Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Captura de tráfego de rede de um processo com base no PCAP

Nuno Miguel Galvão Martins (29603)

Lisboa
(2011)

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Captura de tráfego de rede de um processo com base no PCAP

Nuno Miguel Galvão Martins (29603)

Orientador: Prof. Doutor Vítor Manuel Alves Duarte

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

Lisboa
(2011)

Resumo

A monitorização de aplicações permite analisar e compreender o comportamento das mesmas, sendo possível monitorizar durante execuções reais os seus recursos computacionais, nomeadamente a utilização do *cpu*, da memória, dos dispositivos de *IO* (incluindo os dispositivos de rede), etc.

As ferramentas de monitorização de rede, em geral, utilizam a biblioteca *LibPCap*, de modo a capturar os fluxos das interfaces de rede. Como esta biblioteca é genérica, permite abstrair o modo como o sistema de operação lida com a captura dos fluxos, sendo que no *Linux* o suporte é garantido pelo sistema de captura e filtragem de pacotes de rede *Linux Socket Filter*.

De modo geral, o volume de tráfego obtido pela monitorização é elevado, sendo necessário filtrá-lo, de forma a que apenas os fluxos de dados relevantes sejam transmitidos para o monitor, em nível utilizador. Quer no libPCap quer no suporte dos sistemas não existe suporte para a captura baseada nas interacções dos processos. Esta é uma funcionalidade bastante útil para os utilizadores e com vantagens, como na sobrecarga do sistema e desempenho.

A solução proposta pretende resolver esta problemática, através da Monitorização de Rede Orientada ao Processo (*MRoP*), estendendo o sistema de monitorização de rede do *Linux*, por meio da introdução de um módulo no núcleo, permitindo capturar as interacções das aplicações e filtrando os fluxos de rede da aplicação alvo. Esta solução foi avaliada funcionalmente, verificando que apenas os fluxos de dados pretendidos existiam na captura. Para além desta avaliação, foi também realizada a avaliação de desempenho, a qual dependeu do conjunto de pacotes capturados, relativos ao processo alvo.

Palavras-chave: Monitorização, rede, aplicação, LibPCap, captura de pacotes, instrumentação do núcleo de sistema, filtro

Abstract

The monitorization of the applications allow us to understand and analyse their behaviour, during real executions, their computation resources, namely the cpu, memory, IO devices (which includes the network devices), etc.

Network monitoring tools, in general, use the libpcap library, in order to capture the flows of network interfaces. Since this is a generic library, it allows to abstract the way the operating system deals with captured and filtering network packets flows. In Linux, this support is guaranteed by Linux Socket Filter. Overall, the traffic's amount obtained by monitoring is high, being necessary to filter it, so that only relevant data streams can be transmitted to the monitor, at user level. Either in libpcap, or on systems' support, there is no capture support based on processes interactions. This is a very useful and advantageous feature for users, such as in the system overhead and performance.

The presented solution aims to solve this problematic through *Monitorização de Rede orientado ao Processo* (Network Monitoring oriented Process), extending the Linux network monitoring system, through the inclusion of a kernel module, allowing to capture the applications' interactions and filtering network flows of the target application. This solution was functionally assessed, checking that only the desired data streams could be captured. Besides this evaluation, performance was also assessed, which depended on the number of captured packets, regarding the target process.

Keywords: Monitoring, network, application, LibPcap, packet capture, kernel instrumentation, filter, filtering

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

1.1 Contexto

A monitorização de uma aplicação destina-se normalmente à obtenção de informações relevantes acerca do seu comportamento durante a execução, mas pode igualmente servir para verificar a correcção, recursos atribuídos e usados, desempenho de execução, etc.

A maioria dos sistemas de operação generalistas apresentam métodos de monitorização, devido à importância de que estes se revestem, quer no desenvolvimento das aplicações, quer na gestão destes sistemas. Se algumas ferramentas são específicas na monitorização de determinados recursos (como a biblioteca *LibPCap*, que é específica nas interacções com o exterior utilizando interfaces de rede), outras são mais generalistas podendo monitorizar recursos diversos (como o *LTT*, *OProfile*, etc.).

Esta dissertação foca-se na monitorização no núcleo das interacções das aplicações através de interfaces de rede.

O dinamismo das aplicações pode dificultar bastante o processo de monitorização. Esta situação é particularmente sentida ao nível da monitorização dos dispositivos de *IO*, e interfaces de rede. Nas aplicações onde as ligações e interacções são bastante dinâmicas e efectuadas em cada execução de um modo nem sempre previsível, a monitorização é particularmente difícil de ser realizada.

