FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Driver de rede para sistemas distribuídos de tempo-real

Rui Duarte Teixeira Henriques

Mestrado Integrado em Engenharia Electrotécnica e de Computadores - Ramo de Automação

Orientador: Prof. Dr. Luis Miguel Pinho de Almeida

Co-orientador: Dr. Ricardo Roberto Duarte Marau

27 de Junho de 2011

Driver de rede para sistemas distribuídos de tempo-real

Rui Duarte Teixeira Henriques

Mestrado Integrado em Engenharia Electrotécnica e de Computadores - Ramo de Automação

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Dr. Mário Jorge Rodrigues de Sousa

Arguente: Prof. Dr. Paulo Bacelar Reis Pedreiras

13 de Julho de 2011

Resumo

Nos dias de hoje, o recurso a sistemas distribuídos é cada vez maior. E assim sendo, é indispensável que exista uma rede de comunicação robusta que garanta que todos os módulos comunicam entre si. O FTT-SE, assim como outros protocolos, fornece a possibilidade de comunicação tempo-real com recurso à tecnologia Ethernet. Contudo, por vezes é necessária a utilização de aplicações não tempo-real, sob redes tempo-real. Isso implica que os módulos possuam uma interface genérica, para as aplicações poderem comunicar, e que os protocolos ofereçam um meio em que seja possível às tramas comuns circularem na rede, sem serem confundidas com tramas tempo real.

Esta dissertação apresenta uma série de soluções, para o protocolo FTT-SE que possibilitam a criação dessa interface genérica, para aplicações tradicionais. Indica vantagens e desvantagens de cada uma delas, e propõe uma. A solução sugerida recorre à utilização da tecnologia Tun/Tap para a criação de interfaces virtuais, aos quais as aplicações se podem conectar, permitindo assim, que o tráfego não tempo-real circule.

Foram realizados alguns testes, tais como medições de "round-trip delay", ligações de "streaming" e acesso a um servidor web. Estes apresentaram resultados positivos e de acordo com o esperado, tendo sido possível efectuar a circulação de tráfego não tempo-real numa rede FTT-SE.

Abstract

Nowadays, distributed systems are more and more used. Therefore, it is needed a reliable communication network that guarantees that every modules communicate with each other. Like other protocols, FTT-SE enables real-time communication over Ethernet technology. However, sometimes, we have non real-time applications over real-time networks, and they need to exchange data. That means that the modules should have a generic interface for such applications. Besides, protocols must allow in their networks non real-time packets, without being confused with real-time ones.

This thesis shows a series of solutions to FTT-SE protocol, that provide a generic interface for the non-real time applications. Moreover, this thesis explains their advantages and disadvantages, and then it suggests one. The proposed solution use the Tun/Tap Technology, to build a network interface and to enable non real-time applications and, at the same time, allowing theirs communications.

In order to assure the well-functioning of the solution¸ tests were carried out, such as: Round-Trip Time, audio and video streaming, and the deployment of a web server on the FTT-SE network. The test results were very positive and coherent with what we were expecting. The generic interface was successfully created and, at this time, non real-traffic flows over the FTT-SE network are supported.

Agradecimentos

Em primeiro lugar, e acima de tudo, aos meus pais, que sempre me apoiaram e ajudaram em tudo o que puderam, sendo sempre um grande exemplo de trabalho e humildade.

Ao meu orientador Professor Doutor Luís Almeida, e co-orientador Doutor Ricardo Marau, por todo o tempo despendido comigo e pela preciosa ajuda prestada, tanto a nível prático como teórico.

A toda a restante família que sempre me apoiou, tendo sempre uma palavra de amizade e de incentivo. Aos meus amigos CF's por todas as jantaradas e encontros que tivemos e que muito me enriqueceram, com um abraço especial ao Carlos Ferreira. Aos meus amigos da FEUP, que sempre me ajudaram e contribuíram para o bom sucesso do trabalho, com conselhos e ajudas técnicas.

Por último, mas não menos importante, à minha namorada Joana, que sempre me apoiou e ajudou ao longo de todo o projecto.

A todos estes o meu muito obrigado
Rui Henriques

"O meu avô disse-me uma vez que havia dois tipos de pessoas:
as que fazem o trabalho e as que ficam com os louvores.
Ele disse-me para tentar ficar no primeiro grupo; há menos concorrência."

Indira Gandhi

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AMS	Asynchronous Messaging System
EC	Elementary Cycle
EDF	Earliest Deadline First
EPL	Ethernet POWERLINK
FIFO	First In, First Out
FTT-SE	Flexible Time-Triggered Communication over Switched Ethernet
IP	AInternet Protocol
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PPM	Plug'n Play Message
RTP	Real-Time Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round-Trip Time
SMS	Synchronous Messaging System
SRDB	System Requirements DataBase
TCP	Transmission Control Protocol
TM	Trigger Message
UDP	User Datagram Protocol

Capítulo 1

Introdução

Na indústria de hoje, com a crescente complexidade dos sistemas de supervisão e controlo verifica-se cada vez mais a necessidade de introduzir sistemas distribuídos.

Inicialmente, os sistemas de controlo eram bastante centralizados e toda a lógica era concentrada num só ponto. Essa técnica, nos dias de hoje, é cada vez menos adoptada e isto deve-se ao tamanho físico dos sistemas a controlar, bem como à sua complexidade. Verifica-se então, que é vantajoso dividir as aplicações de controlo em pequenas tarefas, que cooperam entre si para atingir o objectivo global do sistema.

Para que tudo seja controlado em conformidade, é forçoso que exista uma comunicação sincronizada e temporalmente correcta.

Uma das redes de comunicações mais usada na indústria é a Ethernet. Contudo, e como será descrito no Capítulo 2, este tipo de comunicação não possui as diferentes propriedades necessárias, por exemplo, assegurar a pontualidade do tráfego. Assim, surge a necessidade da introdução de protocolos adicionais, que permitam o escalonamento de tráfego, neste tipo de redes, de forma a obter comunicações temporalmente correctas.

1.1 Objectivo da Dissertação

Esta Dissertação insere-se no desenvolvimento do protocolo de comunicações Flexible Time-Triggered Communication over Switched Ethernet (FTT-SE). Este protocolo, que será descrito em pormenor no capítulo 3, fornece serviços de escalonamento de tráfego em redes Ethernet, para sistemas de tempo-real, e usa uma arquitectura Mestre/Escravo, cabendo ao Mestre o controlo temporal das comunicações.

A dissertação tem como objectivo o desenvolvimento de uma camada protocolar de um Slave (Escravo) para este protocolo, com execução em ambiente Linux. Cada Escravo tem que realizar duas tarefas principais. Por um lado, terá que responder a pedidos do

Mestre, para efectuar transmissões nos instantes adequados. Por outro, o Escravo terá que notificar o Mestre sempre que possuir informação a transmitir.

A arquitectura a usar para o desenvolvimento do trabalho será estudada perante diferentes soluções.

A dissertação centrará o seu tema no tráfego normal, isto é, não tratará propriamente do tráfego tempo real, para o qual o protocolo foi desenvolvido, mas sim outras aplicações que pretendam comunicar e não possuam informação tempo-real. O objectivo principal é, então, fornecer uma interface genérica a aplicações standard, para que estas possam comunicar sem alterações, estando numa máquina no qual o protocolo FTT-SE está a operar.

1.2 Tarefas da Dissertação

Os trabalhos da Dissertação estão organizados de acordo com o seguinte plano:

- Pesquisa dos diferentes protocolos Tempo-Real sobre Ethernet;
- Estudo do protocolo FTT-SE em pormenor e mais especificamente o Slave;
- Desenvolvimento de uma possível máquina de Estados para o Slave FTT-SE;
- Estudo global dos Device Drivers sobre a plataforma Linux;
- Estudo de possíveis soluções para solucionar o problema em questão;
- Desenvolvimento de um protótipo de Device Driver Ethernet para o Slave FTT-SE baseado na maquina estados acima referida;
- Avaliação do protótipo e efectuar as devidas correcções;
- Validação do protótipo e realização de testes finais;

1.3 Estrutura da dissertação

Neste primeiro capítulo foram traçados os objectivos da dissertação e feito um enquadramento geral da problemática com os sistemas de tempo real.

O restante documento está então, organizado da seguinte forma:

O capítulo 2 introduz o tema dos sistemas distribuídos e das comunicações tempo real. Aborda a sua problemática quando usada a tecnologia Ethernet e a necessidade de introdução de switches. São referidos alguns protocolos que procuram solucionar o problema.

O capítulo 3 aborda o protocolo de comunicação em questão (FTT-SE), onde é explicado com algum detalhe o protocolo e em especial a parte de Slave. Refere as trocas de

mensagens existentes no protocolo, e alguns comportamentos específicos para situações particulares. É apresentado também o diagrama de estados que o Slave FTT-SE executa.

O capítulo 4 expõe o sistema operativo Linux. Explica a sua estrutura e sua arquitectura modular, incide particularmente sobre o Kernel Linux. Aí, aborda levemente cada um dos módulos que o constituem e em particular o módulo de rede, onde é feita uma exposição mais detalhada.

No capítulo 5 são referidas algumas soluções que foram equacionadas para o desenvolvimento da dissertação. É explicado o problema em causa com mais detalhe e os requisitos da solução pretendida. São então expostas as diferentes soluções, referindo as suas vantagens e desvantagens, e principal razão para não ter sido adoptadas.

No capítulo 6 é possível encontrar a solução escolhida. Explica com algum detalhe a tecnologia usada, as suas potencialidades, assim como vantagens e desvantagens. Aborta, também, os problemas da solução usada e como resolve-los para que possa ser utilizada.

O capítulo 7 refere alguns testes executados e os seus resultados. Expõe em que consistiam os testes feitos e protocolos usados e propriedades dos mesmos. Refere os resultados e se estão enquadrados com o esperado ou não.

O capítulo 8 apresenta as conclusões da dissertação assim como possíveis trabalhos futuros.

Capítulo 2

Estado da Arte

Este capítulo pretende contextualizar o protocolo FTT-SE, os protocolos baseados em Ethernet e os sistemas distribuídos no âmbito das comunicações tempo real. Neste contexto, este capítulo refere e descreve brevemente alguns dos protocolos já existentes para comunicações de tempo-real sobre Ethernet, assim como as suas características.

2.1 Sistemas distribuídos de Tempo Real

Os sistemas distribuídos sofreram uma grande evolução devido à crescente complexidade dos sistemas de controlo, desenvolvidos para a indústria.

Um sistema distribuído (Figura 2.1) é constituído por um conjunto de unidades de processamento independentes, que através da troca de informação entre elas, executam uma série de processos cooperantes que fazem parte de uma aplicação global. Do ponto de vista do utilizador existe a ilusão de que toda a aplicação é executada na mesma unidade. Mas para que a aplicação execute convenientemente, é forçoso que as unidades estejam sincronizadas.

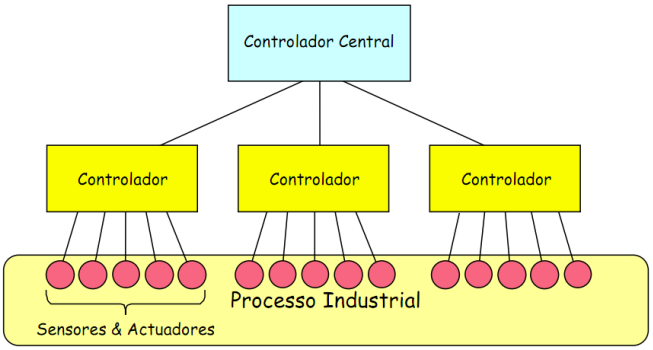

Figura 2.1: Sistema de Controlo distribuído [1]

Contudo, nos sistemas distribuídos a sincronização não é simples. Para que as unidades se sincronizem, precisam de trocar informações entre si.

Um sistema distribuído é então, um sistema no qual o Hardware e Software estão divididos em múltiplas unidades dispersas em locais físicos diferentes, que estão ligados em rede, para que comuniquem e coordenem as suas acções através da troca de mensagens.

Este tipo de sistemas possui algumas limitações à sincronização, tal como o facto de não existir um relógio global. Em vez disso, existem vários relógios que têm que ser sincronizados, usando para isso, um protocolo adequado para a troca de informação de relógio. Os atrasos de comunicação, particularmente quando são variáveis, acarretam uma falta de precisão na sincronização dos relógios de cada uma das unidades.

A coordenação das diferentes unidades para a execução de uma dada aplicação, pode ser conseguida com relógios sincronizados ou com troca de mensagens explícitas de controlo, tal como as mensagens de comando de um mestre.

As vantagens associadas à utilização de um sistema distribuído prendem-se com vários factores.

O factor geográfico é uma das suas principais vantagens uma vez que permite que a unidade de processamento fique fisicamente perto do processo que está a controlar. Por outro lado, existe uma fácil extensibilidade, que possibilita que a qualquer momento, se assim for desejado, o sistema possa crescer ou ser alterado. Existem também, vantagens ao nível da disponibilidade e do desempenho, ou seja, o sistema tipicamente está menos sobrecarregado e é mais rápido. Além de todas estas vantagens existe o factor custo, que dependendo da aplicação em causa, tipicamente é mais barato que um sistema centralizado (Figura 2.2), por exemplo, devido a redução de cablagem.

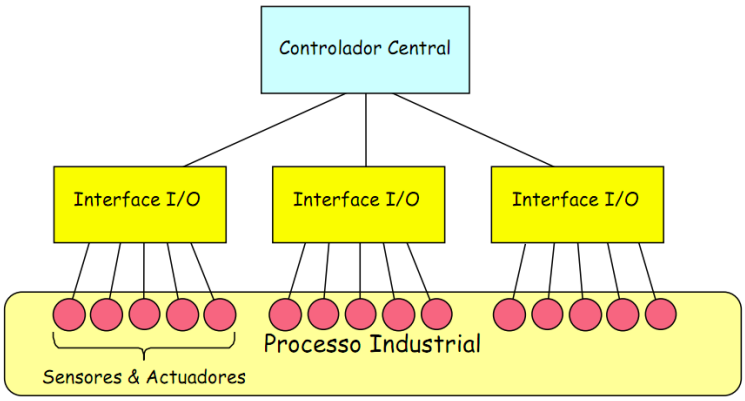

Figura 2.2: Sistema Centralizado [1]

Os sistemas de controlo distribuídos estão frequentemente condicionados a requisitos temporais. Um dos aspectos com mais impacto na pontualidade de um sistema distribuído

é o controlo do acesso à rede de comunicação. Este aspecto é particularmente relevante em sistemas distribuídos baseados em redes partilhadas como se pode ver na Figura 2.3.

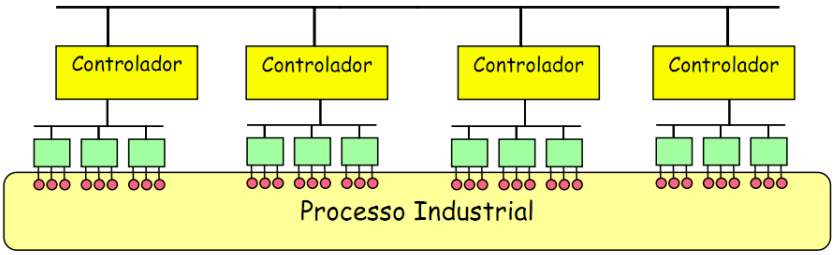

Figura 2.3: Sistema de controlo distribuído, baseado em redes partilhadas [1]

A expressão tempo real está intrinsecamente ligada à natureza dos processos que se quer controlar. A noção de tempo real ao nível dos sistemas de controlo indica que uma determinada tarefa desse mesmo sistema possua um tempo de execução definido. Esse tempo pode ser curto ou longo, mas do ponto de vista do sistema, o que interessa é que a tarefa seja executada dentro da janela temporal previamente definida. Um sistema de tempo real tem que produzir as saídas necessárias, não só logicamente mas também temporalmente correctas.

O tempo real é então, neste contexto, a capacidade do sistema produzir dados correctos quando estes são necessários. Certos processos impõem restrições mais apertadas, exigindo tempos de resposta do sistema bastante reduzidos, por exemplo, 1ms. Estas situações mais exigentes obrigam à utilização de tecnologias adequadas.

2.2 A problemática

Em sistemas distribuídos a comunicação é feita através de módulos, que estão ligados a uma rede de comunicações. A comunicação é usada para partilhar informação entre os dispositivos que dela necessitam. Essa informação circula pela rede e vai perdendo actualidade à medida que o tempo passa, razão pela qual é necessário usar técnicas e terminologias adequadas.

Nesta dissertação vamos apenas considerar redes de comunicações baseadas em tecnologia Ethernet com o recurso a switches. Portanto, o problema que se coloca é se os switches são capazes de fazer circular a informação com a velocidade exigida (Figura 2.4), para que cada uma das unidades (módulos), possua a informação certa no momento certo.

O *switch* é então, responsável por fazer a comunicação entre dois ou mais nós.

