada na tecnologia *Ethernet* e segundo a metodologia *Switched Ethernet*, onde é possível um controlo e gestão de tráfego com características de tempo real.

## 7.2 Trabalho Futuro

O trabalho desenvolvido deixou em aberto algumas ideias interessantes que podem ser aplicadas em investigações futuras.

Aliado aos mecanismo para gestão de tráfego periódico e esporádico de tempo-real, era interessante dotar a rede elaborada com a capacidade para acomodar outros tipos de tráfego, nomeadamente tráfego sem requisitos temporais, e com técnicas para que este não comprometa o tráfego mais prioritário como é o caso de tráfego de tempo real.

Seria também interessante acrescentar redundância, nomeadamente através a existência de um outro anel, transparente ao utilizador, dotando a rede de uma maior tolerância a falhas.

Adoptar um modelo de gestão dinâmica de tráfego e da topologia da rede, é também um mecanismo aliciante para este tipo infraestruturas.

Uma vez que a infraestrutura apresentada neste documento apenas dá suporte à comunicação *unicast*, era interessante adicionar mecanismos que permitam também a comunicação em *broadcast* e em *multicast*.

Outra abordagem bastante interessante e a considerar, é a utilização de FPGAs para implementação dos nodos *switch*. As FPGAs são dispositivos cada vez mais utilizados em ambientes industriais e em sistemas que necessitam de cumprimentos de *deadlines* restritos, devido à sua grande capacidade de processamento e versatilidade.

# Bibliografia


[1] P. Pedreiras, L. Almeida, and J. A. Fonseca, "The quest for real-time behavior in ethernet," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–14.

[2] B. Z. Dias and N. A. Jr., "*Evolução da Ethernet*," May 2002.

[3] A. S. Tanenbaum, *Computer Networks.* Amsterdam: Prentice-Hall, Inc., 1996.

[4] A. Moreira. (1998,revisto em 2002) Redes Ethernet (IEEE 802.3). [Online]. Available: http://www.dei.isep.ipp.pt/~andre/documentos/ethernet.html

[5] (Acessed:Outubro, 2011) (IEEE 802.3). [Online]. Available: http://pt.wikipedia.org/wiki/IEEE_802.3

[6] H. W. Johnson, *FastEthernet Dawn of a New Network.* Prentice-Hall, Inc., 1996.

[7] A. N. Pinto, *Apontamentos da cadeira Redes e Telecomunicações*, Universidade de Aveiro, 2010.

[8] D. Griffiths and M. Hein, *Switching Technology in the Local Network.* Thompson Computer Press, 1997.

[9] *The Ethernet,A Local Area Network, Data Link Layer and Physical Layer Specifications.* Digital Equipament Corporation, Intel Corporation, Xerox Corporation, September 1980.

[10] J. Gouveia and A. Magalhães, *Redes de Computadores.* FCA - Editora de Informática, Julho 2005.

[11] M. Norris, *GigaBit Ethernet - Techonoloy and Applications.* Artech House, Inc., 2003.

[12] M. Jones, *White Paper - Ethernet Driving Down Automotive Cost of Ownership, Investigating the Emergence of Ethernet in Automotive Applications.* Senior FAE, Micrel Inc., October 2008.

[13] J. M. S. Pinheiro, "Switches em redes locais de computadores," March 2005.

[14] Cisco. (2007, August) *How LAN Switches Work*, document id: 10607. [Online]. Available: http://www.cisco.com/application/pdf/paws/10607/lan-switch-cisco.pdf

[15] ——. (1998) *Using VlanDirector, Appendix C - Understanding Spanning-Tree Protocol*. [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/sw_ntman/cwsimain/cwsi2/cwsiug2/vlan2/stpapp.htm

[16] D. P. Black, *Building Switched Networks: Multilayer Switching, QoS, IP Multicast,Network Policy and Service Level Agreements*, 1999.

[17] D. Networks. (2011, November) http://www.daxnetworks.com. [Online]. Available: http://www.daxnetworks.com/Technology/TechDost/TD-110806.pdf

[18] SysKonnect, *White Paper - Link Aggregation according to IEEE 802.3ad.*

[19] (Acessed:February, 2012) Ethernet powerlink. [Online]. Available: http://www.ethernet-powerlink.org/

[20] R. Marau, *Real-time communications over switched Ethernet supporting dynamic QoS management.* PhD thesis, University of Aveiro, December 2009.

[21] C. Venkatramani and T. Chiueh, *Supporting Real-Time Traffic on Ethernet.* In Proceedings of the 15th IEEE International Real-Time Systems Symposium, December 1994.

[22] P. Pedreiras, S. Schoenegger, L. Seno, and S. Vitturi, *"Ethernet Powelink". Book chapter of The Industrial Electronics Handbook*, 2nd ed., Bodgan Wilamowski, Auburn University, Alabama, USA and J. David Irwin, Auburn University, Alabama, USA, Eds., February 28 2011.