A monitorização no núcleo pode gerar um elevado volume de dados que podem

mostrar-se irrelevantes para as análises efectuadas. Se considerarmos que a transferência dos dados gerados pela monitorização, do núcleo para o nível utilizador, onde se localizam as ferramentas que procedem à análise dos mesmos, é necessário efectuar a cópia destes e proceder à sua filtragem, de modo a obtermos apenas os eventos pretendidos. Se analisarmos atentamente este processo de monitorização, chegar-se-à à conclusão que este irá produzir uma sobrecarga sobre o sistema. Assim, de forma a capturar apenas dados relevantes e simultaneamente minimizar os efeitos da monitorização, são aplicados filtros logo que possível, no núcleo do sistema. Considerando que aplicações *P2P* utilizam diversos portos de comunicação, revela-se difícil capturar os pacotes com base nos actuais filtros, sem que se assista a uma elevada degradação do desempenho. Não sendo exclusiva das aplicações *P2P*, a utilização de um elevado número de portos, também se verifica em sistemas *Voice over IP*. Estas aplicações, nas suas diversas comunicações, não utilizam sempre portas conhecidas *à priori*, pois por vezes fazem uso de protocolos em que no início da sessão negoceiam portos, o que dificulta substancialmente a utilização de filtros, por se revelar necessário conhecer todos os protocolos específicos das aplicações. Para além desta dificuldade, existem aplicações que são executadas pelo administrador que não podem ser monitorizadas em nível utilizador, designadamente no caso da utilização do carregamento dinâmico de bibliotecas instrumentadas, para efectuar a monitorização de aplicações.

No contexto de uma dissertação anterior [Far09], utilizou-se a biblioteca *LibPCap* para a captura do tráfego de rede com vista à monitorização das interacções entre processos distribuídos, sendo que um dos principais desafios solucionados consistiu no isolamento de pacotes pertencentes a um processo. Porém, a solução encontrada, não se revela de fácil integração em qualquer outra ferramenta, e acarreta elevada sobrecarga.

Nos actuais sistemas de monitorização de rede não existe suporte para a monitorização das interacções de rede dos processos. Para levar a cabo esta tarefa de monitorização é imprescindível que métodos alternativos de monitorização de processos sejam combinados com a monitorização genérica de rede. Esta combinação, quando possível, manifesta fraco desempenho e implica uma elevada especificidade na monitorização dos programas.

1.2 Principais contribuições

É objectivo desta dissertação investigar mecanismos, incluindo os internos ao núcleo, que permitam incorporar a filtragem com base no identificador do processo, assim como a sua possível integração nas funcionalidades do *PCap*, e proceder igualmente à

avaliação da sobrecarga introduzida e respectiva optimização.

A abordagem efectuada beneficia dos mecanismos de instrumentação do núcleo para obter as interacções das aplicações com as interfaces de rede, criando uma extensão ao sistema de filtragem de pacotes do *Linux*, que apenas devolve à monitorização os pacotes referentes à aplicação instrumentada, reduzindo substancialmente o seu número, de modo a transferir apenas os dados relevantes para o monitor, evitando trocas de contexto e cópias de dados desnecessárias.

A possibilidade de monitorizar apenas o fluxo de rede de um determinado programa, poderá permitir que filtros construídos até ao momento possam ser simplificados. Para além desta simplificação, a monitorização do fluxo de rede de uma aplicação, permite a observação dos dados, sem necessitar de um sistema que previamente identifique os protocolos de mais alto nível, aplicados sobre a rede. A existência de um sistema de captura de tráfego de rede genérico, torna-se benéfico para a análise de protocolos, na medida em que nem sempre se tem acesso às especificações destes, presentes nas aplicações. Com este componente, o desempenho na obtenção dos dados relevantes poderá ser incrementado, mitigando anteriores problemas constatados (tais como trocas de contexto, cópias de dados, entre outros), entre o núcleo de sistema de operação e as ferramentas de análise de tráfego. Merece igualmente referência, a possibilidade de análise dos fluxos do processo sem necessidade de instrumentar o código da aplicação, uma vez que a instrumentação é efectuada no núcleo. À possibilidade anteriormente referida acresce a de monitorizar o fluxo de diferentes máquinas virtuais dentro de um sistema, sempre que estas sejam implementadas utilizando processos, permitindo individualizar e capturar o tráfego de cada uma. Esta funcionalidade pode ser particularmente interessante em centros de dados, visto ser possível efectuar a análise ao tráfego, sem necessidade de proceder à paragem da máquina.

Foi submetido e aceite um artigo para o *Inforum - 2011 Simpósio de Informática*[1], contendo uma descrição sucinta do trabalho do trabalho realizado nesta dissertação, bem como os resultados obtidos através das validações.[MD11]

1.3 Organização do Documento

Os restantes capítulos do documento, encontram-se assim distribuídos e estruturados:

- **Capítulo 2 - Monitorização de processos -** Introdução à monitorização de processos, evidenciando a monitorização de rede. Apresentação do estado da arte da

[1] http://inforum.org.pt/INForum2011

monitorização do núcleo do *Linux* e trabalhos relacionados com a monitorização de processos.