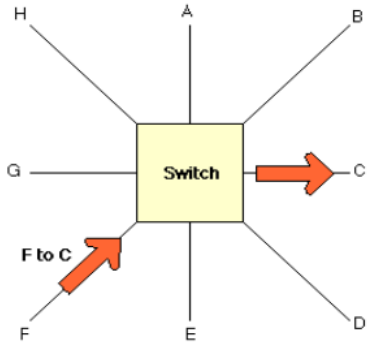

Figura 2.4: Comunicação via Switch [2]

2.3 A Ethernet em sistemas de Tempo-Real

Em aplicações distribuídas de tempo-real é necessária uma interligação entre os vários módulos, que garanta que estes consigam executar atempadamente as diferentes tarefas de uma mesma aplicação.

Hoje em dia, Ethernet é a tecnologia de interconexão mais comum e por isso, é normal que se pense nela como uma possível solução. Com a introdução de switches nas comunicações Ethernet, esta tornou-se bastante interessante devido à sua capacidade de isolamento de tráfego e à eliminação de colisões no acesso a rede.

Os switches não possuem ligação física entre os seus portos e as mensagens são reencaminhadas de cada porto de entrada para um ou mais portos de saída (Figura 2.4).

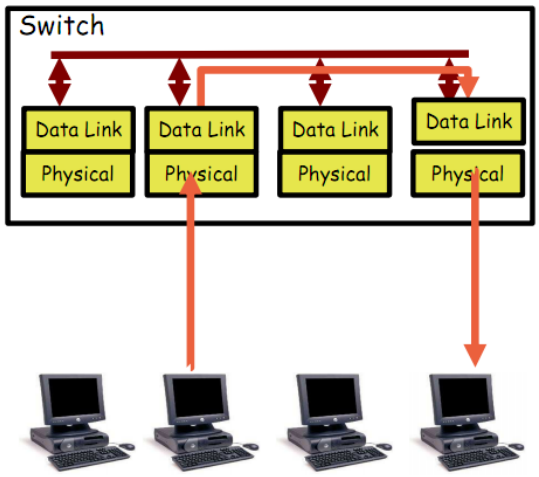

Figura 2.5: Comunicação através de switch [2]

Quando uma mensagem chega a um dado porto do switch, é analisada e colocada no buffer do porto de destino. Contudo, se o porto em questão estiver ocupado essa

mensagem será colocada em fila de espera, para posteriormente ser transmitida (Figura 2.5).

Actualmente, os switches não têm dificuldades em receber todas as mensagens que chegam, o que permite que não se construam filas nos buffers de chegada. Porém, no envio, a formação de filas é quase sempre verificada, devido por exemplo, a um grande conjunto de mensagens recebidas num curto espaço de tempo, para o mesmo porto. Esta situação, se não for limitada, pode levar a um excesso de mensagens no switch, fazendo ultrapassar a memória existente e levando, assim, a que algumas mensagens sejam descartadas. Este tipo de situações afecta gravemente o comportamento temporal de um sistema distribuído com rede Ethernet, e faz com que o Switch, por si só, não seja capaz de garantir tempo real. Um outro problema é a existência de latência de comunicação, que dificulta a previsão consistente do tempo que leva uma mensagem a viajar na rede [3].

Apesar disso, os switches trouxeram um meio bastante viável para resolver o não determinismo inerente ao protocolo original de controlo de acesso ao meio em Ethernet, facilitando assim, a sua aplicação em sistemas de Tempo Real.

Para solucionar os problemas introduzidos pelos switchs, foram desenvolvidas uma série de técnicas que permitem melhorar o respectivo comportamento temporal. Essas técnicas foram desenvolvidas segundo diferentes abordagens, algumas das quais são brevemente descritas nas secções seguintes.

2.3.1 Switch com escalonamento EDF

Este protocolo propõe um escalonamento de tráfego do tipo EDF (Earliest Deadline First) e assim, garante tempo-real, sendo apenas necessária a introdução de um camada na pilha protocolar dos módulos, entre o device driver Ethernet e a pilha TCP/IP. Os switches são modificados, uma vez que tanto estes dispositivos como os módulos executam o escalonamento do tipo EDF (que não é comum em switches convencionais) [4].

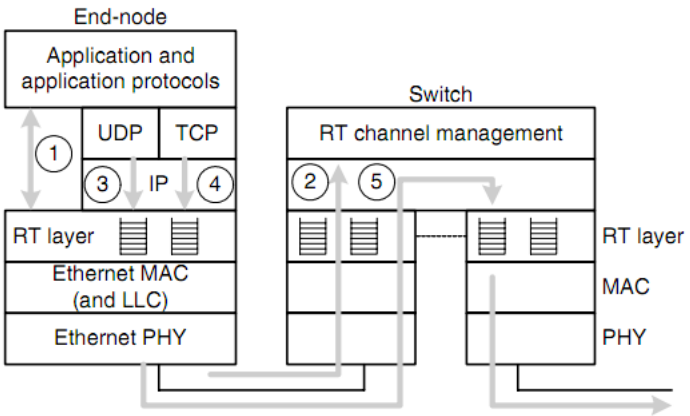

Figura 2.6: Ethernet com Switchs Tempo-Real com recurso a EDF [4]

A grande vantagem da utilização deste protocolo é que, apesar de este ser aplicado em processos de tempo real, é possível a ligação a Internet dos diferentes módulos, sem que isto afecte a funcionalidade das comunicações de tempo-real [5].

2.3.2 PROFINET IRT

O PROFINET IRT (Isochronous Real Time), que integra a família de protocolos PROFINET, foi desenvolvido especificamente para aplicações de tempo real, mas necessita também de hardware adaptado quer no lado do switch quer no dos módulos.

Este protocolo baseia-se então, no estabelecimento de ciclos de duração pré-determinada que são divididos, numa fase assíncrona e outra síncrona. As durações do ciclo e das referidas fase são calculadas por um mecanismo de escalonamento offline, que avalia as necessidades de comunicação, assim como outros factores relacionados com a topologia aplicada. Vários tipos de topologias são suportados, por exemplo, em linha, em estrela e em anel [6].

O protocolo PROFINET IRT permite obter variações no atraso de rede (Jitter) inferiores a 1μs. O tempo de ciclo tem que ser superior a 250μs. Esta limitação está relacionada com a disponibilização de mais de metade do tempo para a transmissão de mensagens assíncronas e também com o tamanho dos pacotes Ethernet na rede (Figura 2.7) [5].

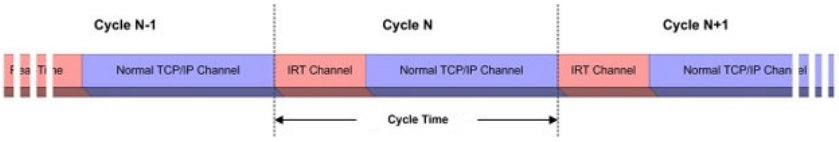

Figura 2.7: Tempo de ciclo segundo PROFINET IRT [7]

2.3.3 EtherCat

Uma outra topologia usada é a Master/Slave. Neste tipo de abordagem o mestre, controla e inicia todas as transacções, enquanto que os restantes módulos, os escravos, recebem ordens e executam-nas. Um protocolo que implementa esta técnica é o protocolo EtherCat.

Em teoria, este protocolo consegue trabalhar usando hardware Ethernet standard. Contudo, na prática os escravos necessitam de hardware especial, para facilitar o reencaminhamento de pacotes com alteração do respectivo conteúdo "on-the-fly". No que diz respeito ao mestre, este pode ser implementado com componentes standard [5].

Todas as mensagens do protocolo (Figura 2.8) são enviadas pelo mestre, que constrói uma trama e envia-a para a rede. Cada escravo intercepta-a, lê-a e escreve a informação que quer partilhar (Figura 2.9). Quando a mensagem passar por todos os escravos, volta ao mestre e termina o ciclo. Cada escravo possui um tempo limite para ler e escrever na

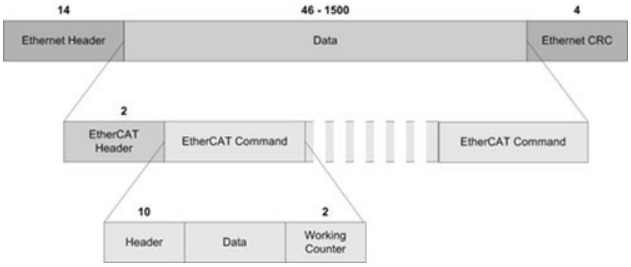

Figura 2.8: Trama EtherCAT [7]

mensagem, normalmente abaixo dos 500ns, dependendo da topologia usada, sendo estas operações efectuadas enquanto a mensagem é reencaminhada do módulo anterior para o seguinte.

O protocolo EtherCAT possui ainda um processo de sincronização dos escravos, que funciona correctamente para sincronizações abaixo da centena de nanossegundos. Em princípio este método é aplicável em qualquer topologia, contudo a topologia em linha é a mais comum [7].

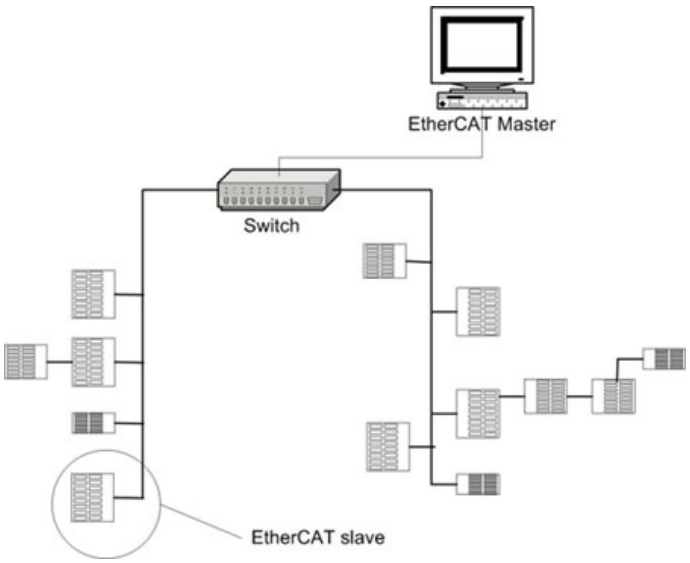

Figura 2.9: Exemplo de aplicação do protocolo EtherCAT [7]

2.3.4 Ethernet POWERLINK

O protocolo Ethernet POWERLINK (EPL), também sugere uma topologia do tipo Mestre/Escravo. Em cada rede existe apenas um mestre e este é o único membro activo da rede, isto é, os escravos só comunicam quando forem solicitados.

Figura 2.10: Período do protocolo EPL

As comunicações são organizadas em ciclos de duração finita e fixa, e cada ciclo é constituído por quatro fases distintas. Numa primeira fase o mestre envia uma mensagem ("Start-of-Cyclic", SoC), esta tem como objectivo a sincronização dos diferentes escravos. Na segunda fase do período, Cyclic Period, o mestre inquire cada escravo para transmitir informação periódica, para isso envia uma mensagem de "Poll Request", a qual cada escravo deve responder com uma "Poll Response". No final desta fase o mestre envia uma trama "End-of-Cyclic Period"(EoC). Na fase de Asynchronous Period a transferência de dados também é feita sobre o controlo do mestre, este vai inquirindo os escravos que previamente tenham informado o mestre através de um "Poll Response", que têm informação para transmitir. Nesta fase o mestre inquire um módulo em cada ciclo.

O protocolo EPL não necessita do recurso a switches para evitar colisões uma vez que, o mestre visualiza as comunicações. Assim, a arquitectura da rede pode ser construída com hubs repetidores. Neste protocolo é mesmo recomendado o uso de hubs repetidores ao invés de switches, uma vez que estes introduzem mais jitter e latência reduzindo o determinismo das comunicações [7].

2.3.5 FTT-SE

O protocolo Flexible Time Triggered comunication over Switched Ethernet (FTT-SE) segue também a topologia de mestre escravo e foi pensado para suportar aplicações de tempo-real adaptáveis e reconfiguráveis.

Neste tipo de aplicações existem várias propriedades entre as quais previsibilidade, tratamento de tráfego periódico com diferentes períodos, tratamento de tráfego esporádico, latência limitada, reconfiguração dinâmica e gestão dinâmica de QoS (Qualidade de Serviço).

Para alcançar todas estas propriedades, o protocolo FTT-SE utiliza uma arquitectura centralizada de agendamento e controlo de transmissão Master/Multislave. Este sistema permite controlar o tráfego que está na rede a cada instante evitando assim, que o switch fique sobrecarregado, mantendo a previsibilidade do sistema, e em particular os tempos de propagação na rede. O Master é então responsável por informar os diferentes escravos, de quando e que informação devem transmitir(Figura 2.11).

O protocolo funciona ciclicamente, e em cada ciclo o Master envia uma mensagem de controlo para a rede, indicando que mensagens devem ser transmitidas naquele período

de tempo. Cada ciclo tem o nome de Elementary Cycle (EC) e a mensagem enviada pelo Master, Trigger Message (TM) [8].

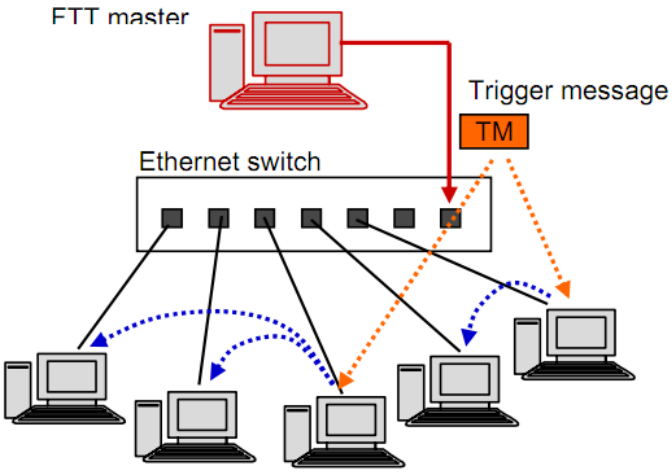

Figura 2.11: Arquitectura do protocolo FTT-SE [9]

Cada EC está dividido em duas partes, uma dedicada à transmissão de mensagens síncronas, SMS (Synchronous Messaging System) e uma outra a mensagens assíncronas, AMS (Asynchronous Messaging System).

A estrutura do protocolo que está apresentada na figura 2.12, é então constituído por um Master que organiza a transferência de dados, baseado na tabela SRDB, e envia a cada EC a TM. Por outro lado existe o Slave, que possui a parte da tabela SRDB que lhe corresponde, a NRDB, e envia as variáveis quando são solicitadas.

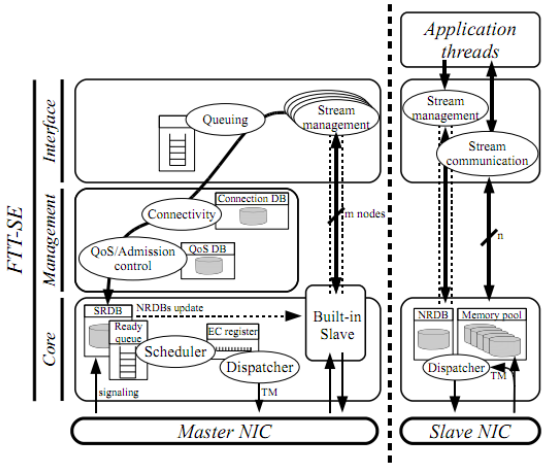

Figura 2.12: Detalhes internos do FTT-SE [10]

Capítulo 3

O protocolo FTT-SE

O presente capítulo pretende descrever o protocolo com um pouco mais de detalhe do que o apresentado no capítulo anterior. Serão explicadas as mensagens trocadas no sistema e a sua estrutura. E abordadas também, as principais situações possíveis durante a execução. Será, ainda, brevemente referido, o uso de switches full-duplex.

3.1 Apresentação

O protocolo FTT-SE permite um controlo da transmissão de dados para o meio, evitando assim que cada máquina envie dados quando bem entende, o que poderia conduzir a uma sobrelotação dos buffers do switch, levando a atrasos temporais, ou até a perdas de mensagens. Ao invés, o protocolo estabelece ciclos e em cada um deles, são seleccionadas as mensagens que devem ser transmitidas. Assim sendo, controla-se o número de mensagens em circulação, possíveis sobrecargas no switch e ainda a temporização, ou seja, o momento em que cada máquina deve transmitir e receber mensagens.

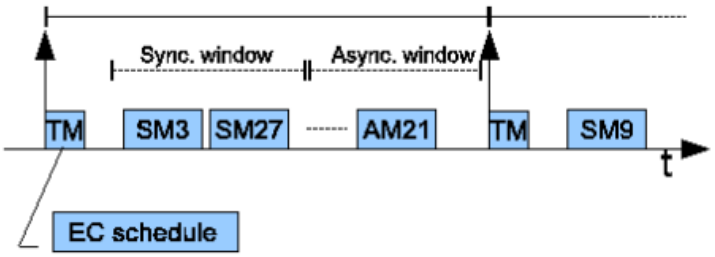

Figura 3.1: Elementary Cycle [8]

Cada ciclo do FTT-SE tem o nome de Elementary Cycle, como já foi referido no capítulo anterior, e está representado na figura 3.1. Este possui um tempo definido para cada sistema, ou seja, a periodicidade do EC é dinâmica contudo, apenas em termos de projecto, o que quer dizer que uma vez dimensionada para um dado sistema, esse valor é fixo e igual ao valor de menor periodicidade necessária para este sistema.