[23] M. Popp, J. Feld, and R. Büsgen, "Principles and features of profinet." in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/IITHandbook2005.html#PoppFB05

[24] (Acessed:February, 2012) "High level Ethernet protocol". HSM Industrial Network. [Online]. Available: http://www.anybus.com/technologies/profinet.shtml

[25] (Acessed:February, 2012). [Online]. Available: http://www.ixxat.com/profinet_intro_en.htmlt

[26] P. I. Profibus&Profinet, *A PI White Papter - PROFINET and IT*, December 2008.

[27] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "Ttethernet dataflow concept," in *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*, july 2009, pp. 319 –322.

[28] TTTech Computertechnik AG, "Scalable real-time ethernet platform : Ttethernet - a powerful network solution for all purposes," *Ensuring Reliable Networks TRREch*, 2009.

[29] (Acessed:March, 2012). [Online]. Available: http://standards.ieee.org/develop/regauth/ethertype/eth.txt

[30] M. Tim Jones, "Anatomy of the linux networking stack from sockets to device drivers," *developerWorks*, June 2007.

[31] K. Wehrle, F. Pählke, H. Ritter, D. Müller, and M. Bechler, *The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel.* Prentice Hall, August 2004.

[32] P. J. Salzman and O. Pomerantz, *The Linux Kernel Module Programming Guide*, 2001. [Online]. Available: http://tldp.org/LDP/lkmpg/2.4/html/book1.htm

[33] C. Benvenuti, *UnderStanding the Linux Network Internals.* O'Reilly Media, Inc., December 2005.

[34] (Acessed:March, 2012). [Online]. Available: http://packeth.sourceforge.net/

[35] (Acessed:March, 2012). [Online]. Available: http://www.wireshark.org/

# Apêndice A

# Como instalar um *kernel* em Linux - Fedora 14

1. Fazer *download* do código fonte de *kernel* linux.

   Nota1: Abrir um terminal, utilizar o comando "uname -a"de modo a saber qual a versão do *kernel* já implementado para evitar problemas de configurações;

   Nota2: Código fonte da versão semelhante pode ser encontrado em www.kernel.org.

2. Ir para cd /usr/src e copiar o código fonte para esta pasta.

3. Descompactar o código fonte

   tar -xvjf /usr/src/linux-2.6.35.tar.bz2

4. Criar um link simbólico de nome linux.

   ln -sf linux-2.6.45/ linux

5. Ir para cd linux.

6. Copiar o ficheiro de configuração do *kernel* já existente.

   cp /boot/config-2.6.35.21-170.2.56.fc14.i686 .config

7. Executar o comando make oldconfig

8. Executar o comando xconfig ou make menuconfig (nesta situação gravar como ficheiro de configuração o .config copiado no ponto 6).

9. Editar na Makefile o campo "Extraversion"para algo com significado. Exemplo: "Extraversion =.101".

10. Executar o comando make.

11. Executar o comando make modules.

12. Executar o comando make modules_install.

13. Executar o comando make BzImage.

14. Criar uma imagem de arranque inicial dos módulos existentes para a pasta /boot.

    mkinitrd -v /boot/initrd-2.6.35.101.img 2.6.35.101
    ou
    mkinitramfs -v /boot/initrd-2.6.35.101.img 2.6.35.101

15. Copiar o BzImage criado para a pasta /boot e atribuir um nome com significado.

    cp arch/x86/boot/bzImage /boot/bzImage-2.6.25.101

16. Criar uma nova entrada no Grub editando o ficheiro menu.lst com ficheiros criados no ponto 14 e 15.

    vim /boot/grub/menu.lst

17. Reiniciar o computador. Uma nova entrada para o *kernel* criado deverá aparecer.

# Apêndice B

# Código do Matlab utilizado na análise dos pacotes recebidos

```
clear all
close all
clc
format long

%% Ler os ficheiros .txt retirados do Wireshark
source = fopen('source.txt');
destination = fopen('destination.txt');

src =textscan(source, '%d %f %s %s %s %s %s','headerlines',1);
dst =textscan(destination, '%d %f %s %s %s %s %s','headerlines',1);

fclose(source);
fclose(destination);

%%Calcular a duração
duracao = zeros(length(src{2}),1);
for i=1: length(duracao)
   duracao(i)= dst{2}(i)-src{2}(i);
end

%%Tempo Médio,Máximo,Mínimo e Desvio Padrão
media=mean(duracao);
maximo=max(duracao);
minimo=min(duracao);
desviopad=std(duracao);

%%Histograma
hist(duracao,1000);
xlabel('Tempo de "turn around"');
ylabel('Número de pacotes');
```