- **Capítulo 3 - Sistema Proposto -** Estrutura de comunicação e monitorização de rede do *Linux*, bem como os seus constituintes. Apresentação da estrutura do *MRoP* e da sua interligação com a estrutura de rede do *Linux*.
- **Capítulo 4 - Implementação do sistema proposto -** Implementação do *MRoP* e discussão da implementação.
- **Capítulo 5 - Avaliação -** Avaliação funcional e de desempenho do *MRoP* e dos seus componentes. Análise do desempenho do sistema de monitorização utilizado (*KProbes*).
- **Capítulo 6 - Conclusões e Trabalho Futuro -** Apresentação das conclusões referentes à avaliação efectuada ao *MRoP* e propostas para a sua evolução.

2

Monitorização de processos

2.1 Visão geral

O recurso à monitorização do comportamento das aplicações, permite-nos obter dados relevantes sobre os recursos realmente utilizados, de modo a proporcionar-nos um conhecimento mais profundo do seu comportamento em execuções mas permitem também analisar o plano de execução, os métodos mais executados, e detectar situações de eventos interessantes. As informações recolhidas podem servir para analisarmos o desempenho ou correcção da aplicação sequencial ou distribuida, porquanto ao conseguir-se obter dados da utilização do *cpu*, da memória, dos dispositivos de *IO*, interacções, etc, é possível compreender o comportamento dinâmico das aplicações. Daí ser comum a utilização da monitorização como auxiliar na avaliação e depuração de programas conseguindo um bom compromisso entre a qualidade dos dados recolhidos e a perturbação das aplicações. [Dua05].

2.1.1 Obtenção de informação

Os dados relativos ao comportamento das aplicações, obtidos de diferentes fontes, são normalmente coligidos e posteriormente analisados por ferramentas especializadas. Existem diferentes formas de coligir, visualizar e até interactuar com os monitores, cada uma com as suas especificidades e capacidades próprias. Algumas são desenvolvidas para objectivos específicos. Tendo em vista o conhecimento geral das capacidades de cada uma, podemos analisar estes sistemas de variados pontos de vista.

Se considerarmos por exemplo quanto à análise e visualização face ao instante da execução da aplicação alvo, podemos dividir os sistemas em:

Online - Enquanto decorre a monitorização da aplicação é possível observar os dados que são recolhidos pelo monitor. Como os eventos estão a ser recolhidos e visualizados em simultâneo, apenas podemos observar a história até ao momento, mas temos uma baixa latência entre os acontecimentos e a sua observação.

Offline ou *Post-Mortem* - A história do programa é analisada após este se ter completado daí a designação *Post-Mortem*. Este método permite-nos analisar integralmente a sua história e correlacioná-la. Permite análises mais completas e computacionalmente mais exigentes.

Se a monitorização necessitar de interactividade do utilizador, é possível defini-la de duas formas:

Activa - Por iniciativa explícita do utilizador, é possível inquirir o sistema de monitorização sobre o estado da computação ou mesmo alterá-la. Este método, por vezes descrito como *computacional steering*, é a forma com maior interactividade, uma vez que permite ir analisando e modificando os parâmetros da monitorização ou mesma da aplicação.

Passiva - Esta forma de monitorização é especialmente utilizada em ambientes onde é relevante a obtenção da totalidade dos dados e apenas no final nos debruçarmos sobre a sua análise. Esta forma é designada de passiva, por o utilizador não ter intervenção na forma como os dados estão a ser obtidos, o que reduz a perturbação do sistema. Quanto muito a sua acção acontece antes da execução, para configuração da informação a recolher.

A própria instrumentação da aplicação pode ser de dois tipos: a estática e a dinâmica, cada uma com as suas características próprias:

Estática - Na instrumentação estática o código instrumentado é definido em tempo de compilação ou utilizando bibliotecas próprias para o efeito, como a utilização da função *assert*, que define os pontos a serem monitorizados. Durante a execução não podem ser adicionados ou removidos pontos de análise.

Dinâmica - Em contraste com a instrumentação estática está a dinâmica. É mais complexa e permite a inserção e remoção dos pontos a serem monitorizados. O dinamismo verifica-se pela ausência do ciclo $introduzirponto \rightarrow compilarprograma \rightarrow executar \rightarrow removerponto$. A utilização de pontos de instrumentação dinâmica, pode ajudar a reduzir o grau de perturbação, uma vez que apenas são definidos os que se desejam observar. Tal pode ser efectuado no início ou durante a sua execução, criando, alterando, destruindo os pontos de observação sobre os recursos monitorizados.

A recolha de informação, é uma das componentes mais sensíveis relativamente ao grau de perturbação da monitorização. Em geral os pontos de instrumentação levam a que os dados sejam armazenados num *buffer* em memória. Caso exista algum evento que indique que o *buffer* se encontra cheio ou por acção explícita do utilizador, tal é transferido para a interface com o utilizador ou armazena em memória persistente, para posterior análise.