A TM é enviada pelo Master no início de cada EC em broadcast, que sincroniza a rede e informa todos os módulos das mensagens, síncronas e assíncronas, a serem transmitidas nesse ciclo. A Figura 3.2 mostra a organização lógica das TM enquanto o significado dos respectivos campos é descrito na Figura 3.3 [8].

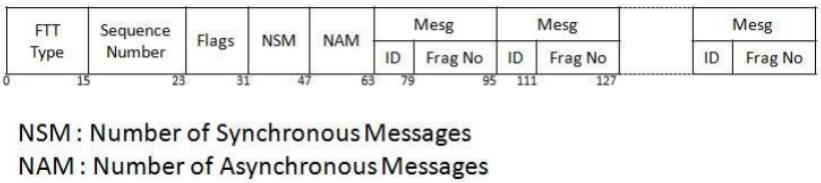

Figura 3.2: Trigger Message [8]

Campo	Tamanho(*bytes*)	Descrição
FTT Type	2	Tipo de mensagem
Seq No	1	Numero de sequência
Flags	1	conjunto de *flags*
NSM	2	Número de mensagens Síncronas
NAM	2	Número de mensagens Assíncronas
MSG Index	4	Informação sobre mensagem
ID	2	Identificador da mensagem
Frag No	2	Número de Fragmentação

Figura 3.3: Descrição dos campos da Trigger Message [8]

Após a recepção da TM cada um dos escravos irá enviar para a rede a informação solicitada. As mensagens enviadas pelos escravos possuem a estrutura descrita na figura 3.4 e os respectivos campos são descritos na figura 3.5 [8].

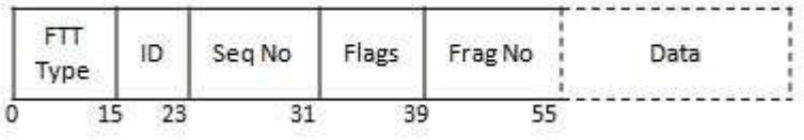

Figura 3.4: Syncronous/Asyncronous FTT Message [8]

Campo	Tamanho(*bytes*)	Descrição
FTT Type	2	Tipo de mensagem
ID	2	Identificador da mensagem
Seq No	1	Numero de sequência
Flags	1	conjunto de *flags*
Frag Seq No	2	Número de Fragmentação
Data	Variável	Dados a serem transmitidos

Figura 3.5: Descrição das mensagens dos Escravos [8]

Para permitir ao Master comandar a transmissão de mensagens assíncronas, o protocolo usa um mecanismo específico que permite aos módulos informar o Master que tem informação para transmitir nesse período.

Para que o protocolo execute correctamente o controlo das mensagens síncronas (periódicas), é necessário ainda que o Master possua a informação das suas respectivas propriedades, tais como, tamanho, período e fase, para que possa decidir quem vai transmitir em cada ciclo. Essa informação está armazenada na SRDB (System Requirements DataBase), que reside no Master e contém ainda outros dados tais como, a capacidade máxima e a duração de cada EC. Esta informação está contida numa tabela que é partilhada pelo Master e pelo Slave. Contudo, no Slave tem o nome de NRDB e não possui toda a informação que o Master possui, uma vez que, do ponto de vista do Slave o importante é saber que informação tem que enviar, e qual a que vai, possivelmente receber.

3.2 Master FTT-SE

O Master FTT-SE é o principal elemento da rede. Ele coordena o tráfego, é responsável por manter a SRDB, efectuar o escalonamento das mensagens e enviar as Trigger Message, em broadcast, para todos os nós da rede.

A figura 3.6 apresenta a estrutura interna do Master FTT-SE.

A imagem apresenta a estrutura típica do módulo de rede, de um Master numa rede FTT-SE, e esta possui uma estrutura com três camadas distintas. A camada “Core” trata das tarefas básicas de comunicação, nomeadamente controlo de transmissões e escalonamento do tráfego.

Logo de seguida existe a camada “Management”, que realiza o controlo de sessões e estabelece as conexões aos diferentes Slaves, no que diz respeito aos canais nos quais se pretendem associar como produtores ou consumidores. Esta camada possui ainda funções de controlo de qualidade, isto é, verifica se é possível o escalonamento das mensagens perante as condições impostas pelas aplicações.

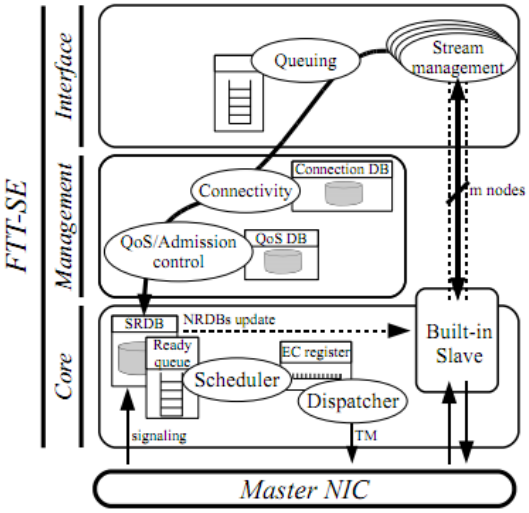

Figura 3.6: Estrutura interna do Master FTT-SE [10]

Por fim, a "Interface" fornece um conjunto de funções que permitem às aplicações comunicar com o protocolo, por exemplo, escrever, ler, registo de canais, etc.

Além disso, existe ainda um módulo relevante, o "Built-in Slave". O protocolo foi desenvolvido para apenas serem estabelecidas comunicações entre Slaves, o que significa que estes não podem comunicar com o Master, caso necessitem. Este módulo permite que essa comunicação se realize, assim sendo, é possível aos Slaves, no intervalo temporal correcto, comunicarem com o Master.

3.3 Slave FTT-SE

A estrutura interna do Slave, como se pode ver na figura 3.7, assemelha-se à estrutura do Master. Contudo, não possui a camada de "Management", uma vez que não cabe ao Slave efectuar escalonamentos nem controlo de sessões e conexões.

O Slave do protocolo tem que responder correctamente e no tempo certo ao Master. Sendo assim, encontram-se três situações importantes de troca de mensagens no sistema.

A primeira situação que importa referir, é o funcionamento normal do sistema, em que existe um Master que coordena todas as comunicações. Um exemplo desse modo de funcionamento pode ser observado na figura 3.8.

Na presente figura pode observar-se o funcionamento normal de um EC do protocolo. Na primeira parte, o Master envia a todos os Slaves a Trigger Message, que tem por um lado, o objectivo de informar que mensagens vão circular na rede nesse ciclo, e por outro, sincronizar todos os Slaves. Esta mensagem chega em simultâneo a todos os dispositivos,

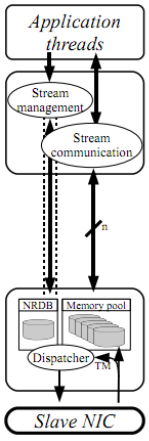

Figura 3.7: Estrutura interna do Slave FTT-SE [10]

uma vez que as filas dos portos de saída no switch estão vazias, garante-se assim, a sincronização dos módulos. Ainda nesta primeira fase, todos os Slaves devem enviar uma mensagem de sinalização (SIG), que tem como objectivo informar o Master se o Slave tem ou não informação assíncrona a transmitir. Mesmo que não possua informação para transmitir, estes devem enviar a mensagem de sinalização e garante-se assim, que o Master sabe que Slaves estão activos. A esta parte do Elementary Cycle, deu-se o nome de Gard Window.

É de salientar também, a fase que na imagem tem o nome de Turn Arround. Este tempo, 200 μs, destina-se ao processamento da TM por todos os Slaves. Era possível reduzir este tempo, mas seria necessário recorrer a hardware especial, o que pode não ser vantajoso.

A fase que se segue é a transmissão de dados (Transmition Time), quer assíncrona quer síncrona. No exemplo da figura o Slave1 envia uma mensagem, Var1, para o Slave2.

Uma particularidade interessante é que não é necessário separar o período síncrono do assíncrono, isto porque o escalonamento feito pelo Master antecipa essas situações. Assim, os Slaves têm que ler a TM e enviar todas as mensagens pedidas. Deste modo, o sistema funcionará correctamente, uma vez que no fim de cada ciclo as filas de saída do switch estão todas vazias.

Uma outra situação possível é quando um Slave se conecta ao sistema quando este já está em funcionamento(figura 3.9). Nesta situação, o Slave deve esperar pela TM para se sincronizar, e no momento destinado ao envio da mensagem de sinalização, deve avisar o Master que está activo. Esta mensagem deve conter apenas o seu endereço Ethernet (MAC), e deve repetir este procedimento até que o Master informe o Slave, que recebeu o seu aviso, e que pode começar a trabalhar.

A mensagem de sinalização nesta fase tem o nome de Plug'n Play Message (PPM), e a mensagem de resposta chama-se Plug'n Play Message Response.

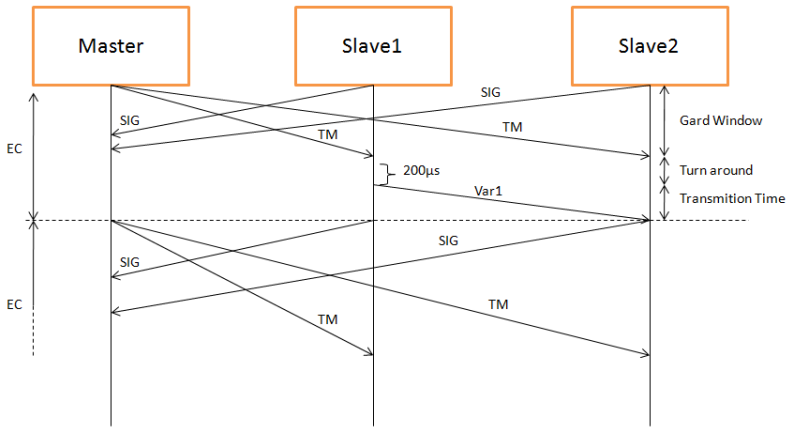

Figura 3.8: Funcionamento normal do FTT-SE

Ainda uma outra situação é quando o Master quer actualizar as tabelas de escalonamento dos Slaves (NRBD). O Master possui a tabela com as necessidades do sistema (SRDB), e todas as alterações a essa tabela são feitas no Master, e só depois propagadas aos Slaves. Para isso, o Master envia uma mensagem assíncrona broadcast, no período de transmissão, a notificar todos os Slaves que ocorreu uma alteração na tabela e cedendo a nova tabela. Essa mensagem tem o nome de NRBD update. Os Slaves afectados por essas alterações devem substituir as suas tabelas. Esse processo pode ser visto na figura 3.10.

O processo de transmissão de mensagens decorre sem colisões. Além disso, o escalonamento prevê quais as comunicações que poderiam provocar filas nos portos de saída, e evita-as no mesmo ciclo.

O uso de switches full-duplex possibilita a troca de dados em sentidos contrários e em simultâneo. De outro modo, não seria possível a primeira fase do EC, em que o Master envia a TM a todos os módulos, pela ligação de uplink, ou seja, a ligação envia dados do Master para o switch e ao mesmo tempo recebe todas as mensagens de sinalização, pela ligação downlink, vindas dos Slaves.

Na figura 3.11 é possível verificar como as mensagens são trocadas entre os Slaves e o Master. O Token na imagem representa a TM que o Master envia para os Slaves, e A e B são as mensagens de sinalização dos Slaves para o Master.

O algoritmo do Slave FTT-SE tem que ser capaz de responder a todas estas situações específicas que foram referidas anteriormente. Assim sendo, desenvolveu-se o seguinte diagrama de estados, que pretende ilustrar o seu funcionamento.

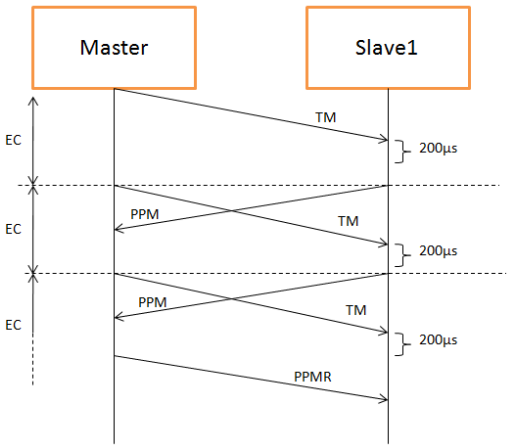

Figura 3.9: Slave FTT-SE regista-se no servidor

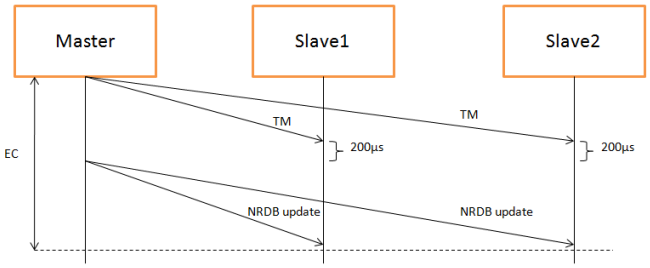

Figura 3.10: Actualização da NRDB

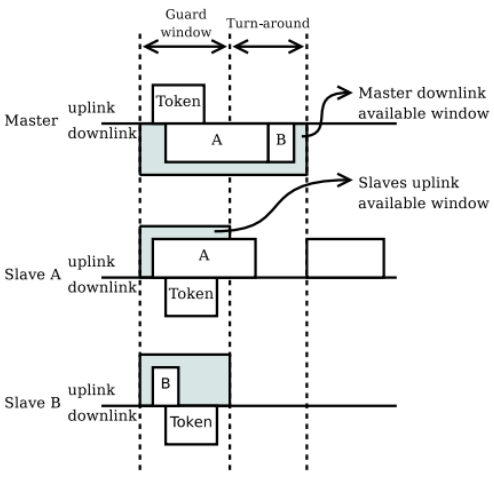

Figura 3.11: Comunicação Full-Duplex do FTT-SE [10]

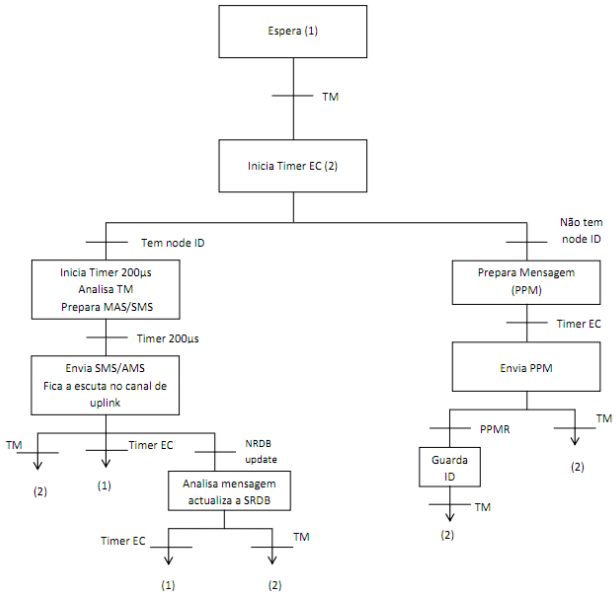

Figura 3.12: Diagrama de Estado do Slave FTT-SE

Capítulo 4

Plataforma Linux

O protocolo FTT-SE foi desenvolvido sobre o sistema operativo Linux, o que significa, que a interface a desenvolver terá que ser, também desenvolvida em Linux. Assim sendo, é necessário um estudo mínimo do funcionamento do sistema operativo, para perceber o que é necessário fazer para criar essa mesma interface.

Este capítulo descreve a arquitectura Linux e explica sucintamente os subsistemas do sistema operativo. Trata mais especificamente do subsistema Kernel do Linux e foca-se, acima de tudo, nas estrutura de rede do Linux e na pilha protocolar de rede que é usada por este sistema.

4.1 Arquitectura Linux

O Linux é um sistema operativo baseado no sistema Unix, e nos dias de hoje tem-se tornado cada vez mais utilizado, principalmente por programadores, devido à sua arquitectura modular. Isto é, está dividido em camadas que executam funções diferentes, e que podem ser removidas e substituídas por outras, facilmente. O Linux possui 4 camadas principais, como é possível ver na figura seguinte.

Estas camadas possuem tarefas e objectivos distintos, e serão brevemente abordadas de seguida.

- User Applications - Este subsistema é o conjunto de aplicações que são executadas nos sistemas Linux, estas aplicações dependem do tipo de uso que é dado ao computador. Contudo, tipicamente, baseiam-se em processadores de texto, navegadores Web, etc.
- O/S Services – Este tipo de serviços são considerados tipicamente, parte do sistema operativo. Por exemplo, a linha de comandos ou aplicações como compiladores estão englobados neste subsistema.

