ADAPTAÇÃO DO FORETHOUGHT AO FREEBSD

António Alves, Gonçalo Quadros e Edmundo Monteiro
Departamento de Engenharia Informática, Universidade de Coimbra - Pólo II
3030 COIMBRA
Tel.: +351-39-790000, Fax: +351-39-701266, E-mail: aalves@dei.uc.pt

Sumário
A distribuição ForeThought[1] da FORE Systems, Inc., disponibiliza um conjunto de serviços que permitem suportar de uma forma transparente aplicações IP sobre ATM. Este trabalho descreve a adaptação para o sistema operativo *FreeBSD* ao abrigo de um acordo entre o fabricante e o Laboratório de Comunicações e Serviços Telemáticos[2]. No trabalho foi dada especial atenção às questões de performance, nomeadamente no que respeita ao transporte dos dados através das várias camadas protocolares tendo sido construídos e testados novos mecanismos (em ambiente *FreeBSD*) para esse efeito.

1. INTRODUÇÃO

O principal problema surgido no início da constituição de uma rede piloto ATM no **LCST**, estava relacionado com a falta de *device drivers* para o sistema operativo tradicionalmente usado no laboratório para projectos de estágio e investigação, o *FreeBSD*. Este sistema operativo é um *clone* UNIX com base no código BSD da Universidade da Califórnia.

Deste modo foi necessário iniciar um projecto que contemplasse a criação de um *device driver* para placas de rede ATM numa arquitectura Intel ix86 (bus PCI) e sistema operativo *FreeBSD* [5][6][7]. Como base para o início do projecto, foi assinado um acordo com a empresa FORE Systems, Inc. ao abrigo do programa ``University and Research (U & R)``. Este programa permite o acesso ao código fonte da distribuição ForeThought para a plataforma de referência *SunOS*.

Este artigo descreve as etapas mais relevantes do projecto de adaptação do código original do ForeThought ao *FreeBSD*. Tal inclui a construção de novos mecanismos (em ambiente *FreeBSD*) para transporte de informação na pilha protocolar do sistema operativo bem como testes de avaliação da sua eficácia e ainda a estruturação de um conjunto de tarefas executadas na altura da inicialização da placa.

Uma breve descrição da estrutura do ForeThougth é feita na secção 2. De seguida são apresentadas respectivamente nas secções 3, 4 e 5 as alterações feitas ao nível da inicialização do *hardware*, na gestão dos *buffers* e recepção de *pdu's*. Por fim o desempenho do *driver* é analisado na secção 6.

2. ESTRUTURA DO FORETHOUGHT

A figura 1 mostra os componentes do ForeThought, incluindo interface com o *hardware*, *driver* e utilizador [1].

CLASSICAL IP - O Classical IP é uma norma, criada pelo IETF[3], para transporte de tráfego IP sobre ATM. O RFC 1577 descreve um ambiente que é uma tentativa para emular uma rede em ambiente de difusão [13].

FORE IP - O FORE IP é uma implementação simples de transporte de tráfego IP sobre ATM. Utiliza o serviço *connectionless* incorporado no *software* dos comutadores

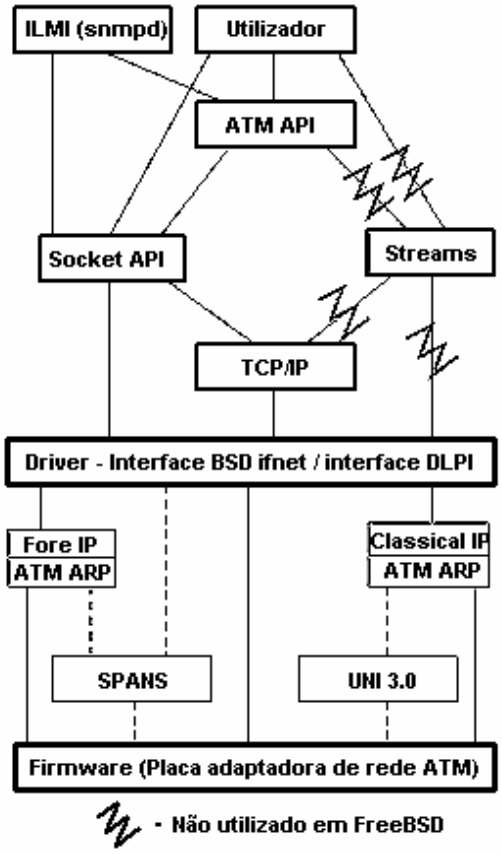

Figura 1 Componentes do ForeThought e Interface

ATM da FORE para fazer *broadcasts* e utilizá-los na resolução de endereços [3].