Existe, claro a preocupação de que o sistema a ser monitorizado tenha um baixo grau de perturbação, pois pode levar à alteração dos resultados obtidos e mesmo a comportamentos diferentes da aplicação (especialmente perante execuções concorrentes). Daí que, diversas abordagens foram criadas, para reduzirem o impacto da monitorização num sistema em produção. Uma destas abordagens traduz-se na utilização de instruções especializadas, de que alguns processadores dispõem para *debug*, de modo a utilizar os recursos que mais se adequem à monitorização. Estes métodos, nem sempre são utilizados, devido à sua dependência da arquitectura, o que dificulta a sua portabilidade. Com vista a minimizar a perturbação, alguns sistemas de monitorização utilizam uma técnica de amostragem, que permite obter indicações sobre os estado da computação a cada intervalo de tempo. Esta técnica, em oposição à criação de um traço de execução, permite obter dados sobre os recursos apenas por amostra, limitando à partida a perturbação, enquanto que na criação de um traço de execução, é possível obter a totalidade dos eventos de forma a criar uma história completa, mas pode levar a uma grande sobrecarga do sistema perante uma elevada taxa de eventos.

No entanto, capturar estes dados oriundos da monitorização pode revelar-se insuficiente, se não dispusermos de uma ferramenta onde estes possam ser tratados, de modo a obtermos análises mais completas e relacionar com os detalhes de funcionamento da aplicação. Este não será o foco deste trabalho.

2.1.2 Monitorização de Rede

Em geral, as ferramentas de monitorização de rede, são baseadas na captura de pacote de forma passiva. Estas capturam os pacotes que fluem na rede, para posterior análise ao tráfego, que pode incidir sobre a largura de banda utilizada, principais protocolos, eventuais problemas de segurança, etc.

A monitorização das interacções dos processos com o exterior, pode ser efectuada de duas formas distintas: Utilizando bibliotecas instrumentadas, que obtêm os pacotes de dados quando estes chegam à aplicação ou recorrendo a mecanismos genéricos de monitorização de rede do *Linux*. Relativamente à primeira, é necessário conhecer o código da aplicação e instrumentá-lo, obtendo-se apenas os dados que chegam à aplicação. Este processo é específico a cada aplicação e pode ver o seu uso limitado por questões de segurança do sistema. No que à segunda forma se refere, a monitorização é efectuada de forma genérica, não sendo intrusiva para as aplicações, necessitando apenas de efectuar a monitorização do processo e da rede, separadamente.

Dinamismo das aplicações - As aplicações são dinâmicas quanto às interacções via rede (por exemplo criando e destruindo de canais de comunicação). A este dinamismo se devem algumas dificuldades com que nos deparamos, ao monitorizar as interacções dos vários processos das aplicações.

Relativamente à monitorização das interacções com o exterior, utilizando os mecanismos do sistema de operação, este dinamismo é um factor chave, pois existem dificuldades na forma de identificar os fluxos pertencentes a um processo face aos restantes, irrelevantes na nossa análise.

Formas de reduzir o volume de dados utilizando filtros - A utilização de filtros na captura do tráfego que circula na rede, é uma forma eficiente de apenas se obterem os dados relevantes, com vista à satisfação dos nossos objectivos e são particularmente importantes, quando o volume de dados que circula na rede é extremamente elevado. Tendo em vista a eficiência, estes filtros são implementados no núcleo do sistema de operação, baseando-se em regras simples, que podem ser combinadas para contemplar situações mais complexas.

Dificuldade de criação e alteração de filtros - Os filtros actualmente suportados para capturar pacotes no *Linux*, são definidos *a priori*, não existindo forma eficiente de os alterar dinamicamente, uma vez que a captura tem de ser interrompida a fim de ser criado um novo filtro, posteriormente aplicado, procedendo-se em seguida à retoma da captura dos pacotes, de acordo com as novas regras.

Filtros mais complexos e inteligentes - De forma a aumentar o desempenho da captura de pacotes, os filtros são aplicados o mais cedo possível, ou seja, logo que chegam à interface de rede. Face a esta situação, o tipo de filtros que se podem aplicar têm de ser simples, baseados apenas no conteúdo do pacote. Quando os filtros são demasiado complexos, os módulos do núcleo têm de passar grande parte da informação para as camadas superiores, de forma a puderem ser aplicados filtros mais elaborados, que analisam um nível mais abstracto da informação, presente no *payload* dos pacotes, ou a relacionam com outra informação, para obterem apenas os dados relevantes.

Técnicas para o aumento de desempenho Diferentes técnicas têm vindo a ser desenvolvidas para aumentar o desempenho da monitorização das interfaces de rede. Como já referido, o aumento da sobrecarga é muito penalizante, daí que esta deva ser mantida bastante reduzida, o que contribuirá para aumentar o desempenho da captura.

Hoje em dia a utilização de máquinas equipadas com processadores *multi-core* é uma realidade, daí que a sua utilização permita ultrapassar algumas dificuldades sentidas na captura de pacotes. Se for os processadores *multi-core* atenderem em parelelo as diversas interrupções, originadas pelo envio ou recepção de pacotes, o desempenho da rede é passível de ser aumentado. No entanto, revela-se díficil atingir este paralelismo para todas as interfaces de rede, pois nem todas estão preparadas para beneficiar de arquitecturas com múltiplos *cores*.