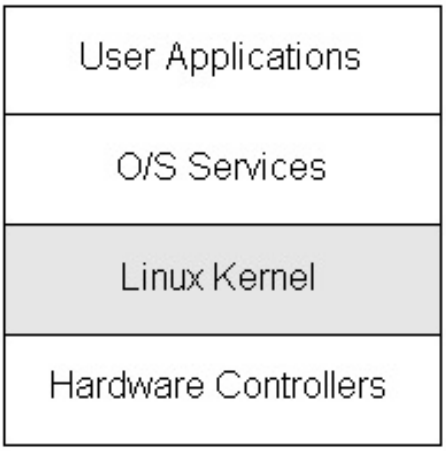

Figura 4.1: Principais Subsistemas do Linux [11]

- Linux Kernel – O Kernel do Linux é o subsistema que permite as camadas superiores abstrairem-se por completo do hardware que existe na máquina. Além de fazer a comunicação entre as camadas superiores e o hardware, este subsistema faz também a mediação do acesso a cada hardware em especifico, ou seja, gere a disponibilidade de cada dispositivo.

- Hardware Controllers – Este subsistema possui todas as informações necessárias para comunicar com qualquer dispositivo que se possa ligar a um sistema Linux. Ou seja, este subsistema faz a ponte entre o Kernel e o hardware. [11]

A arquitectura do kernel é uma das principais razões para o Linux ter sido adoptado por muitos programadores, isto porque, foi pensado para que seja modular. O que significa que todas as suas funcionalidades são módulos, que podem ser adicionados, removidos ou actualizados a qualquer altura. Assim, a qualquer momento podem ser introduzidas funcionalidades adicionais ao Kernel.

Observando então o Linux de um ponto de vista superior, existem duas camadas principais (User-Space, Kernel-Space).

Na topologia apresentada na figura 4.2 a camada User Applications e O/S Services são englobadas numa só, formando uma só camada chamada User-Space.

Uma parte crucial do sistema operativo Linux é o GNU C Library (glibbc). Estas livrarias permitem a realização de funções básicas como printf (imprimir mensagens no ecrã), mallocs (alocar um espaço de memória), etc. Definem também o que normalmente se chama de interface de chamadas ao sistema, isto possibilita o fluxo de informação entre aplicações User-Space e o Kernel. Este interface é bastante importante no sistema operativo uma vez que o Kernel e as aplicações de User-Space ocupam endereços de memória diferentes.

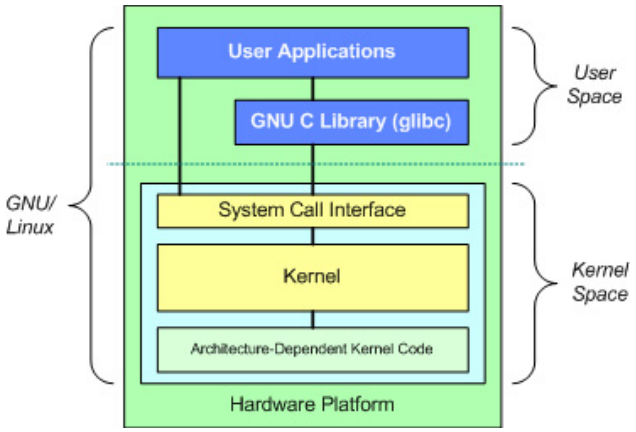

Figura 4.2: Arquitectura fundamental do Sistema Operativo Linux [12]

Do ponto de vista do Kernel este pode ser dividido em três grandes níveis System Call Interface, Kernel e Arquitecture-Dependent Kernel Code.

A um nível superior encontra-se o System Call Interface (SCI), este realiza funções básicas, à semelhança do GNU C Library (glibbc) em User-Space. O SCI implementa então funções do tipo read e write, sendo uma porta de comunicação entre o Kernel-Space e o User-Space. No nível seguinte encontramos o núcleo do Linux, o Kernel code, este código é comum a todas as arquitecturas suportadas pelo Linux, e baseia-se nas principais actividades que o sistema operativos executa, por exemplo, gestão de memória, escalonamento da utilização do CPU, etc. Por fim, encontra-se a Achitecture-Dependent Code este código é específico para cada máquina, ou seja, este código ajuda o sistema operativo a comunicar com o hardware da máquina onde se encontra instalado, daí a necessidade de adaptar o código existente ao hardware existente. [12]

4.2 Kernel Linux

O Kernel do Linux realiza um grande número de importantes tarefas. Para facilitar a sua concepção e alteração, o Kernel está completamente dividido em camadas e em subsistemas distintos, tal como todo o sistema operativo Linux, como é possível ver na figura seguinte.

Ao longo do tempo, com o contributo de muitos programadores, o Kernel foi-se tornando cada vez mais estável. Isto deve-se a uma melhoria significativa da gestão, que este faz dos recursos disponíveis em cada máquina, quer estes sejam memórias, acessos às CPUs, cartas de rede, etc. Contudo, mais importante que isso, a utilização de sistemas operativos com Kernel oferece uma vantagem muito importante, a portabilidade. Isto permite configurar estes sistemas em qualquer máquina, seja ela um computador de mesa,

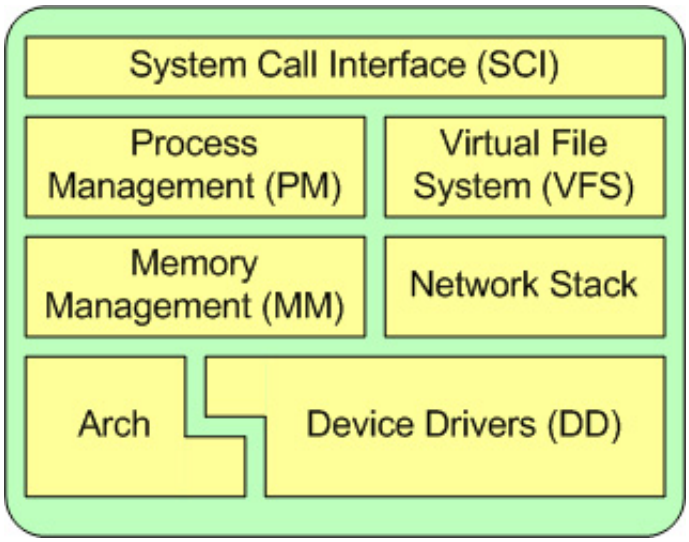

Figura 4.3: Subsistemas do Kernel [12]

perfeitamente normal, ou um computador embebido que controla um processo industrial com requisitos muito apertados, como seja um processo tempo real.

Uma outra propriedade do Kernel Linux é ser monolítico. Isto significa que todas as tarefas do sistema operativo são realizadas pelo Kernel. Todos os serviços são realizados pelo Kernel, o que oferece uma estabilidade maior, não permitindo que o utilizador possa interromper inadvertidamente alguma tarefa importante. [13]

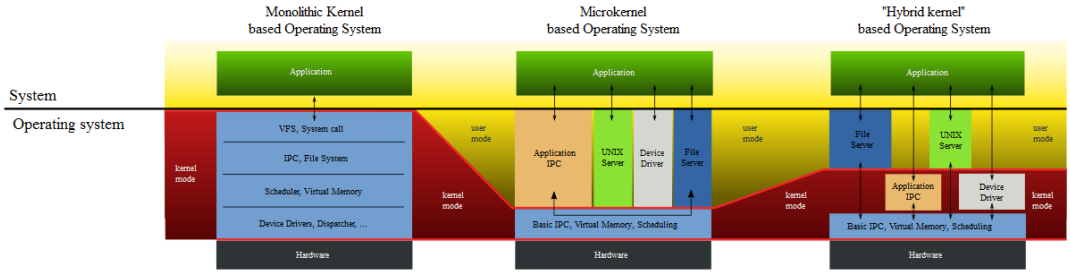

Figura 4.4: Diferentes tipos de Kernel [13]

A existência de um Kernel permite então, aos programadores, desenvolver aplicações User-Space abstraindo-se por completo do hardware que cada máquina contém. Além disso, possibilita que cada processo aja como se fosse o único processo no computador, isto é, possui total acesso a todos os recursos (CPU, memória, etc). Isto é exequível, porque o Kernel gere o acesso dos diferentes processos ao diferente hardware existente.

O Kernel cria uma interface virtual entre os processos User-Space e o hardware da máquina em questão, além disso o Linux é um sistema operativo multitarefa, ou seja,

permite que o processador seja partilhado por várias tarefas aparentemente em simultâneo, permitindo assim que os processos User-Space usufruam dos recursos quase quando solicitam a sua utilização.

O Kernel, como foi visto anteriormente, está dividido em módulos. Cada um com uma função diferente e todos eles, em conjunto com um objectivo comum, o funcionamento correcto do sistema operativo na máquina.

Em primeiro lugar, existe o módulo System Call Interface ou SCI. Este módulo tem como objectivo estabelecer a comunicação User-Space Kernel-Space. Sempre que um processo ou aplicação User-Space necessita de recorrer, por exemplo, à memória, ao CPU ou até a uma carta de rede, faz um pedido a este módulo através da GNU C library, para que se estabeleça a comunicação necessária. [14]

Um outro módulo existente no Kernel é o Process Management (PM). Este é responsável pela execução dos diferentes processos. Em Linux, do ponto de vista do Kernel, não existe diferença entre processo e thread, portanto o Kernel trata todas essas execuções de código da mesma forma. O Kernel cria uma cópia desse processo User-Space em Kernel-Space, e através do SCI inicia, termina e sincroniza as acções com o processo User-Space, para que na realidade o processo executado seja a cópia em Kernel-Space. O módulo é também responsável por garantir que duas aplicações não interajam entre si, quando não o é permitido, ou seja, tem que garantir que cada qual usa a sua própria zona de memória, não sendo possível a uma dada aplicação A aceder a informação gerada por uma aplicação B, pois isso colocaria em perigo os dados privados de uma outra aplicação. Uma outra tarefa é controlar a partilha do CPU pelos diferentes processos em curso no momento. [14] Para isso, possui um algoritmo de escalonamento (Scheduler). Este algoritmo está constantemente a escalonar os processos activos dentro de um dado período de tempo. Esse período é o mesmo quer existam um ou vários processos em curso. Uma outra vantagem é a possibilidade de implementação desse mesmo algoritmo numa máquina multi-processador, por exemplo, a máquina dispõe de dois CPUs, pelos quais o escalonamento deve distribuir os diferentes processos. [12]

A gestão de memória do Kernel é feita pelo subsistema Memory Management (MM), e esta é uma das mais complexas e mais importantes tarefas do Kernel. É feita com o recurso a pages, conjuntos de 4Kb nas arquitecturas mais comuns. Mas, gestão de memória não se baseia em controlar buffers de 4kb, é necessário também, verificar quais as pages que estão cheias, parcialmente usadas ou vazias. Isto para que seja feita uma gestão de alto nível tendo em vista o máximo rendimento do computador. Além disso, e principalmente no caso de a máquina ser controlada por múltiplos utilizadores, deve ser capaz de ir transferindo dados para o disco de modo a que a memória não fique lotada.

Um outro módulo extremamente importante é o que diz respeito aos Device Drivers (DD). Este representa a maior parte do código existente no Kernel, isto justifica-se pelo facto do sistema operativo ser capaz de comunicar com quase todo o hardware existente.

Quase todo o código existente é independente da arquitectura na qual o sistema operativo esta englobado. Contudo, o Arch (Architecture-dependent code) é um código que diz respeito a cada arquitectura em particular. Este código tem como objectivo tirar o maior rendimento e eficiência de cada arquitectura, em particular, adaptando-se ao hardware existente. [12]

O módulo Virtual File System (VFS) é um subsistema que cria uma interface comum para todos os sistemas de ficheiros. Fornece uma ligação simples entre o SCI e o sistema de ficheiros em uso nessa arquitectura como se pode ver na figura abaixo.

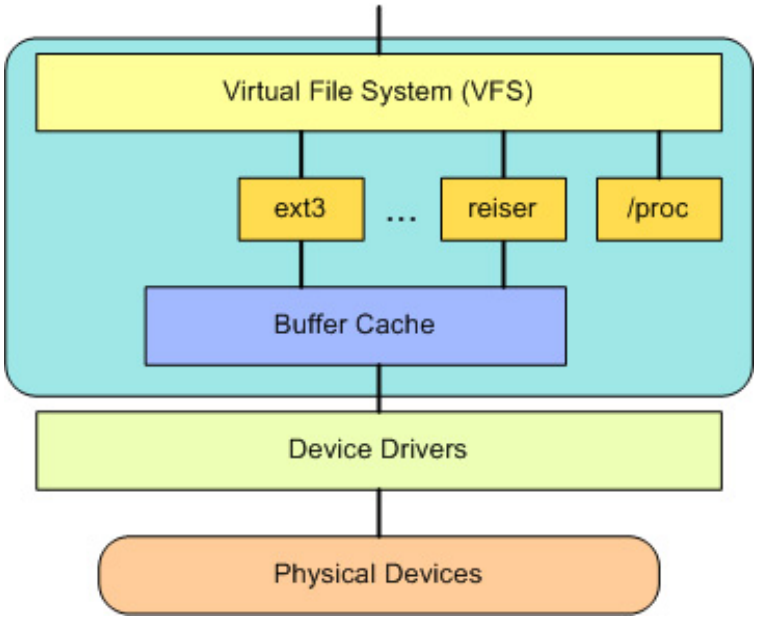

Figura 4.5: Abstracção do Kernel ao sistema de ficheiros

O VFS é um sistema que executa funções simples tais como open, close, read e write. Por baixo da camada do sistema de ficheiros pode-se encontrar o Buffer Cache. Esta fornece um conjunto de funções que o sistema de ficheiros pode executar, independentemente do sistema em causa. Estas funções optimizam o acesso aos dispositivos físicos, uma vez que mantém os dados por um dado período de tempo, permitindo assim retransmissões sem necessitar de os pedir novamente. [14]

Por último, o módulo Network Stack oferece um conjunto de protocolos que permitem a comunicação com outras máquinas. Este subsistema será explicado com mais detalhe na secção seguinte. [12]

4.3 Módulo de Rede no Kernel

Genericamente o módulo de rede é estruturado em camadas, incluindo nessas camadas os próprios protocolos de comunicação. Vendo de alto nível, o modulo de rede é composto

pelos network devices drives (por exemplo eth0) na base da pilha de rede, em seguida encontra-se o protocolo IP e logo acima o protocolo de transporte, normalmente TCP ou UDP. [12] O módulo de rede permite que o utilizador troque informações com outras máquinas, abstraindo-se por completo de como isso é feito, ou seja, não necessita de saber dos protocolos (TCP, UDP, etc), os interfaces de rede (eth0, wlan0, etc) ou até os dispositivos físicos (ethernet, RS-232, etc) que estão a ser usados em cada comunicação.

A figura 4.6 mostra cada uma destas camadas com mais detalhe.

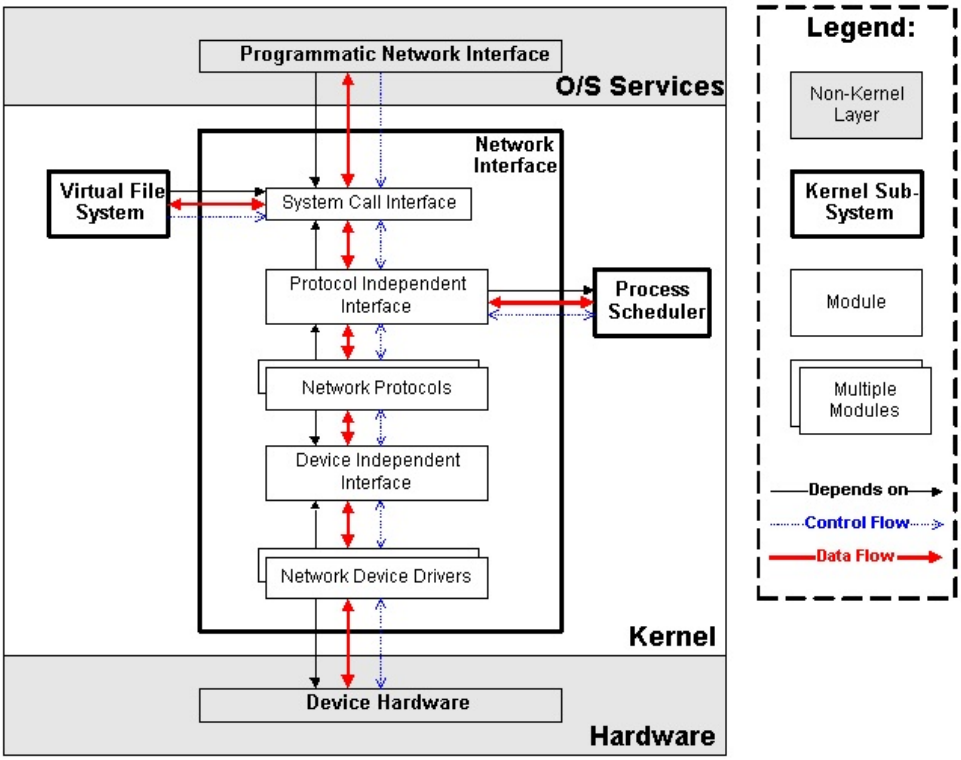

Figura 4.6: Módulo de Rede em detalhe [11]

Os Network Device Drivers ou interfaces de rede, são responsáveis por estabelecer a comunicação com os dispositivos físicos. Cada dispositivo físico possui o seu interface correspondente. Em Linux, tipicamente, os dispositivos de rede por cabo possuem Network Device Drivers com o nome eth, sendo depois atribuído um numero ao nome por cada interface físico existente, por exemplo eth0. Por outro lado, e seguindo o mesmo conceito, quando se trata de hardware que suporte rede sem fios o interface tem o nome de wlan. Cada interface de rede pode possuir um endereço IP, e é permitido ao utilizador estabelecer conexão a um dispositivo físico específico, para isso basta usar o endereço IP da interface que lhe está associada, ou seja, se um utilizador pretender enviar uma mensagem por cabo, basta indicar o IP da interface eth correspondente ao hardware pelo qual deseja enviar a mensagem.