UNI 3.0 - O Uni 3.0 é um protocolo de sinalização[4] normalizado pelo ATM Forum. A sinalização é implementada tanto no *software* dos equipamentos terminais como nos comutadores.

[1] ForeThought é o nome dado pela Fore (http://www.fore.com) à distribuição de *software*, incluindo *drivers* para placas adaptadoras ATM e utilidades de configuração. Este pacote de *software* em binário é fornecido com o *hardware*.

[2] Laboratório de Comunicações e Serviços Telemáticos do Departamento de Eng. Informática da Universidade de Coimbra (http://www.dei.uc.pt)

[3] IETF - Internet Engineering Task Force

[4] O ATM é uma tecnologia orientada a ligações (ponto a ponto). O protocolo de sinalização fornece um conjunto de facilidades que permite que outro *software* estabeleça as ligações de rede necessárias.

SPANS - O SPANS[5] é um protocolo de sinalização desenvolvido pela FORE antes do aparecimento de normas de sinalização, como o UNI 3.0.

ATM ARP - Protocolo de resolução de endereços (Variante do ARP usado sobre Ethernet).

ILMI - O ILMI[6] é um agente de troca de informação de gestão. É principalmente usado em operações de registo de endereços na sinalização UNI.

O interface com o sistema operativo é realizado por intermédio do *device driver*. Dependendo do sistema em causa o *driver* fornece suporte através de um interface *sockets* (típico de um sistema BSD) e/ou *streams* (Unix sistema V). Deste modo na figura 1 podemos verificar que a adaptação ao *FreeBSD* não utiliza o suporte *streams*. Por sua vez o interface com o *hardware* é feito por intermédio do *firmware*. Este código corre num processador próprio da placa adaptadora ATM fornecendo um interface uniforme ao *driver*. A interacção com o *firmware* é realizado por comandos e filas. A este interface a FORE Systems dá o nome de **AALI**[7].

3. A INICIALIZAÇÃO DO HARDWARE

A placa de rede ATM PCA200E (155Mb/s UTP5) da FORE utilizada no projecto é uma placa com bus PCI. Esta placa tem um processador próprio, um banco de memória RAM e outros componentes auxiliares. A este conjunto, incluindo o código do *firmware* executado por este processador a FORE dá o nome de *Cell Processor* (CP). A comunicação entre a CPU no *host* e o CP é feita através da partilha da memória do CP e memória central no *host*. O CP faz os acessos à memória do *host* utilizando DMA (Direct Memory Access).
De modo a que seja completado o processo de inicialização da placa e do *device driver* é necessário carregar previamente o respectivo *firmware*. O método de carregamento utilizado pela FORE foi objecto de restruturação de modo a garantir um maior sucesso na reserva inicial de recursos durante o processo de inclusão do *device driver* no *kernel*. Na secção seguinte é apresentada uma breve descrição das alterações efectuadas no processo de carregamento do *firmware*.

3.1. O FIRMWARE

Após uma inicialização correcta do processador da placa (i960), o código no *driver* espera que o *firmware* seja carregado para a placa adaptadora. Se o carregamento do *firmware* for executado com sucesso, o *driver* envia o comando *initialize* para a placa. Este comando é executado apenas uma única vez e é responsável pela inicialização de vários parâmetros inerentes ao interface do *driver* com o *firmware* - por exemplo, tamanhos de filas, número de *buffers* configurados no *host*, etc.
O carregamento do *firmware* tal como é utilizado na plataforma *SunOS*, é feito através de um utilitário em *user space*. Este utilitário é executado pelo utilizador ou através de um dos *scripts* presentes na directoria */etc*, logo após o arranque do sistema operativo. Um dos grandes problemas deste método de carregamento de *firmware* prende-se com o facto de o mesmo só ser feito após o sistema arrancar completamente. Como o *driver* só reserva os recursos de memória necessários aos *buffers* após um carregamento do *firmware* com sucesso, pode-se dar o caso de já não existirem nessa altura blocos de memória contínuos em memória física suficientes para os *buffers* do *driver*. Se isto acontecer o *driver* aborta.
Por outro lado se o processo de reserva dos *buffers* de memória for feito na altura da incorporação do *driver* no ambiente do sistema operativo (*attach*), existe uma maior probabilidade de reservar com sucesso os blocos de memória contíguos necessários.
Tendo em conta esta última aproximação, foi redesenhada a maneira de carregar o *firmware* na versão do ForeThought para o *FreeBSD*. Basicamente foram criadas algumas rotinas que permitem carregar o *firmware* a partir do *kernel* e não do espaço de utilizador, tal como é apresentado no código original. Deste modo é possível carregar o *firmware* logo após o *attach* do *driver* ao sistema operativo, passando a reserva de buffers a ser feita antes do início de qualquer processo em espaço de utilizador.