2.2 Sistemas de monitorização no núcleo do *Linux*

Como o anteriormente analisado, na secção 2.1.1 referente à monitorização, esta secção é dedicada à apresentação das diferentes ferramentas de monitorização do núcleo do *Linux*, sendo as mais recentemente utilizadas neste sistema de operação. A utilização dos sistemas de monitorização ao nível do núcleo, permitem efectuar a monitorização genérica dos processos, de forma não intrusiva para as aplicações. Ao efectuar a monitorização no núcleo, a sobrecarga imposta ao sistema é menor que em nível utilizador, por ser possível recolher apenas as informações relevantes, evitando filtrar os elementos não relevantes, para análise.

Neste documento constam apenas mecanismos e ferramentas de monitorização dinâmicas, pois apenas estas são as desejadas para a criação de uma componente de monitorização de rede orientada ao processo. De entre as ferramentas de monitorização dinâmica analisadas é possível agrupá-las em duas categorias: eventos pré-definidos

e instrumentação dinâmica. Na categoria instrumentação dinâmica existem dois sistemas mais relevantes: o *KProbes* e o *Linux Kernel Stace Tracer*, enquanto que para os eventos pré-definidos foram verificados o *Linux Trace Toolkit* e o *OProfile*. Estes quatro sistemas são apresentados em seguida considerando as suas categorias:

2.2.1 Eventos pré-definidos

Nesta categoria encaixam-se duas ferramentas o *Linux Trace ToolKit* e o *OProfile*. Em cada uma destas ferramentas a monitorização é efectuada sobre pontos previamente definidos, não permitindo a adição de novos pontos de monitorização. Cada um destas ferramentas analisa todo o sistema, sendo a filtragem extra dos eventos efectuada em nível utilizador. Como os eventos detectados através destas ferramentas geram um mapa de execução, a utilização de apenas alguns eventos pré-definidos não é limitativa de uma ferramenta.

2.2.1.1 Linux Trace ToolKit

O *Linux Trace ToolKit* é uma ferramenta um pouco mais antiga que o *KProbes*, constituída por quatro componentes: o *Kernel Patch*, o *Kernel Module* o *Trace Daemon* e o *Data Decoder*, foi substituída pelo *Linux Trace Toolkit New Generation* que se apresenta em seguida.

O *Linux Trace Toolkit New Generation*, e permite utilizar mecanismos como o *KProbes*, o *Tracepoints*[MR09] e *Linux Kernel Markers*[MR09] para efectuar as suas monitorizações. Os *Tracepoints* e Linux *Kernel Markers* fazem parte da instrumentação estática pertencente ao núcleo de sistema do *Linux*.

A transferência dos dados do núcleo de sistema para a aplicação, é efectuada utilizando o sistema de ficheiros virtual *RelayFs*.

O *Linux Trace Toolkit Viewer (LTTV)* é um projecto desenvolvido em paralelo com o *LTT* e *LTTng*, de forma a possibilitar uma análise visual dos dados recolhidos por estas aplicações. Esta ferramenta possibilita igualmente realizar um traço de execução, uma vez que os dados recolhidos contêm uma estampilha temporal, do momento em que foram obtidos.

2.2.1.2 OProfile

As ferramentas anteriormente referidas utilizam mecanismos para obter um traço de execução. Com o *OProfile*, em vez de a cada evento utilizar uma função para obter os dados, apenas a utiliza decorridos um certo número de eventos. O recurso a este critério permite ao *OProfile* ser menos perturbador, uma vez que nem sempre é necessário o

universo dos eventos que as ferramentas de monitorização capturam, mas tão somente uma amostra. Deste modo ao utilizar a amostragem, o *OProfile*, beneficia de um menor grau de perturbação no sistema[Wil].

2.2.2 Suporte à monitorização

Como suporte à monitorização destacam-se duas ferramentas: o *KProbes* e o *Linux Kernel Stace Tracer*. Em ambos é possível definir novos pontos, funções ou eventos a serem monitorizados, não ficando restrito a funcionalidades já existentes nestas ferramentas.

2.2.2.1 KProbes

O *KProbes* é um mecanismo de instrumentação dinâmica do núcleo do *Linux*. A *API* permite que aplicações, como o *DProbes* ou o *SystemTap* possam aceder às suas funcionalidades. Esta encontra-se na versão principal do núcleo do sistema *Linux* desde a versão 2.6.9, o que indica tratar-se de uma ferramenta bastante estável[Pan04, KPr].

Existem três tipos de instrumentação que podem ser efectuados: *KProbe*, *JProbe* e *KRetProbe*. Cada um destes três tipos é especifico de uma determinada funcionalidade.

- **KProbe -**

  Utilizando um *KProbe* pode-se analisar a chamada de uma função ou de uma instrução. Para analisar apenas uma instrução, tem de ser indicado o *offset* relativamente à distância ao inicio da função instrumentada. A indicação da função a instrumentar, pode ser definida recorrendo ao seu nome ou ao endereço de memória. Múltiplos *KProbe* podem ser definidos para uma mesma função ou instrução.