No caso de o utilizador não especificar que hardware deve ser utilizado, a camada Device Independent Interface (DII) faz uma selecção de qual o interface a qual se deve entregar o pacote a transmitir. A selecção é feita com base no IP de destino do pacote. Todos os pacotes que chegam a esta camada fazem parte de uma de três categorias. Em primeiro lugar pode ser um pacote que tem como destino a própria maquina ou seja, localhost. Por outro lado, pode ser um pacote cujo destino é atingível directamente, isto é, o computador está ligado directamente a outra máquina. Ou então, e por último, trata-se de um pacote com um destino não alcançável directamente, ou seja, o pacote necessita de passar por dispositivos adicionais (por exemplo routers). No primeiro caso a DII faz um redireccionamento directo, devolvendo o pacote à camada superior novamente. No segundo caso, a camada usa uma tabela de routing, esta indica-lhe qual a interface que deve receber a mensagem. O ultimo caso é a situação mais complicada. Aí o sistema operativo recorre a uma gateway, previamente configurada, para saber em que interface deve depositar o pacote. [14] Além disso, esta camada possui a tarefa de fornecer uma interface comum para que a camada superior comunique com qualquer que seja a interface que esteja ou venha a ser usada.

Uma das camadas mais importantes na pilha de rede do Linux é a Network Protocol. Esta camada é responsável por construir os pacotes a enviar, segundo um conjunto de normas e regras pré estabelecidas pelos protocolos. Isto para que, quando a mensagem for recebida pela máquina destino, esta seja capaz de a descodificar seguindo o mesmo conjunto de regras. Esta camada é capaz de utilizar vários protocolos, sendo os mais comuns para transporte o TCP e UDP e para interconexão o IP. O protocolo IP foi pensado para que cada computador tenha um número associado, não podendo haver dois computadores com o mesmo número na mesma rede. Assim, todos são identificados individualmente. O protocolo TCP ou UDP foi desenvolvido para que seja possível transmitir grandes quantidades de informação, ou seja, quando é necessário dividir os dados em partes mais pequenas. O protocolo garante que quando chegam ao seu destino os dados são ordenados e juntos novamente.

A camada Protocol Independent Interface fornece uma interface genérica a todos processos User-Space ou Kernel-Space que queiram transmitir dados, independentemente do protocolo que venha a ser usado na camada seguinte.

Para que um dado utilizador envie uma dada informação para uma outra máquina, tem apenas que utilizar um socket, isto é, uma interface padronizada para comunicação entre computadores, estas interfaces permitem que qualquer aplicação User-Space utilize a pilha de rede do kernel.

Uma outra propriedade indispensável da pilha de rede é o Process Scheduler, isto permite gerir o fluxo de informação, ou seja, evita que o kernel envie mais dados para o buffer da placa de rede que aqueles que ela pode suportar, evitando assim que esta os descarte e seja perdidos dados. [11]

Capítulo 5

Soluções para a pilha protocolar FTT-SE

Este capítulo aborda o funcionamento natural da transmissão de dados na rede como o recurso a sockets. Descreve também, uma lacuna do protocolo FTT-SE não fornecendo uma interface genérica às aplicações standard. Por fim, enumera algumas possíveis soluções que foram equacionadas, expondo vantagens e desvantagens da sua utilização.

5.1 Funcionamento natural das comunicações em rede

No capítulo anterior foi exposto o processo de transmissão de mensagens através do Kernel, mas para que as mensagens cheguem ao Kernel, o utilizador tem que recorrer a sockets, como também foi referido.

Os sockets fornecem às aplicações uma interface genérica para enviar e receber informação da rede. Este método é uniforme, para qualquer que seja o protocolo que a aplicação pretenda usar. Para isso basta configurar o socket segundo três parâmetros: família, tipo e protocolo. [15]

A figura 5.1, oferece uma visão genérica de como os sockets são encaixados na pilha protocolar. Hoje em dia uma das famílias de protocolos mais usados é a PF_INET. É sobre ela que incidem quase todas as comunicações. A família PF_PACKET permite o envio de pacotes para a rede acedendo directamente ao dispositivo de rede. Por outro lado, e com outro objectivo, encontramos a família PF_NETLINK, que no Linux serve para apenas para configurar as diferentes partes da arquitectura de rede.

Neste tipo de comunicação existe sempre o conceito de Cliente e Servidor. Ou seja, existe uma máquina que possui a informação, e a disponibiliza através de um socket(servidor), e uma outra que acede aquela informação, conectando-se a esse mesmo socket(cliente). A

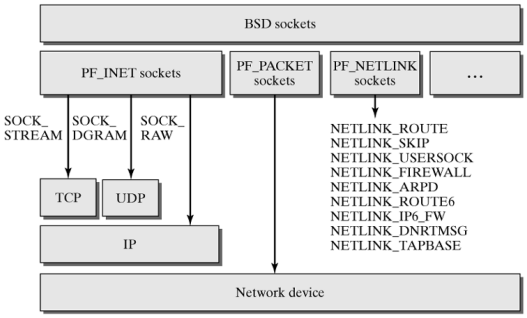

Figura 5.1: Estrutura de Sockets actualmente adoptada [15]

figura seguinte apresenta o funcionamento normal da comunicação com recurso a sockets. [15]

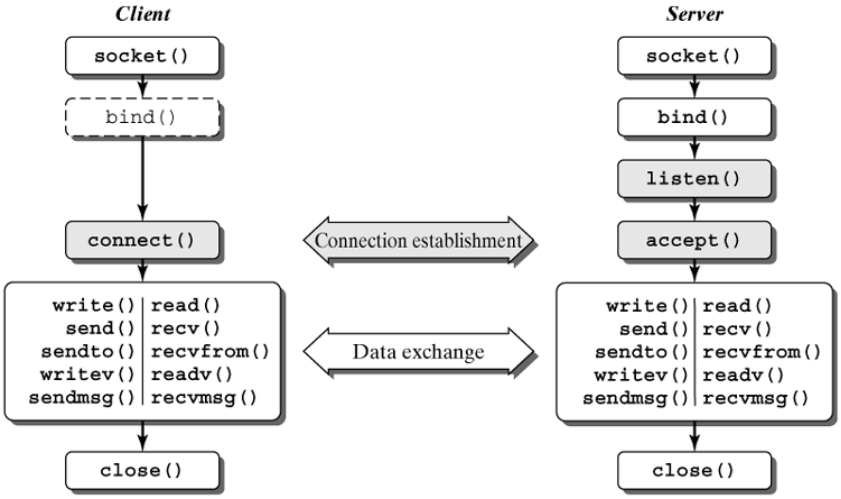

Figura 5.2: Troca de dados com o uso de Sockets [15]

As diferentes funções usadas para a comunicação de sockets serão brevemente explicadas em seguida:

- A comunicação é iniciada com a criação do socket, através da utilização da função socket(), esta necessita de informação relativa ao protocolo que vai ser usado para comunicar (por exemplo TCP ou UDP). Esta função retorna um valor inteiro que identifica o socket, o qual necessita ser especificado nas funções seguintes.
- A função bind () atribui um endereço ao socket. Quando este é criado, a função socket() configura-o apenas não lhe atribuindo nenhum endereço. Esta função deve ser executada antes que o socket possa aceitar conexões. Para os sockets de Internet este endereço consiste no IP da interface à qual o socket se vai ligar, e o número do porto no qual este vai estar a escutar. Os clientes não necessitam de executar esta função.

- Após configurado o socket a aplicação deve informar o sistema operativo que está pronta a receber conexões para isso executa a função listen(). Contudo, esta instrução só é necessária se a comunicação for orientada à conexão, isto é, seja usado o protocolo TCP.

- A conexão é iniciada por parte do cliente quando este executa a função connect(), esta função recebe o endereço do servidor e espera que este aceite a conexão.

- A função accept() é usada pelo servidor para aceitar conexões no socket previamente configurado.

Estas duas últimas funções são unicamente usadas no caso do protocolo de transporte usado ser o TCP, uma vez que a comunicação é orientada à conexão.

- Depois de configurado o socket e estabelecidas as conexões as duas aplicações podem trocar informação usando as função de write() e read().

- Depois de trocados os dados a ligação é terminada através da função close(). [15, 16]

O fluxo de informação típico quando se usam aplicações de User-Space é apresentado, de uma forma simplificada, na figura seguinte.

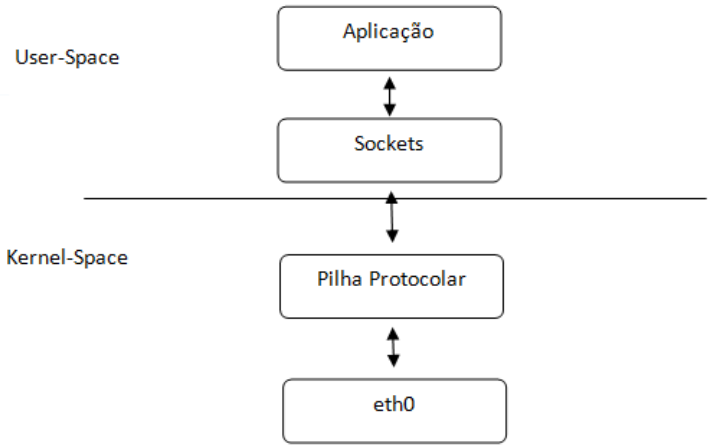

Figura 5.3: Camadas usuais para a comunicação de rede

A aplicação inicia um ou vários sockets, dependendo do que necessita, e estes vão enviar a informação para a pilha protocolar, que constrói os pacotes e entrega-os ao respectivo interface de rede, tipicamente o eth0.

5.2 A problemática e diferentes soluções FTT-SE

O protocolo FTT-SE foi pensado de modo a controlar o número de mensagens que são entregues à rede em cada momento. Contudo, nenhum protocolo ou camada usada normalmente nos sistemas operativos permite esse controlo da forma pretendida. O problema que se coloca é, onde colocar uma camada FTT-SE que seja capaz de receber a Trigger Message, processá-la e executar todas as outras tarefas que um Slave FTT-SE tem que cumprir e além disso, fornecer uma interface genérica às aplicações User-Space para que estas se possam abstrair da utilização do protocolo FTT-SE. Isto é, a comunicação com a rede deve ser feita através de sockets como se o protocolo FTT-SE não existisse.

Uma série de soluções foram equacionadas para a resolução do problema. Estas serão descritas nas subsecções seguintes.

5.2.1 FTT-SE logicamente acima do eth0

Como o controlo de mensagens que são enviadas para a rede é uma das principais características do protocolo, entendeu-se que a solução teria que passar por uma das camadas inferiores, isto é, a resolução teria que controlar a interface de rede. Assim, pensou-se que uma possível solução poderia passar por filtrar e controlar as mensagens que são entregues ao interface, para isso adicionar-se-ia uma camada à pilha de rede do Kernel, como se pode observar na figura 5.4.

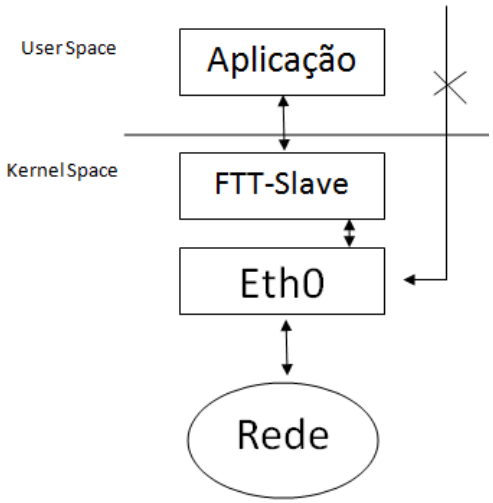

Figura 5.4: Camada FTT-SE logicamente acima do eth0

Na imagem estão suprimidas as camadas intermédias apenas por uma questão de simplificação.

A solução seria então, colocar uma camada, em Kernel-Space, imediatamente acima do eth0. Essa camada absorveria todas as mensagens que chegavam à interface, e serviria

de ponte entre as aplicações User-Space e a interface eth0. Além disso a comunicação de outras aplicações com a interface teria que ser bloqueada, para que não fossem inseridos pacotes na rede inadvertidamente.

Esta solução resolvia todas as questões, a camada FTT-SE iria executar as funções necessárias para que o protocolo funcionasse correctamente. Mas, dois problemas foram detectados. Em primeiro lugar o desenvolvimento de uma aplicação em kernel-Space, depois o reencaminhamento de todos os pacotes que a interface recebesse para a interface FTT-SE. O desenvolvimento de aplicações em Kernel-Space iria exigir um estudo detalhado de como as interfaces funcionam, mas seria exequível. Contudo, o reencaminhamento das tramas chegadas ao eth0 para o FTT-SE, não possuía uma solução aparente, ou então, iria exigir que o normal funcionamento da interface fosse alterado. O objectivo era manter a estrutura do Linux exactamente como estava, para que o protocolo pudesse ser carregado de uma forma simples, não alterando em nada o sistema operativo original.

5.2.2 FTT-SE logicamente abaixo do eth0

Uma outra solução que surgiu logo após a primeira foi construir uma arquitectura inversa. Ou seja, da mesma forma que é possível colocar a camada FTT-SE acima da de interface de rede, também pode ser possível introduzi-la abaixo, como se pode ver na figura seguinte.

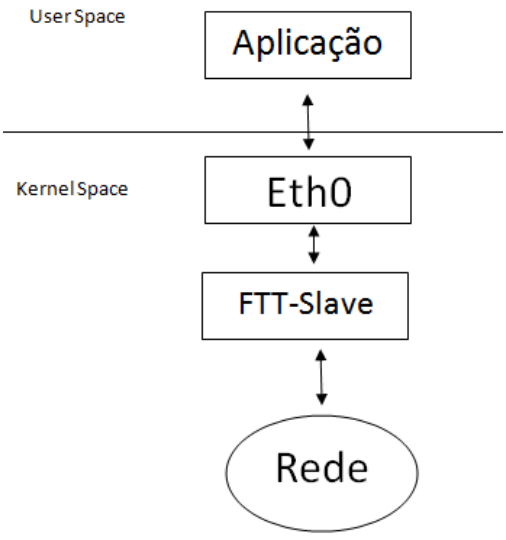

Figura 5.5: Camada FTT-SE logicamente abaixo do eth0

Esta solução baseia-se no mesmo princípio da solução anterior. Mas, aqui rapidamente foram encontrados vários problemas. O primeiro deles foi o mesmo encontrado

na solução anterior. Como fazer o eth0 comunicar com a camada FTT-SE e não com o device driver da placa? E como detectar qual a placa de rede que a maquina possui para comunicar com o device driver respectivo?

Esta solução aparente tornou-se bastante complicada. E optou-se por pensar em outras soluções antes de tentar resolver estes problemas.

5.2.3 FTT-SE e eth0 logicamente ao mesmo nível

Uma outra solução possível seria colocar os dois módulos ao mesmo nível e não como camadas com níveis diferentes.

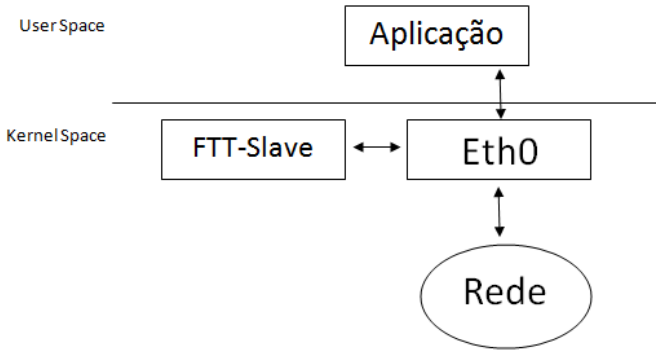

Figura 5.6: FTT-SE e eth0 logicamente ao mesmo nível

Esta solução resolveria a questão da comunicação do device driver, neste caso é o eth0 que comunica com a placa, e ao mesmo tempo comunicaria com a aplicação.

O princípio de funcionamento desta solução seria bastante simples, todas as aplicações comunicam com o eth0, este envia todos os dados que receber para o módulo FTT-Slave, e por sua vez, o FTT-Slave envia para o eth0 apenas os dados que o interface deve enviar para a rede e para as aplicações.

O problema desta solução é que era necessário fazer uma alteração do funcionamento do eth0 o que não é vantajoso como já foi visto na solução anterior.