4. GESTÃO DE BUFFERS

Um conceito fundamental na arquitectura de rede do BSD são os *buffers* de memória ou *mbufs* [4]. Estes *buffers* são utilizados para guardar vários tipos de informação incluindo os dados enviados e recebidos pelos interfaces de rede. Existem vários tipos de *mbufs*, para serem utilizados de acordo com o tipo e quantidade de informação a transportar. Para grandes quantidades de informação temos ainda o conceito de *buffers* externos, que funcionam como auxiliares dos *mbufs*. Podemos ter dois tipos de *buffers* externos; os *clusters*, cujo tamanho máximo é de 2048 bytes e cuja gestão dos recursos utilizados é feita pelo próprio sistema operativo, e os *user buffers* em que o tamanho é definido pelo programador e cuja gestão é também da responsabilidade deste. Neste último caso o programador deve implementar algumas funções para fazer a sua gestão. Estas funções são responsáveis pela libertação dos recursos de memória reservados inicialmente para o *buffer* e pela gestão das referências relativas aos *user buffers*.

5. A RECEPÇÃO DE PDU'S

A recepção de pacotes de dados num interface de rede depende de uma rotina principal de serviço de interrupções. O ponto de entrada desta rotina é configurada no momento em que é feito o *attach* do *device driver* ao sistema operativo. As interrupções são geradas pelo *hardware*, no presente caso pela placa de rede ATM, logo que tenha uma ou mais *pdu's* completas para entregar ao *driver*. Os dados que formam a pdu são transferidos para a memória do *host* por DMA. A figura 2 dá uma ideia do fluxo de dados através do código do *device driver*, até estes serem entregues às camadas superiores.
O objectivo desta secção é descrever dois métodos de transporte de dados através do subsistema de rede.

[5] Simple Protocol for ATM Network Signaling (Interface Utilizador-Rede)
[6] Interim Local Management Interface
[7] ATM Adaptation Layer Interface [2]