- **JProbe -**

  Um *JProbe* destina-se a analisar os parâmetros, da função a instrumentar. Quando é declarado um *JProbe* a função de *handler* tem de conter os mesmos tipos de argumentos que a função que se destina a ser instrumentada, de forma a serem monitorizadas.

- **KRetProbe -**

  Este tipo de análise tem como objectivo obter o valor de retorno da função que se pretende analisar. Para efectuar esta operação é utilizada a técnica do trampolim [HMC94], uma vez que uma função pode terminar em pontos diversos.

Para se obterem informações sobre as funções ou instruções que estão a ser monitorizadas, o *KProbes* disponibiliza-as através de um ficheiro no *DebugFs*. Para utilizar

o *KProbes* é necessário criar um módulo para o núcleo do sistema de operação, com informações relativas às rotinas a instrumentar. Para além destas, o módulo tem de conter igualmente os *handlers* de análise, assim como outros parâmetros necessários à realização da instrumentação.

Quando o módulo é inserido no núcleo de sistema, o registo dos *handlers* das funções a serem instrumentadas é efectuado de forma atómica. A execução do *KProbes* não utiliza nenhum *mutex* ou outra forma de controlo de concorrência, apenas funciona com a preempção desligada. Dependendo do contexto, os *handlers* podem ser executados com as interrupções desligadas.

A instrumentação, processa-se substituindo por interrupções as instruções ou funções a serem analisadas, utilizando neste caso o *int 3*. Quando possível o *KProbes* utiliza instruções do processador especializadas no *debug* de forma a minimizar o grau de perturbação.

Apesar de ser possível instrumentar praticamente todo o núcleo do sistema de operação, existem algumas situações que podem requerer alguma perícia na forma como são tratadas. Quando o compilador realiza substituição de funções por código *inline*, as instruções que anteriormente estavam dentro da função passam a integrar o ponto onde foram substituídas, deixando de existir um ponto de entrada na função. Assim sendo, a função deixa de existir, impossibilitando a instrumentação destas funções.

O *DProbes* utiliza o *KProbes* como suporte para efectuar a instrumentação dinâmica do núcleo de sistema. Como a criação de pontos de análise é escrito em linguagem C ou *assembly*, foi desenvolvida uma linguagem de mais alto nível de forma a simplificar a utilização desta aplicação.

Após os dados terem sido obtidos, estes podem ser transferidos para um ficheiro, para o *daemon logging* do núcleo de sistema, ou para uma porta série. Existe igualmente a opção de interoperabilidade com o *Linux Trace Toolkit*[DPr].

O *SystemTap* é uma ferramenta que possibilita o desenvolvimento de módulos para o *KProbes*, utilizando uma linguagem específica, de forma a ser segura e fácil de trabalhar.

Uma vez que os dados gerados estão no espaço de endereçamento de memória do núcleo, e o programa que os analisará se situa no espaço de endereçamento do utilizador, é necessário efectuar uma transferência de dados de um para outro espaço. O *SystemTap* efectua esta transferência, recorrendo à utilização do sistema de ficheiros *ReplayFs*, onde é possível escrever de forma rápida, sem comprometer a segurança do sistema[DIH+07, Jon09].

Apesar de no pacote de aplicações desta ferramenta, existir um visualizador gráfico de dados, recolhidos pelo *SystemTap*, esta só consegue analisar os *TapSets* pré-definidos

no *SystemTap*. Para ultrapassar esta limitação, novas ferramentas estão a ser desenvolvidas. Uma destas ferramentas designada por *bootlimn*, está actualmente a ser desenvolvida no *Google Summer of Code*, com o objectivo de utilizar o *SystemTap* para recolher informações sobre o arranque do sistema, agregando estes dados no formato *XML*. Os dados recolhidos, permitem visualizações gráficas do processo de inicialização do sistema, bem como da utilização de disco e *CPU*.

Outro projecto conhecido como **Systemtap GUI**, que engloba o **System Tap Editor Plug-in** para a ferramenta Eclipse, é um ambiente onde podem ser analisados e visualizados os dados recolhidos pelo *SystemTap*.

O *SystemTap* actualmente está a ser desenvolvido por empresas de reconhecido mérito internacional, nomeadamente a *Red Hat*, a *IBM*, a *Hitachi* e a *Oracle*.

2.2.2.2 Linux Kernel State Tracer

O *Linux Kernel State Tracer(LKST)* obtém informações referentes ao núcleo de sistema, de forma a criar um traço de execução, e consegue capturar diferentes eventos como trocas de contexto, envio de sinais, alocação de memória, transmissão de pacotes, etc.

Actualmente, está a ser desenvolvido um subprojecto do *Linux Kernel Stace Tracer* designado por *Direct Jump Probe*. O *Direct Jump Probe* é uma optimização à utilização do *trap int3*, presente em alguns processadores, e que pode trabalhar em conjunto com o *KProbes*. Esta optimização pode ser verificada no relatório[**?**].