5.2.4 FTT-SE e eth0 logicamente ao mesmo nível com recurso a IP Table

Uma outra solução equacionada baseava-se no recurso à capacidade de NAT das IP Tables em Linux. A IPtable do sistema operativo iria ser configurada para redireccionar todo o tráfego do eth0 para o FTT-Slave e assim seria o módulo FTT-SE que comunicaria com as aplicações. [17]

Desta forma, todas das aplicações comunicariam com o FTT-SE e este enviaria os pacotes para o eth0 para serem encaminhados para a rede. Por outro lado, os pacotes que o eth0 recebesse iriam ser reencaminhados para o FTT-Slave.

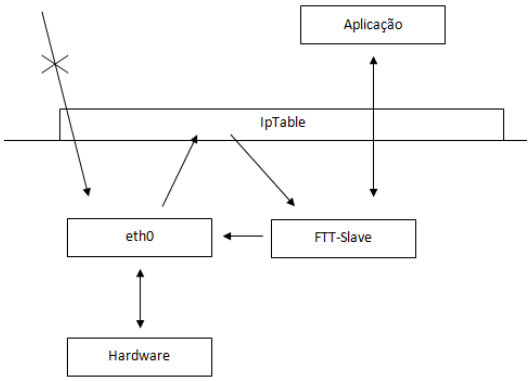

Figura 5.7: FTT-SE e eth0 logicamente ao mesmo nível com recurso a IP Table

A principal razão para não usar esta solução deve-se ao tempo que a ipTable toma para fazer os diferentes redireccionamentos. Essa limitação iria afectar directamente o desempenho temporal do protocolo.

5.2.5 FTT-SE e Tun/Tap

Uma solução que pareceu bastante interessante foi o uso de interface Tun/Tap. Esta tecnologia permite criar interfaces virtuais de rede, através de aplicações User-Space, sendo possível ler e escrever, facilmente nesses interfaces. A figura seguinte mostra a arquitectura idealizada. [11, 14]

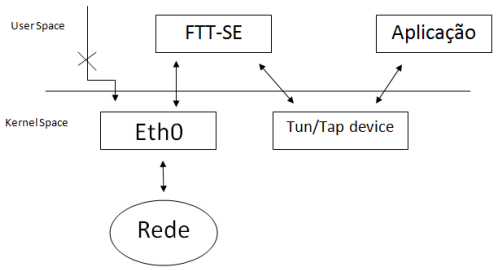

Figura 5.8: FTT-SE com uso da tecnologia Tun/tap

Todos processos relativos ao protocolo são realizados pela aplicação User-Space FTT-SE. As restantes aplicações ligam-se ao interface Tun/Tap Device que envia todos os dados para a aplicação FTT-SE. As aplicações que tentem comunicar com o eth0 terão o acesso negado através de uma configuração prévia da IPTable.

5.2.6 FTT-SE como protocolo na pilha protocolar

Uma outra solução seria introduzir o FTT-SE como protocolo, de modo a que seja possível criar sockets do tipo FTT-SE. Assim, todos os processos do protocolo seriam executados pela pilha protocolar tal como o TCP ou UDP.

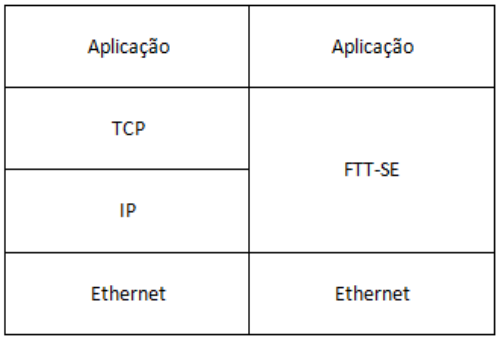

Figura 5.9: Pilha TCP/IP vs Pilha FTT-SE

Esta solução iria resolver uma série de problemas relacionados com redireccionamentos, que foram encontrados nas soluções vistas até agora. A criação de sockets FTT-SE permitiria uma interface simples, genérica e com todas as potencialidades do protocolo. Contudo, uma solução deste tipo iria exigir muito mais tempo que o disponível e um conhecimento bastante mais profundo do Kernel Linux.

Capítulo 6

Solução Universal Tun/Tap Driver

Neste capítulo é exposta a solução adoptada. Descreve em que consiste a tecnologia Tun/Tap, assim como as suas potencialidades e possíveis aplicações. É explicada também, a estrutura do programa que foi desenvolvido para a criação da interface genérica para as aplicações

6.1 Universal Tun/Tap Driver

O Tun/Tap é uma tecnologia que permite a criação de interfaces de rede virtuais, em Kernel-Space. Fornece uma plataforma genérica para envio e recepção de pacotes para aplicações User-Space. Estas interfaces podem ser vistas como simples ligações ponto-a-ponto ou como dispositivos Ethernet. Contudo, não recebe pacotes vindos de um hardware, recebe-os de uma aplicação, e em vez de os enviar para uma placa de rede, envia-os para um programa User-Space.

Quando um programa, User-Space, inicia um interface Tun/Tap, é criado e registado um dispositivo de rede com o nome de tunX ou tapX. Quando o programa termina a sua execução, os dispositivos e todos os caminhos de routing, que possivelmente tenham sido criados, são automaticamente apagados. [18]

A tecnologia tun/tap permite a criação de dois tipos de dispositivos. Um, o Tap, que opera ao nível Ethernet, ou seja, camada dois com pacotes Ethernet, e um segundo, Tun, que trabalha ao nível da camada três com pacotes do tipo IP. [16]

Os dispositivos virtuais Tun/Tap são bastante flexíveis e este tipo de interfaces pode ser usado por qualquer programa que queira receber ou enviar pacotes para a pilha protocolar. Normalmente, são usados em duas situações distintas, em redes VPN e em simulação, neste último permite que uma aplicação comunique com um falso interface.

6.2 A solução adoptada

Adoptou-se então a solução apresentada em 5.2.5.Optou-se por esta opção devido à sua capacidade de cumprir todas as restrições e sendo bastante mais simples de implementar que todas as outras soluções.

Em quase todas as outras seria necessário a criação de um interface em Kernel-Space, o que poderia representar um acréscimo na complexidade da solução do problema. Na solução em que isso não era necessário, FTT-SE como protocolo na pilha protocolar, era indispensável um conhecimento profundo do kernel, o que também iria ser bastante complicado.

Neste caso, o interface Tun/Tap Device é criado pela aplicação User-Space FTT-SE (figura 5.8), e lá encontram-se as função de read e write, para comunicar com a interface. Portanto, de uma forma simplificada, o fluxo de informação inicia-se na aplicação que gera dados e pretende transmiti-los. Contudo, a máquina está ligada a uma rede FTT-SE o que impede que a informação circule livremente. Assim sendo, a aplicação irá enviar a informação para o Tun/Tap Device, uma vez que este será o único interface de rede acessível pela aplicação. A interface guardará os dados até que a aplicação FTT-SE execute a função read(). Aí os dados serão transferidos para a aplicação.

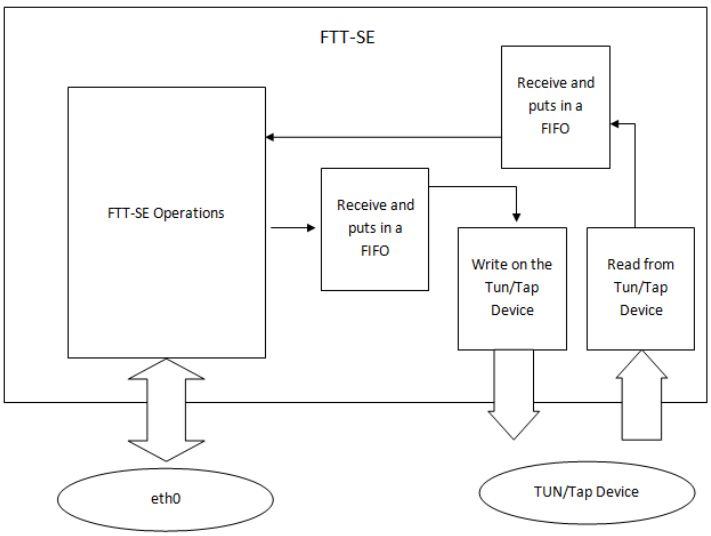

Figura 6.1: Aplicação FTT-SE

Observando a figura 6.1 é possível perceber as diferentes tarefas da aplicação. Para além das tarefas directamente ligadas ao protocolo, a aplicação possui ainda 4 threads, com funções distintas. Uma tem como objectivo ler a informação vinda do Tun/Tap device e enviar os pacotes para um buffer. A segunda, retira as tramas desse mesmo FIFO e constrói uma trama agregada, com um máximo de 65000 bytes. Este processo permite a uma maior rentabilização do EC, assim através da fragmentação efectuada pelo FTT-SE

Operations é possível escalonar, no Master, mais do que uma mensagem por ciclo. Uma outra thread tem como objectivo receber informação vinda do protocolo e armazenar essa informação num FIFO. Um FIFO é uma fila onde são armazenados os dados para posterior utilização. Por último, uma thread que tem como objectivo enviar pacotes para a interface. A utilização do FIFO, neste caso, prende-se com um certo atraso que existe quando são enviados os pacotes para a interface. Desta forma, garante-se que os dados não serão perdidos, nem o processo fica à espera que aquela função termine a sua execução. Todos os dados são armazenados no FIFO para serem enviados em seguida.

Mas continuando o fluxo de dados, a aplicação depositou então, os dados no Tun/Tap Device e estes foram recolhido pela thread Read(). Em seguida, os pacotes são enviados para a pilha do FTT-SE Operations, onde os dados são encapsulados numa trama FTT-SE, e é feito um pedido ao mestre, indicando-lhe que existem dados assíncronos a serem transmitidos. O pacote é colocado numa fila de espera. O Slave deve esperar que o mestre envie uma Trigger Message que lhe indique que pode enviar a mensagem. Quando esta chegar, o Slave envia a mensagem para o eth0.

No lado do destinatário, quando a mensagem é recebida, o FTT-SE Operations retira os cabeçalhos FTT-SE da mensagem e entrega-a ao FIFO, para em seguida ser entregue ao dispositivo Tun/Tap e a aplicação respectiva.

6.2.1 Maximum Transmission Unit

O MTU define o maior pacote que uma interface pode enviar sem necessitar de o fragmentar. [19] Em redes convencionais o valor tipicamente é de 1500 bytes. Os dados produzidos por uma dada aplicação vão sendo encapsulados à medida que vão passando pelas diferentes camadas da pilha protocolar (figura 6.2). Esse princípio foi também usado pelo protocolo FTT-SE, quando possui informação não tempo-real a transmitir.

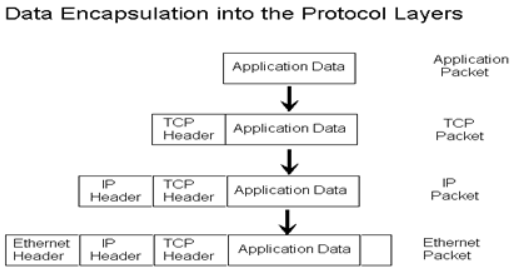

Figura 6.2: Encapsulamento dos dados [20]

Contudo, quando os pacotes possuem mais bytes que o permitido pelo MTU, estes tem que ser divididos em partes mais pequenas, como se pode ver na figura 6.3.

Quando a máquina está na rede FTT-SE é importante garantir que os pacotes não necessitam de ser fragmentados. E quando se trata de tráfego normal não se pode garantir isso, ou seja, não é possível pedir a uma aplicação que produza pacotes com o tamanho correcto para ser enviado. Além disso, o protocolo como faz o encapsulamento das mensagens normais em tramas FTT-SE, vai acrescentar os seus cabeçalhos, o que leva a que o pacote ainda fique maior.

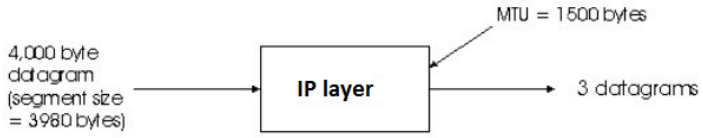

Figura 6.3: Fragmentação da camada IP [21]

Assim sendo, é indispensável que o MTU da interface virtual esteja ajustado para um valor inferior aos 1500bytes. Para que, quando a mensagem passar para a rede física seja processável sem necessitar de mais fragmentações.

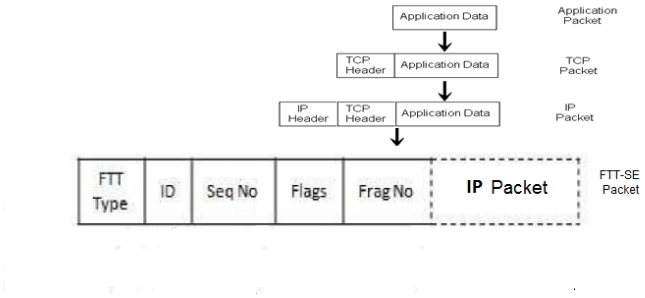

Figura 6.4: Encapsulamento FTT-SE

No capítulo 3,foram especificadas as tramas FTT-SE que os Slaves enviam e concluiu-se que, cada mensagem necessitava de 8bytes para o cabeçalho FTT-SE. Desta forma, o interface virtual FTT-SE deve estar ajustado no máximo para 1492bytes.

O MTU ao longo dos testes realizados foi ajustado para 1450bytes, apenas por uma questão de arredondamento. Para isso, foi digitado o seguinte comando.

```
ifconfig tun0 mtu 1450
```

Esta instrução deve ser executada com root, ou seja, como administrador da máquina em questão.

6.2.2 IpTables

Dentro do módulo de Network Stack do Kernel Linux existe um subsistema chamado Netfilter. As IpTables são os comandos User-Space, usados para o configurar. Esta camada cria uma série de regras capazes de filtrar pacotes e reencaminha-los. Estas são constituídas por condições, isto é, a que pacotes as regras se destinam, e por consequências, ou seja, o que vai acontecer aos pacotes em cada regra. [22] As IpTables operam ao nível IP, na camada 2 do modelo TCP/IP. Para operar abaixo desta camada existe ainda uma outra tecnologia, as ebtables. [23]

Existem 5 alturas específicas para tratamento dos pacotes que podem ser usados pelas IpTables. São elas PREROUTING, INPUT, FORWARD, POSTROUTING e OUTPUT. Assim, é possível criar sequencias de ordens para influenciar e/ou monitorizar o fluxo dos pacotes. O prerouting permite influenciar os pacotes mal eles cheguem à interface, logo após o checksum, ser validado. O Input permite alterar o pacote antes de este, ter sido entregue ao respectivo processo. O output altera o rumo natural do pacote, logo depois ter sido gerado pelo processo. O postrouting permite efectuar alterações após o pacote deixar a interface. Por fim, o Forward, permite reencaminhar pacotes para outras interfaces. [22]

No caso do FTT-SE a única coisa que é necessária é que a IpTable impeça a chegada de pacotes, vindos de aplicações ao eth0. Para que, o utilizador comum não introduza mensagens na rede inadvertidamente. Assim sendo, a única instrução necessária é:

```
iptables -A INPUT -i eth0 -j REJECT
```

Desta forma todos os pacotes de aplicações normais serão descartados antes que cheguem ao interface eth0.

Contudo, mesmo com esta regra é possível enviar pacotes para a interface. O programa FTT-SE, por exemplo, usa Raw Sockets. Este tipo de sockets abrem uma ligação directa ao interface não passando pela pilha protocolar, e assim é possível introduzir mensagens na rede.

Capítulo 7

Testes e Resultados

Este capítulo apresenta os diferentes testes realizados. Aborda levemente as tecnologias e protocolos usados. Refere os programas desenvolvidos e condições em que os testes foram efectuados, assim como esclarecimentos relativos aos resultados obtidos.

7.1 Round-Trip Time

O Round-Trip Time (RTT), ou também chamado Round-Trip Delay, é o tempo necessário para um pacote viajar na rede, desde uma fonte específica até um dado destino e voltar. Neste contexto, a fonte é o computador que inicia a transmissão e o destino é o computador remoto ou sistema que recebe o pacote e o retransmite. Tipicamente, na internet, quando um utilizador quer determinar o RTT, faz ping ao endereço IP do computador em causa.

O RTT pode ser influenciado por vários factores:

- A taxa de transferência de dados que o computador fonte possui. Ou seja, o número de bits que o computador consegue introduzir na rede por unidade de tempo.
- A natureza do meio de transmissão (fibra óptica, cabo coaxial, etc...).
- A distância física entre a fonte e destinatário.
- Número de nós entre a fonte e o destinatário, ou seja, se os dois computadores estão ligados directamente ou se existem dispositivos de interconexão entre eles.
- Quantidade de tráfego na rede.
- Número de outros pedidos a serem processados pelos dispositivos intermediários ou pelo destinatário.

- Velocidade de operação dos dispositivos intermédios e do computador destinatário.
- Possível interferência na cablagem de conexão.

Todos estes factores influenciam o valor do RTT, e este pode variar entre alguns microsegundos, em condições ideais, e alguns segundos, em condições adversas.

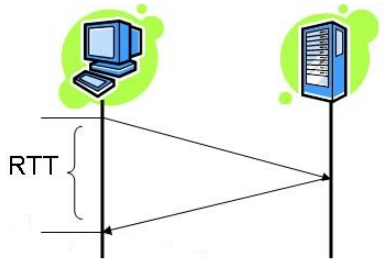

Figura 7.1: Round-Trip Delay [24]

O primeiro teste consistia em saber o RTT quando um programa tentava comunicar com outro usando uma rede FTT-SE e o interface descrito no capítulo anterior, isto quando a rede tem dois Slaves e um dado Elementary Cycle.