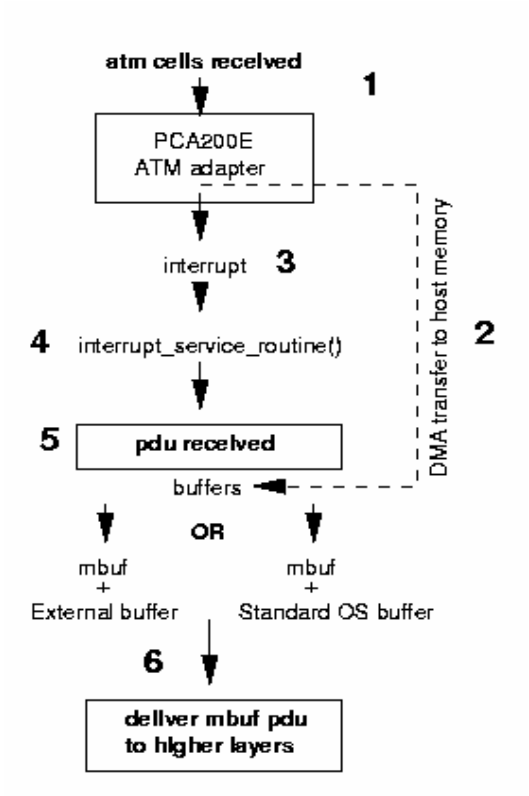

Figura 2 Recepção de pdu's

5.1. MBUFS COM BUFFERS FORNECIDOS PELO SISTEMA OPERATIVO

O *device driver* no momento de inicialização e incorporação no ambiente do *kernel*, reserva para seu uso uma determinada quantidade de *buffers* contínuos em memória física. Estes *buffers* são utilizados pela placa adaptadora ATM, para construir as *pdu's* a partir da parte de carga das células recebidas da rede. Vimos anteriormente que a placa utiliza o processo de DMA para escrever nestes *buffers*. Após a interrupção, estes *buffers* contém as *pdu's* completamente formadas. A partir deste momento o *driver* tem de transportar estas *pdu's* para as camadas superiores. A primeira aproximação a este problema foi resolvida de uma maneira segura utilizando os *buffers* geridos pelo próprio sistema. Basicamente foram utilizados os campos de dados disponíveis nos *mbufs* bem como os *buffers* auxiliares que dão pelo nome de *clusters*. Ou seja, as *pdu's* eram copiadas dos *buffers* externos utilizados pela placa adaptadora para os campos de dados dos *mbufs* e *clusters*. Este método, no entanto, sendo a aproximação mais simples não era o mais eficiente como mais à frente se mostra.

5.2. MBUFS COM BUFFERS EXTERNOS

Uma outra maneira, bastante mais eficiente para transportar as *pdu's*, é conseguir que os *mbufs* transportem directamente os *buffers* utilizados pela placa adaptadora para construir as *pdu's*. Este é o método utilizado pelo código de referência do *SunOS*. Acontece que os recursos do sistema utilizados pelo código do *SunOS* para transportar estes *buffers* não têm uma equivalência directa no *FreeBSD*. Ou seja, é possível utilizar o mesmo tipo de método no *FreeBSD* mas utilizando uma implementação diferente.

Após algum estudo do processo a utilizar, chegou-se à conclusão que seria possível transportar os *buffers* externos, reservados pelo *driver*, para as camadas superiores. Para isso foram necessárias algumas rotinas auxiliares e pequenas alterações no código. O problema maior que existe na utilização destes *buffers* externos é que a gestão dos mesmos não é da responsabilidade do sistema operativo. A gestão tem de ser feita por rotinas auxiliares implementadas no *device driver*.

6. CARACTERÍSTICAS E DESEMPENHO DO DRIVER

De modo a caracterizar o nível de desempenho da versão actual do *driver* tal como se encontra implementado em *FreeBSD* são apresentados alguns resultados obtidos com recurso a testes utilizando TCP como protocolo de transporte, mediante a utilização da aplicação de *benchmarking* **netperf** [12]. O Netperf é uma ferramenta de domínio público desenvolvido pela *Hewlett-Packard*. A sua estrutura é baseada no modelo cliente-servidor. Esta aplicação permite que o utilizador escolha uma determinada quantidade de tempo para a transferência de dados através da rede. A taxa de transferência resultante é calculada pela quantidade de dados transferidos a dividir pelo intervalo de tempo utilizado. A implementação do netperf evita o recurso a acessos a memória secundária bem como a necessidade de grandes quantidades de memória, já que retransmite continuamente o mesmo *buffer* de memória até terminar o tempo especificado no teste.

Os resultados obtidos são valores médios calculados com base em vários grupos de testes, tendo cada teste uma duração de dez segundos.

6.1. TESTE EM MODO LOOPBACK

Este teste foi realizado em ambiente isolado com recurso a uma ficha de *loopback*. A máquina utilizada foi um Pentium 233 MMX. Teve por finalidade demonstrar o aumento de performance permitido pelo novo tipo de transporte de *pdu's* com o auxílio de *buffers* externos controlados pelo próprio *driver*. Para tal o teste foi executado utilizando as duas versões do *device driver*:

- Versão inicial que utiliza os *buffers* disponibilizados pelo próprio subsistema de rede do *FreeBSD*;
- Versão melhorada que utiliza *buffers* externos

Os resultados obtidos mostraram uma melhoria de desempenho de cerca de 11% da primeira para a segunda versão.

6.2. TESTE EM REDE LOCAL

Este teste teve como objectivo principal, dar uma ideia da performance máxima que é possível obter em tráfego TCP sobre FORE IP utilizando o software desenvolvido. O teste foi realizado tendo como emissor um Pentium II 266Mhz e receptor um Pentium 233 MMX através de um comutador ASX200WG.