2.2.3 Comparação entre os diferentes sistemas de instrumentação

Tabela 2.1: Tabela Comparativa dos sistemas de instrumentação

Instrumentação	Amostragem / Traço	Análise de Parâmetros	Deamon
KProbes	Traço	Permite	Não necessita
LKST	Traço	Permite	Necessita
LTT	Traço	Não permite	Necessita
OProfile	Amostragem	Não permite	Necessita

A tabela 2.1 permite fazer uma comparação entre algumas das caracteristicas destes sistemas de instrumentação presentes no núcleo de sistema do *Linux*, segundo os seguintes critérios: metodologia da captura (por traço ou por amostragem), análise dos parâmetros das funções instrumentadas e a necessidade de ter um *deamon* para a recolha de dados provenientes do sistema de monitorização. Se a necessidade de ter um

deamon a executar de forma a coligir e organizar os dados, pode penalizar o desempenho do sistema, a possibilidade de analisar os parâmetros das funções instrumentadas é um ponto a favor dos sistemas *KProbes* e do *LKST*.

2.3 Transferência de dados

Quando se recorre a alguma fonte para a obtenção de dados, não é imprescindível que estes sejam requeridos no sítio onde são capturados. Assim, foram criados dois modos de transferência de dados: os internos e os externos. No intuíto de melhorar a partilha dos recursos externos, o controlo dá-se ao nível do núcleo do sistema, pelo que parte das comunicações dos utilizadores com dispositivos externos, é efectuada através do núcleo de sistema.

2.3.1 Interno ao sistema

Devido há necessidade de transferir informações entre diferentes *buffers* dentro do núcleo de sistema, foram desenvolvidas técnicas tendo sempre como referência a minimização da sobrecarga. De entre as diferentes técnicas de transferência interna de informação ao núcleo do sistema, merecem especial relevo:

MMAP - Esta técnica é implementada utilizando páginas de memória partilhadas, reduzindo as transferências e gastos de memória, proporciona a possibilidade de mapear um canal para memória potenciando assim, a partilha de dados não só entre diferentes processos como igualmente entre os processos e o núcleo, que necessitem de aceder aos dados do mesmo canal. A partilha de dados de *IO* é uma vertente que tem sido objecto de desenvolvimento, de forma a aumentar o desempenho, porquanto permite a partilha de dados entre um ou mais programas, e o núcleo de operação. Esta partilha evita a necessidade de copiar dados dos espaços de endereçamento do núcleo para o do utilizador e vice versa.

Zero Copy - Esta técnica de transferência de dados sem que existam cópias dos *buffers* pertencentes ao núcleo de sistema e ao espaço de utilizador, permite que os dados sejam partilhados por diferentes entidades, sem necessidade de criação de novos *buffers* e cópias de dados, uma vez que apenas são passadas as referências.

Ring Buffers - Esta técnica consiste num aproveitamento dos recursos já alocados, mas que se tornaram desnecessários, estando desta forma disponíveis para nova utilização. Os *Ring Buffers* são utilizados tendo presente a relação custo / benefício, isto

é, quando o peso da criação e destruição de elementos é comparativamente superior à reutilização dos recursos já alocados. É igualmente utilizada em situações de *"produtor-consumidor"*, ou seja, quando é necessário manter a ordem dos elementos.

2.3.2 Interface do sistema com os processos

Devido à necessidade de análise de estruturas e dados do núcleo de sistema, foram criados diferentes subsistemas com vista à sua satisfação. Um destes subsistemas é o *ProcFs*, que existe no núcleo de sistema do *Linux* desde as primeiras versões. Outros como o *DebugFs* ou o *RelayFs* são mais recentes e com novas abordagens.

De forma a obter informações relativas a estruturas dentro do núcleo de sistema, foram referênciados os seguintes sistemas de comunicação entre o utilizador e o núcleo, a saber:

ProcFs - Desenvolvido para obter informações relativas aos processos tem sido utilizado desde as primeiras versões do núcleo de sistema do *Linux*. Apesar de novos sistemas como o *SysFs* terem sido criados, continua a ser extensivamente utilizado pelas aplicações, pois este contém informações disponíveis para o nível utilizador, sobre estruturas e dados dos processos localizadas no núcleo. Esta é uma das *API's* de comunicação, entre o núcleo e as aplicações, mais utilizadas, não obstante ter crescido de forma desorganizada.

SysFs - Este sistema de ficheiros virtual foi desenvolvido para colmatar algumas deficiências encontradas no *ProcFs*. Estes problemas situam-se basicamente na forma desorganizada como a quantidade de informação disponibilizada está distribuída sobre o *ProcFs*. Este novo sistema de ficheiros virtual, evidencia restrições na disponibilização das informações, uma vez que apenas é possível visualizar e modificar um *KObject* por ficheiro. Esta limitação conduziu a que outros sistemas de ficheiros virtuais, nomeadamente o *DebugFs* e o *RelayFs*, fizessem a sua aparição com vista a colmatar esta situação.

DebugFs - Este sistema foi essencialmente criado para ultrapassar as dificuldades sentidas pelos programadores, na utilização de um sistema de ficheiros tão restritivo como o *SysFs*. Apesar de já existirem dois sistemas de ficheiros virtuais, o *ProcFs* e o *SysFs*, este tentou colmatar os problemas que se apresentavam, não obstante não se mostrar tão estruturado como o *SysFs*, tem melhor organização de dados que o *ProcFs*.