Foram então desenvolvidas duas aplicações Cliente/Servidor. Uma sendo o cliente que envia os pacotes e realiza o cálculo dos tempos de viagem, e uma outra que servia de servidor, retransmitia os pacotes que recebia. O teste pretendia avaliar o RTT segundo várias perspectivas. Em primeiro lugar, averiguar qual a influência do protocolo de transporte neste tipo de comunicações, e para isso utilizaram-se dois protocolos distintos, TCP e UDP. Além disso, fez-se variar a dimensão dos pacotes enviados, desta forma é possível saber se o tamanho dos pacotes tem interferência no tempo de viagem dos pacotes.

No primeiro teste feito usou-se o protocolo UDP. Para o teste foram enviados 2000 pacotes, e calculados os tempos de viagem dos mesmos. Desta forma, obtiveram-se as seguintes tabelas resumo, nas quais apenas é apresentado o valor máximo, mínimo e médio, para cada caso.

Tabela 7.1: RTT em UDP com EC=15ms

EC = 15ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	59,9	59,5	56,6	58,1	58,2	58,3	ms
Máximo:	115,9	65,5	74,7	74,7	74,7	74,6	ms
Médio:	59,8	59,7	59,8	59,8	59,8	59,7	ms

Os testes foram feitos com Elementary Cycles (EC) de 15, 30 e 50 ms, e é possível observar o aumento do RTT com o aumento do EC. Um dado importante é o facto do tamanho dos pacotes não influenciar o tempo de viagem. O diagrama seguinte apresenta as mensagens trocadas na rede aquando da realização deste teste.

Tabela 7.2: RTT em UDP com EC=30ms

EC = 30ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	117,6	119,6	119,5	108,4	110,8	118,0	ms
Máximo:	222,6	149,8	179,7	179,8	179,8	241,7	ms
Médio:	119,8	127,3	135,1	131,0	132,4	144,8	ms

Na figura foram suprimidas algumas mensagens de sinalização para simplificar a sua observação, o funcionamento correcto do protocolo FTT-SE está apresentado no capítulo 3.

Através da figura 7.2 é possível perceber o que se passa na rede FTT-SE durante este teste. O Slave1, o cliente, sinaliza o Master que possui uma mensagem, assíncrona, para transmitir, num canal já previamente configurado (mensagem número 1 da imagem). O Master, no ciclo seguinte, através da Trigger Message, notifica o Slave1 que pode enviar a mensagem, e informa o Slave2 que irá receber uma mensagem. Nesse mesmo EC, o Slave1, envia a mensagem para o Slave2, essa transmissão está representada pela mensagem número 2. O Servidor, recebe a mensagem e retransmite-a de imediato, o que implica notificar novamente o Master através da mensagem número 3 e a transmissão propriamente dita, apresentada na mensagem número 4. Isto significa que uma comunicação com o uso do protocolo UDP necessita de pelo menos 3 ECs para poder enviar um pedido e receber a respectiva resposta.

Pela observação das tabelas acima, observa-se que em média o RTT foi aproximadamente igual a 4 ECs. Tendo em conta o tempo de processamento dos pacotes pelas aplicações, pensa-se que os resultados obtidos representam o tempo necessário para a comunicação em UDP quando se está numa rede FTT-SE.

Um outro dado é o facto do tamanho dos pacotes não influenciar o RTT. Este factor prende-se com o facto de o tamanho de cada trama ser fixa, ou seja, o protocolo FTT-SE quando regista um canal para a comunicação, usa diversos parâmetros em que um deles é o tamanho do pacote, o que significa que todos os pacotes numa rede FTT-SE, para transmitir o tráfego não tempo real, têm o mesmo tamanho. Mesmo que assim não fosse, o tempo que é necessário para o Slave1 pedir autorização ao mestre para comunicar, e este responder afirmativamente, é bastante maior que aquele que o Slave1 demora a colocar uma mensagem grande na rede. E desta forma, a diferença temporal de uma mensagem pequena para uma grande torna-se desprezável quando comparada com o tempo dos ECs.

Por fim, de realçar que o máximo valor encontrado, detectou-se sempre com o pacote de menor dimensão, apesar de algum esforço na procura de uma explicação para este fenómeno, não foi encontrada nenhuma explicação plausível.

O mesmo teste foi feito com o uso do protocolo TCP, mantendo os mesmos tempos de EC e número de pacotes e tamanho.

Neste caso, com o recurso ao protocolo TCP, o RTT subiu bastante quando comparado

Tabela 7.3: RTT em UDP com EC=50ms

EC = 50ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	199,6	167,6	199,6	199,6	183,3	165,8	ms
Máximo:	371,6	249,8	299,7	249,8	299,7	299,8	ms
Médio:	242,6	204,3	244,4	218,4	237,3	231,2	ms

com o teste anterior. Agora encontram-se RTT dez vezes maiores que o EC e em algumas situações até mais, e isto deve-se à necessidade que este protocolo tem de estabelecer a comunicação antes de enviar qualquer mensagem, o que implica uma série de trocas de mensagens. Mais uma vez, observa-se que o tamanho dos pacotes não tem qualquer influência no RTT.

7.2 Streaming

O objectivo principal desta dissertação era o desenvolvimento de uma interface genérica, para o protocolo FTT-SE, que permitisse a aplicações não tempo real comunicar. Assim, testar a comunicação de uma aplicação genérica, como uma de streaming, pareceu um teste lógico.

Há alguns anos atrás, para ouvir uma música via rede era necessário descarregá-la previamente e só depois ouvi-la. Contudo, nos dias de hoje, já não é necessário transferir o ficheiro antecipadamente. É então possível ouvir uma dada música enquanto esta é descarregada. Esta tecnologia é chamada de streaming e baseia-se na compressão/descompressão e no armazenamento temporário de dados.

O streaming usa o conceito de cliente/servidor. O cliente pretende ouvir uma musica, e o servidor está a fornecê-la algures na rede. A máquina cliente envia um pedido ao servidor para que este lhe envie a música, este aceita esse pedido e inicia a transferência. Do outro lado da rede, o cliente começa a receber os diferentes dados, e constrói um buffer onde os vai armazenando. Quando o buffer é preenchido com uma quantidade de informação áudio, o cliente inicia a reprodução da mesma, mantendo a transferência em simultâneo. Os dois processos são sincronizados para que a música possa ser transferida e ouvida ao mesmo tempo, a velocidade deste processo é bastante variável e depende da ligação do cliente e do servidor. [25]

7.2.1 VLC media player

Para efectuar o streaming recorreu-se a um programa código aberto, o VLC media player. Este foi inicialmente desenvolvido por um grupo de estudantes, contando depois com a colaboração de um conjunto de programadores por todo o mundo. Devido à sua situação livre, o VLC é um programa que pode ser instalado em praticamente todos os

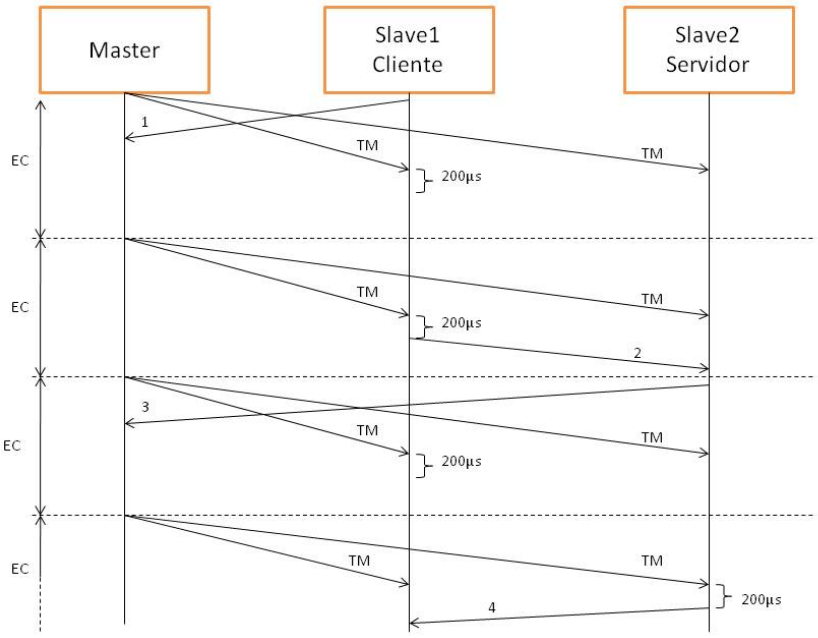

Figura 7.2: Troca de mensagens do teste RTT

sistemas operativos existentes, incluindo o Linux. Além disso, o VLC tem um conjunto de Codecs de áudio e de vídeo que permitem a visualização de quase todos os ficheiros multimedia.

Codec é uma abreviatura para "coder-decoder", e é um algoritmo que possibilita a compressão e descompressão de vídeo e de áudio. Esta aplicação permite a redução do tamanho dos ficheiros, assim sendo, ocupa menos espaço em disco e facilita a sua transferência na rede.

Além de todas estas características, o VLC é capaz de reproduzir um stream, sendo capaz de efectuar o papel de servidor e de cliente. Torna-se assim, na aplicação ideal para efectuar o teste pretendido. O VLC possui uma série de protocolos que permitem o streaming, entre eles o RTP e o RTSP. [25]

O RTP, Real-Time Transport Protocol, fornece ligações ponto-a-ponto para transmissão de dados com características tempo-real, como o vídeo e áudio, sendo possível o envio em unicast e multicast. As aplicações tipicamente usam o RTP juntamente com o UDP, para usufruírem das suas propriedades de checksum e divisão de pacotes. Do ponto de vista das aplicações o RTP e UDP formam a camada de transporte. O RTP define apenas parâmetros de fragmentação e informação de sequência em cada pacote, o transporte é feito com recurso ao protocolo UDP, que não garante qualidade de serviço em relação ao momento da entrega dos pacotes. [26] Contudo, o RTP recorre ao chamado RTCP, Real

Tabela 7.4: RTT em TCP com EC=15ms

EC = 15ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	157,7	145,1	153,6	142,9	154,9	145,7	ms
Máximo:	216,6	194,7	194,7	193,7	193,4	194,8	ms
Médio:	184,4	171,1	175,2	169,9	175,4	171,0	ms

Time Control Protocol, que lhe fornece algumas garantias. Este protocolo encarrega-se de garantir que os pacotes estão a ser entregues e controla os jitters, ou seja, as variações temporais na entrega dos dados, o que origina que a musica pare, à espera de dados que ainda não chegaram, ou então que avance repentinamente suprimindo uma parte. Por outro lado o RTP garante que o cliente recebe os diferentes pacotes com numeração e que é capaz de os ordenar, e alem disso fornece informação acerca do codec usado. [27]

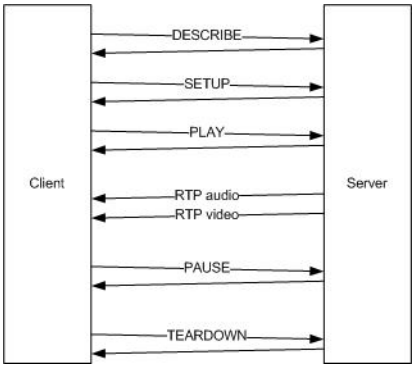

Figura 7.3: Comunicação Cliente/Servidor em RTP [28]

Por outro lado, o RTSP, Real-Time Streaming Protocol, estabelece e controla uma ou várias ligações sincronizadas para envio contínuo de dados multimedia, tais como audio e vídeo. O protocolo não entrega apenas os pacotes de dados, intercala-os com informação de controlo de transmissão. O protocolo permite ainda a distribuição da carga por diferentes servidores, sendo possível localizar diferentes streams em diferentes computadores, tal como é apresentado na figura 7.4. [29]

O protocolo usufrui das potencialidades do protocolo RTP para fazer o envio de mensagens multimedia de uma forma mais precisa, quando comparada com a que é feita com o recurso ao protocolo RTCP.

7.2.2 O Streaming áudio

O teste realizado consistiu na utilização de dois Slaves, em que um deles era o servidor de streaming e o outro era o cliente. Foi colocado um servidor VLC media player, a enviar uma música, na interface Tun/Tap criada pelo processo FTT-SE, num dos Slaves. O outro, através do VLC media player mas em modo Cliente, reproduziu a música. O programa

Tabela 7.5: RTT em TCP com EC=30ms

EC = 30ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	287,5	285,7	275,3	275,0	275,0	276,3	ms
Máximo:	359,8	379,2	388,3	388,6	388,7	388,3	ms
Médio:	327,2	336,1	345,4	344,9	343,7	348,2	ms

possui um conjunto de opções na sua interface que permitem a activação do streaming. Contudo, neste caso foi usada a linha de comandos para iniciar a acção.

Antes de mais, e após terem sido iniciados os processos FTT-SE, ou seja, iniciada a aplicação que cria a interface virtual e todas as tarefas do protocolo, é necessário configurar esses novos dispositivos, como se pode ver na imagem abaixo.

Este procedimento deve ser feito nos dois Slaves, com IPs diferentes. Por fim, a rede ficou configurada da seguinte forma.

Os dois Slaves possuíam uma interface virtual com um dado IP, e a comunicação fluía pela interface, não representada, eth0, que para além de não ter ip estava bloqueada pela IpTable.

O servidor de áudio foi colocado no Slave1, e associado ao interface Tun/Tap com o IP 192.168.6.1, utilizando o seguinte comando [32]:

```
vlc –vvv musica.mp3 –sout '#rtp{sdp=rtsp://192.168.6.1:8080/test.sdp}'
```

Desta forma o streaming é iniciado. Por outro lado o Slave2 deve conectar-se a esse servidor para iniciar o streaming. Para isso usa a seguinte instrução [32]:

```
vlc rtsp://192.168.6.1:8080/test.sdp
```

A imagem 7.8 apresenta as janelas do VLC media player a reproduzir a música. A janela inferior é a janela referente ao Slave2, que está a receber um streaming vindo do Slave1.

Uma outra questão importante é o registo recolhido quando a comunicação foi feita. A imagem 7.9 apresenta a linha de comandos do Slave Servidor, onde indica que a comunicação foi estabelecida

O slave1 com o ip 192.168.6.1, indica no relatório, que esta conectado ao ip 192.168.6.2, que é o ip respectivo ao Slave2.

7.2.3 O streaming vídeo

Seguindo a mesma arquitectura, com dois Slaves e um Master, e empregando os mesmos comandos do teste anterior, tentou-se o streaming de vídeo, já que este exigiria mais

Tabela 7.6: RTT em TCP com EC=50ms

EC = 50ms	1byte	10bytes	50bytes	100bytes	500bytes	1000bytes	
Mínimo:	475,7	506,3	478,7	498,3	464,7	455,1	ms
Máximo:	599,8	648,3	648,4	599,1	349,1	647,7	ms
Médio:	508,8	578,4	570,3	547,6	696,7	606,1	ms

dados e mais canais de ligação entre cliente e o servidor. Para que a transmissão funcionasse correctamente o VLC media player necessita de estabelecer três canais, dois de envio, para áudio e vídeo, e um de recepção de mensagens de controlo. O teste realizou-se com um EC de dez milissegundos.

A figura 7.10 apresenta a imagem que se obteve através de um streaming de baixa qualidade, ou seja, um vídeo com uma taxa de transmissão de dados de 517Kbps. A reprodução executou correctamente e não se observou nenhuma paragem.

Por último, fez-se um novo streaming, desta feita com um vídeo com uma taxa de transmissão de 1104 Kbps, o que indica que possui melhor qualidade que o anterior. O envio de mensagens por ciclo avolumou-se, como era de esperar. Na figura 7.11, observa-se esse streaming, a imagem tem melhor qualidade contudo, é visível um pouco de interferência que a olho nu quase é imperceptível. Ainda assim, observou-se algumas interrupções da reprodução do vídeo, isto deve-se a velocidade da recepção do vídeo em relação a velocidade de reprodução do mesmo.

7.3 O servidor Web

Um outro teste realizado foi a colocação de um servidor na rede FTT-SE. Usufruindo das potencialidades da distribuição Ubuntu do Linux, configurou-se um dos computadores como servidor Web, disponibilizando uma página na rede. O objectivo era ver se um navegador de internet, como por exemplo o firefox, conseguiria abrir essa página.

7.3.1 Web Server

Um servidor Web tem dois significados bem distintos. Por um lado, pode ser o computador, hardware, onde uma dada informação esta alojada. Por outro lado, pode ser o programa, software, que permite que a página esteja disponível na rede. Assim, o termo servidor Web está associado a estas duas definições.

Um sítio na Web não é mais que uma colecção de ficheiros, escritos possivelmente em html, HyperText Markup Language. Assim sendo, para que essa página esteja acessível a todos na internet é forçoso que esteja alojada num computador servidor e esse computador tem o nome de Web Server. Contudo, como já foi referido, o Web Server tem que ter um programa que disponibilize o acesso as páginas alojadas nesse mesmo computador.