A figura 3 mostra um gráfico da taxa de transferência do TCP com base nas variações do tamanho do *buffer* de *socket*. Esta variação implica a variação do tamanho da janela anunciada pelo TCP em valor idêntico. De salientar que o *FreeBSD* apresenta como tamanho por defeito para o *buffer* de *socket*, 16.384 bytes. Este valor corresponde como foi referido atrás, ao tamanho da janela do TCP e não é o ideal para a utilização com ATM. Como se pode verificar na figura o aumento da janela do TCP para um máximo de 64Kbytes corresponde

a uma melhoria significativa na taxa de transferência conseguida. Nesse caso consegue-se transmitir a 120 Mb/s, o que equivale a uma melhoria de cerca de 42% em relação ao débito possível quando é utilizada a janela por defeito (16Kbytes). Este é um resultado bastante interessante considerando que o máximo teórico possível com o tipo de encapsulamento utilizado é de 134,6 Mb/s e ainda os resultados de outras implementações de IP sobre ATM em computadores pessoais [9][10][11]. Como exemplo pode referir-se que com um *driver* desenvolvido em *Linux* para a mesma placa adaptadora (PCA200E) o débito máximo conseguido para tráfego TCP foi de 110 Mb/s, ou seja um valor inferior ao que foi conseguido neste projecto apesar de terem sido utilizadas para a avaliação do *driver Linux* máquinas normalmente consideradas com características mais interessantes para esse efeito (Pentium Pro) [14].

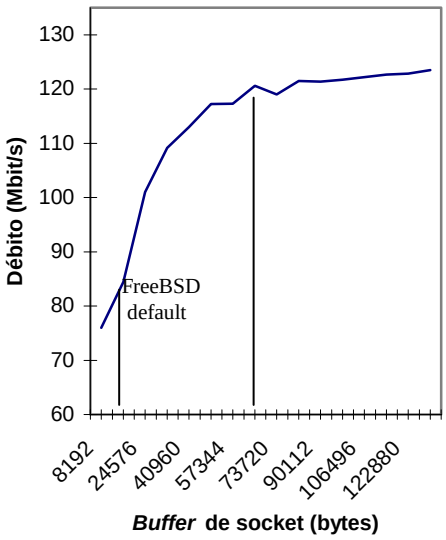

Figura 3 Débito do TCP/ForeIP

7. CONCLUSÃO

No início do projecto surgiram algumas dúvidas relativamente ao nível de desempenho que seria possível alcançar com a adaptação do código ForeThought para um sistema operativo *freeware* numa arquitectura intel ix86, já que a distribuição da FORE era tipicamente implementada em *workstations*. Este projecto permitiu não só desfazer essas dúvidas como também tornou possível a disponibilização da distribuição ForeThought da FORE Systems para um ambiente bastante divulgado na comunidade de investigação universitária.

Para além disso tornou possível a criação de postos de trabalho ATM com custos bastante inferiores às tradicionais *workstations* sem perdas importantes de desempenho.

12. REFERÊNCIAS

[1] Fore Systems, Inc. *ForeThought Porting Guide*, Novembro 96

[2] Fore Systems, Inc. *Programmers Ref. Manual for AALI interface*, Novembro 96

[3] Fore Systems, Inc. *FORE IP Overview*, Novembro96

[4] Stevens, R. *TCP/IP Illustrated*, Volume2, Capítulo II, Addison-Wesley Publishing Group, 1995

[5] Egan, J. *Writing a Unix device driver*, 2ª edição, John Wiley & Sons, Inc., 92

[6] McKusick, M. *The Design and Implementation of the 4.4BSD Operating System*, Addison-Wesley Publishing Company, 96

[7] Adams, P. *Writing Unix device drivers in C*, Prentice Hall 93

[8] Cranor, C. *Integrating ATM Networking into a BSD Operating System*, WUCS, 96

[9] Almesberger, W. *High-speed ATM networking on low-end computer systems*, LRC, Agosto 95

[10] Keshav, S. *Native-mode ATM in FreeBSD: Experience and Performance*, AT&T

[11] Andrikopoulos, I. *TCP/IP Throughput Performance Evaluation for ATM Local Area Networks*, Univ. Surrey UK

[12] Hewlett-Packard Company, *Netperf: A Network Performance Benchmark*, Fevereiro 96 (http://www.netperf.org)

[13] Laubach, M. *Classical IP and ARP over ATM*, RFC 1577, Hewlett-Packard Laboratories, Janeiro 94

[14] Linux support for Fore Systems PCA-200E, http://os.inf.tu-dresden.de/project/atm/