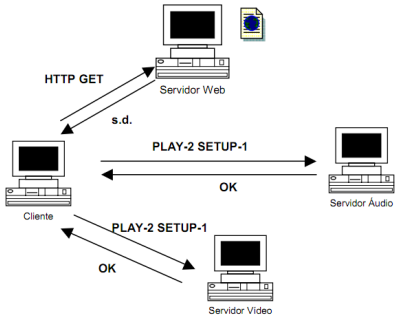

Figura 7.4: Protocolo RTSP [30]

O objectivo principal do Web Server é receber pedidos dos navegadores de internet, os clientes, e responder a esses mesmos pedidos, servidor. [33]

7.3.2 Web Server em FTT-SE

Um outro teste realizado foi a ligação de um servidor a uma rede FTT-SE e tentar aceder a páginas de internet lá alojadas. Como já foi referido anteriormente, a distribuição Ubuntu do Linux, tal como outras distribuições, permitem a instalação de software que fazem a gestão de pedidos Web, e juntamente com o computador em questão formam o Web Server. Para aceder a esse servidor, ou as paginas lá alojadas, basta colocar no endereço URL do navegador o IP do servidor. Por outro lado, o programa Servidor, a partir do momento em que é iniciado, fica automaticamente associado a todas as interfaces existentes no computador. O que significa que é possível aceder a qualquer página alojada nesse computador, através de qualquer um dos ips que esse computador tenha.

Neste caso, o servidor ficou alojado no Slave2, segundo a figura 7.7. Ou seja, tem o IP 192.168.6.2. De seguida, a partir de um navegador no Slave1, o firefox, acedeu-se ao IP do Slave2.

A figura 7.12 apresenta a janela do navegador após carregar a página de teste. O que mostra que a comunicação funcionou correctamente. Um pormenor a realçar é o endereço URL da janela que como se pode ver é o endereço do Slave2.

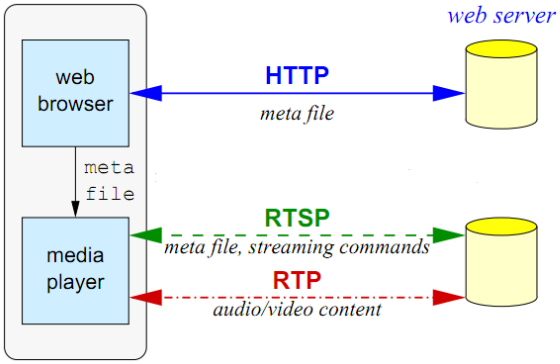

Figura 7.5: Mensagens de controlo do protocolo RTSP [31]

```
rui@rui-Aspire-5532:~$ sudo ifconfig tun0 mtu 1450
rui@rui-Aspire-5532:~$ sudo ifconfig tun0 192.168.6.2
```

Figura 7.6: Configuração da interface Tun/Tap

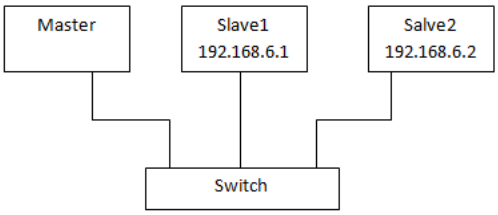

Figura 7.7: Rede FTT-SE

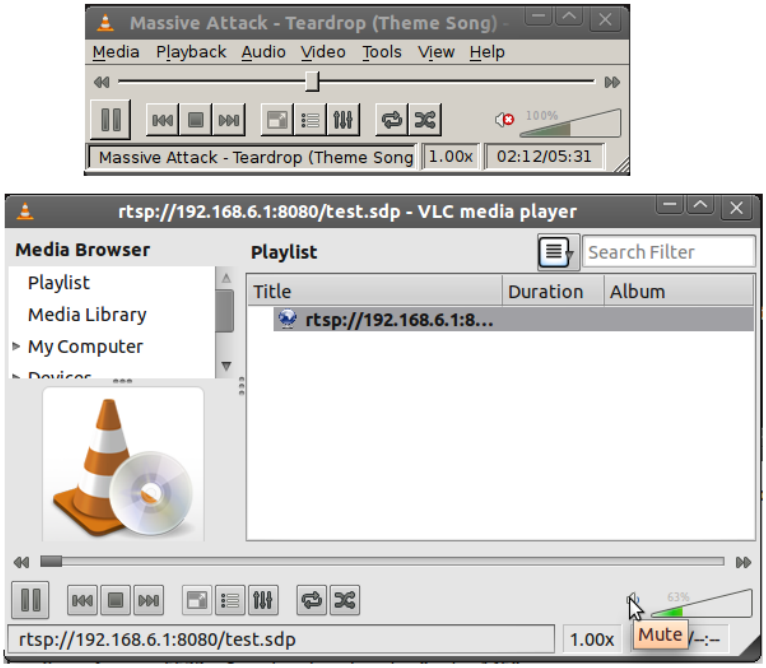

Figura 7.8: Janelas VLC media Player (Streaming)

```
[0xb7009bdc] main stream out debug: net: connecting to [192.168.6.2]:55906
[0xb7009bdc] main stream out debug: net: connecting to [192.168.6.2]:55907 from
[192.168.6.1]:51613
```

Figura 7.9: Registo VLC media player do Slave1

Figura 7.10: Streaming de Video com baixa qualidade

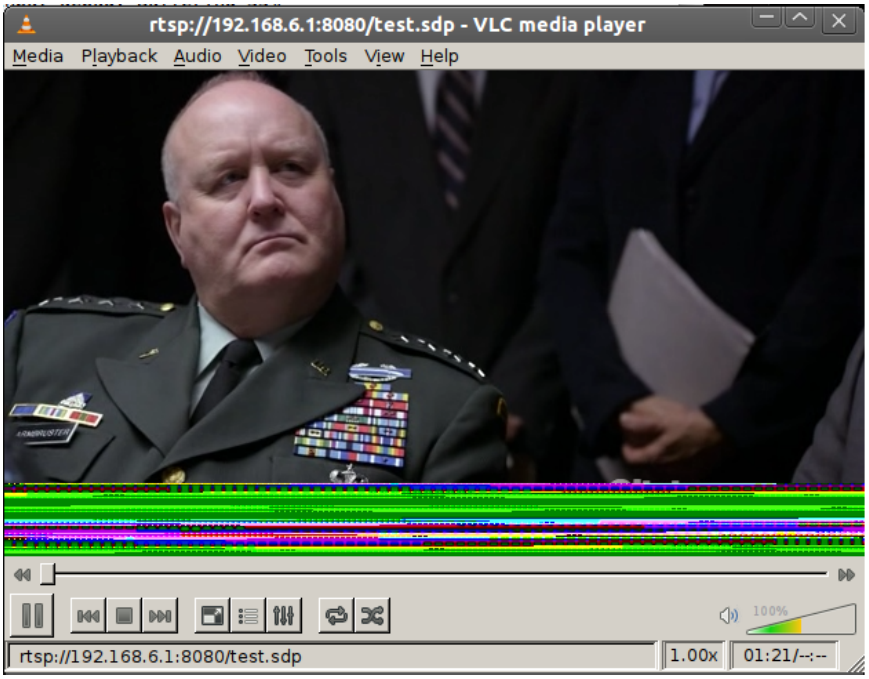

Figura 7.11: Streaming com vídeo de boa qualidade

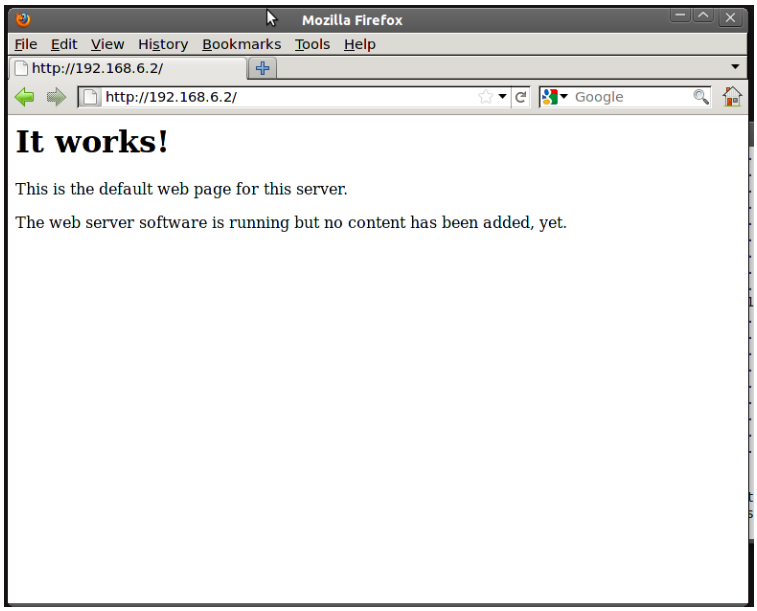

Figura 7.12: Janela Firefox a aceder a um servidor numa rede FTT-SE

Capítulo 8

Conclusão

O propósito da dissertação era criar uma interface genérica para aplicações não tempo-real, e capaz de as fazer comunicar numa rede FTT-SE de forma transparente. Inicialmente, pensou-se no desenvolvimento de uma interface em Kernel-Space, que realizasse todas as tarefas relacionadas com o protocolo, e não uma aplicação em User-Space que utiliza-se interfaces virtuais. O que significa, que em vez de uma solução em kernel-Space, foi usada a tecnologia Tun/Tap para a criação desses mesmos interfaces.

Apesar de tudo, a solução proposta cumpre os requisitos necessários, fornecendo ao protocolo, uma interface genérica de comunicação para aplicações não tempo-real standard. Este capitulo sistematiza e analisa criticamente o trabalho produzido, identificando as principais contribuições e termina com sugestões de trabalho futuro.

8.1 Comentários Gerais

Crê-se que uma interface Kernel-Space poderia trazer vantagens, principalmente em termos temporais, reduzindo desta forma o tempo de resposta. Contudo, para aplicações tradicionais esses tempos não representam atrasos significativos e não produzem atrasos visíveis para o utilizador.

A solução aplicada, quando comparada com as outras apresentadas, destacou-se devido à simplicidade da solução. Note-se, também, que o código que diz respeito ao FTT-SE (leitura da TM, envio da mensagem de sinalização, gestão da NRDB) já estava disponível, o que facilitou o trabalho. Os vários testes efectuados, obtiveram resultados positivos, sendo possível a comunicação entre máquinas usando aplicações não tempo-real, em cima do protocolo FTT-SE.

Pensa-se que este trabalho representa um passo importante no que diz respeito à aplicabilidade geral do protocolo FTT-SE, principalmente em termos de comunicações não tempo real.

8.2 Trabalhos Futuros

Uma melhoria interessante seria a utilização do protocolo FTT-SE como protocolo de transporte e de interconexão, substituindo as camadas de TCP/UDP e IP na pilha protocolar, através da criação de sockets do tipo FTT-SE.

Um outro aspecto seria a utilização de um EC dinâmico. Neste momento, o EC é configurado estaticamente, mas poderia desenvolver-se um mecanismo que ajustasse o EC em cada ciclo e que fosse apenas delimitado por um limite máximo. Desta forma, seria possível rentabilizar mais o funcionamento de um dado processo.

Uma outra sugestão consiste na impossibilidade do protocolo receber uma série de mensagens assíncronas e escalona-las todas ao mesmo tempo. Ou seja, quando um escravo FTT-SE recebe uma trama, informa o mestre que possui uma mensagem assíncrona para transmitir, antes de a enviar. O mestre efectua o respectivo escalonamento e informa o escravo que pode a enviar. Contudo, este processo não está optimizado, isto é, se o Escravo receber várias mensagens vai notificar o mestre tantas vezes quantas as mensagens assíncronas que tem para enviar. Sugere-se então, que o escravo possa notificar o Mestre de quantas mensagens tem para transmitir. E que este escalone todas as mensagens de uma só vez, se possível no mesmo EC.

Referências

[1] Paulo Portugal. Arquitecturas de sistemas de automação, 2008. FEUP.

[2] Paulo Portugal. Ethernet, 2008. FEUP.

[3] R. Marau, L. Almeida, e P. Pedreiras. Enhancing real-time communication over cots ethernet switches. Em *Factory Communication Systems, 2006 IEEE International Workshop on*, páginas 295–302.

[4] Hoang Hoai, M. Jonsson, U. Hagstrom, e A. Kallerdahl. Switched real-time ethernet with earliest deadline first scheduling protocols and traffic handling. Em *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, páginas 94–99.

[5] G. Prytz. A performance analysis of ethercat and profinet irt. Em *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, páginas 408–415.

[6] D. Gunzinger, C. Kuenzle, A. Schwarz, H. D. Doran, e K. Weber. Optimising profinet irt for fast cycle times: A proof of concept. Em *Factory Communication Systems (WFCS), 2010 8th IEEE International Workshop on*, páginas 35–42.

[7] Industrial Ethernet University. Real-time ethernet ii. Disponível em http://www.industrialethernetu.com/courses/402_2.htm acedido a 18 de Dezembro 2010.

[8] Gomes César e Pedreiras Paulo. Ftt-se: Desenvolvimento de um dissector para um protocolo de tempo real, 2010. Dissertação de Mestrado, Universidade de Aveiro.

[9] R. Marau, L. Almeida, e P. Pedreiras. Real-time communication over cots ethernet switches.

[10] Marau Ricardo. Real-time communications over switched ethernet supporting dynamic qos management, 2009. Tese de Doutoramento, Universidade de Aveiro.

[11] Ivan Bowman. Conceptual architecture of the linux kernel. Disponível em http://oss.org.cn/ossdocs/linux/kernel/a1/index.html, acedido a 18 de Maio 2011.

[12] M. Tim Jones. Anatomy of the linux kernel. Disponível em http://www.ibm.com/developerworks/linux/library/l-linux-kernel/, acedido a 17 de Maio 2011.

[13] Wikipédia. Monolithic kernel. Disponível em http://en.wikipedia.org/wiki/Monolithic_kernel, acedido a 20 de Maio 2011.

[14] Wolfgang Mauerer. *Profissional Linux Kernel Architecture*. Wiley Publishing, Inc., 2008.

[15] Hartmut Ritter Daniel Müller Marc Bechler Klaus Wehrle, Frank Pählke. The linux networking architecture: Design and implementation of network. Disponível em http://ngn.ee.tsinghua.edu.cn/~lujx/linux_networking/index.html, acedido a 24 de Maio 2011.

[16] Wikipédia. Berkeley sockets. Disponível em http://en.wikipedia.org/wiki/Berkeley_sockets, acedido a 17 de Abril 2011.

[17] Kishan Thomas. Port forwarding. Disponível em http://www.hackorama.com/network/portfwd.shtml, acedido a 8 de Abril 2011.

[18] Maxim Krasnyansky. Universal tun/tap device driver. Disponível emhttp://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt, acedido a 25 de Maio 2011.

[19] Cisco. Understanding maximum transmission unit (mtu) on atm interfaces. Disponível em http://www.cisco.com/en/US/tech/tk39/tk371/technologies_tech_note09186a00800c8279.shtml, acedido a 18 de Maio 2011.

[20] W. Richard Stevens. Network protocol categories. Disponível em http://www.codewit.com/network/basic/netnetworkcategories.php, acedido a 23 de Maio 2011.

[21] James F. Kurose e Keith W. Ross. *Computer Networking A Top Down Approach Featuring the Internet*. Wiley Publishing, Inc., 2000.

[22] Gregor N. Purdy. *Linux iptables Pocket Reference*. O'Reilly Media, 2004.

[23] Bart De Schuymer. Ebtables. Disponível em http://ebtables.sourceforge.net/, acedido a 17 de Maio 2011.

[24] Wikibooks. Computer networks/http. Disponivel em http://en.wikibooks.org/wiki/Computer_Networks/HTTP, acedido a 04 de Maio 2011.

[25] Felipe Cesar Cardoso. Conceitos de rede virtual privada para streaming seguro de vídeo. Universidade São Franscisco, 2010.

[26] H. Schulzrinne. A transport protocol for real-time applications. Disponível em http://www.ietf.org/rfc/rfc1889.txt, acedido a 07 de Maio 2011.

[27] Colin Perkins. Rtp: Audio and video for the internet. Addison Wesley, 2008.

[28] W3C. Ua server rtsp communication. Disponível em http://www.w3.org/2008/WebVideo/Fragments/wiki/UA_Server_RTSP_Communication, acediddo a 06 de Junho 2011.

[29] H. Schulzrinne. Real time streaming protocol. Disponível em http://www.ietf.org/rfc/rfc2326.txt, acedido a 04 de Junho 2011.

[30] Albuquerque Márcio Albuquerque Valeriana, Roncero Marcelo. Monitoramento do protocolo rtsp (real time streaming protocol) utilizando ntop (network top). Centro Brasileiro de Pesquisas Físicas, 2008.

[31] Henning Schulzrinne. Internet media-on-demand: The real-time streaming protocol. Columbia University, 2001. N.Y., U.S.A.

[32] VLC. Videolan. Diponível em http://wiki.videolan.org/, acedido a 07 de Junho 2011.

[33] Webmaster. what is web server? Disponível em http://www.webdevelopersnotes.com/basics/what_is_web_server.php, acedido a 11 de Junho 2011.