O sistema de instrumentação do núcleo de sistema *Linux*, *KProbes*, utiliza este sistema de ficheiros virtual, de forma a apresentar os pontos, onde existe instrumentação no núcleo de sistema.

RelayFs - Este sistema de ficheiros virtual foi desenvolvido tendo em mente a transferência de grandes quantidades de dados entre o núcleo de sistema e o espaço de utilizador. O *RelayFs* responde a esta exigência, fazendo uso de novas primitivas onde são mais reduzidas as zonas de controlo de concorrência[DIH+07, TZYW03].

NetLink - O *NetLink* utiliza uma método de comunicação baseado em *sockets*, para estabelecer a comunicação entre o núcleo de sistema e os processos, em nível utilizador. Tendo como abstracção os *sockets*, este sistema tem igualmente a possibilidade de enviar dados para múltiplos processos, devido à utilização das primitivas de envio colectivo (*Multicast*). É com base no *NetLink*, que a comunicação entre processos dentro da mesma máquina (*IPC*) é implementada. A comunicação com diferentes partes do subsistema de rede é efectuada recorrendo a este sistema de *sockets*.

2.3.3 Comparação entre sistemas de transferência de dados

Sistema	Estruturação de Dados	Volume de dados
ProcFs	Com	Reduzido
SysFs	Com	Reduzido
DebugFs	Com	Reduzido
RelayFs	Sem	Bastante Elevado
NetLink	Sem	Elevado

Tabela 2.2: Tabela Comparativa de transferência de dados entre processos e núcleo de sistema

Como se pode verificar através da tabela 2.2, que compara a estruturação dos dados de cada Sistema, com o volume de dados que é possível transferir, o *RelayFs* e o *NetLink* são os dois sistemas de comunicação, que não tendo estruturação fixa dos dados, conseguem transferir um volume superior de dados, comparativamente com os restantes, objecto da análise.

2.4 LibPcap e Linux Socket Filtering

A monitorização de rede existente em muitos dos sistemas tipo *Unix* e mesmo no *Windows* permite capturar os pacotes de rede logo que eles chegam ao controlador de rede. Assim a biblioteca *LibPcap*[Lib] permite às ferramentas fazerem uso deste mecanismo. Uma das principais características desta biblioteca, é a sua *API* de alto nível para a captura de pacotes, que é igual em todas as plataformas. Para filtrar os pacotes indesejados a *LibPcap*, utiliza filtros baseados no *BPF (Berkeley Packet Filtering)*, normalmente implementados no núcleo de sistema, de modo a que a monitorização se torne mais eficiente e menos intrusiva.

Em seguida é apresentada a arquitectura da biblioteca *Pcap* e o seu suporte no *Linux*, *LSF*.

2.4.1 Arquitectura

A arquitectura do *Pcap* divide-se em duas partes: a biblioteca a nível utilizador, e o módulo no núcleo de sistema composto por *sockets* e filtros que são utilizados na captura, como se constata na figura 2.1. Este sistema de captura, evita imediatamente a captura de pacotes irrelevantes para a captura e permite que aqueles que não são visíveis às aplicações, devido à *firewall* do *Linux*, possam ser capturados e analisados pelas ferramentas de monitorização de rede.

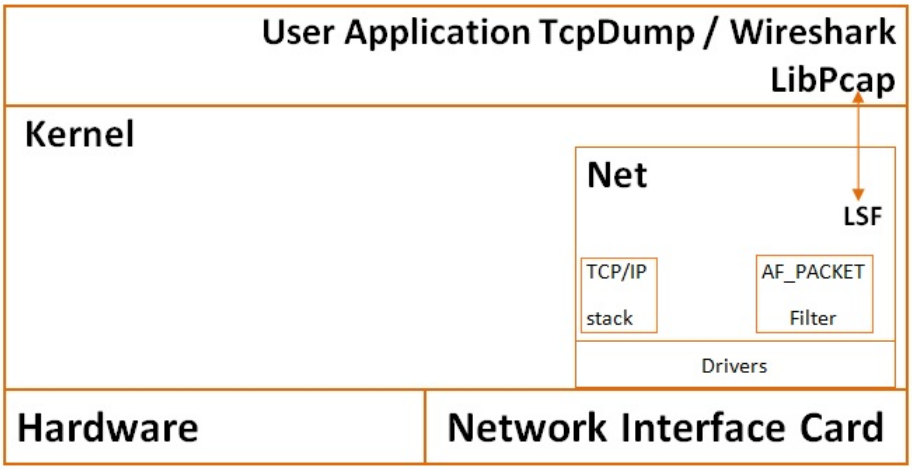

Figura 2.1: Arquitectura do LibPcap

A arquitectura de rede do núcleo de sistema *Linux*, utiliza *socket buffers*, que contêm dados referentes às diferentes camadas da *stack TCP/IP*. Esta estrutura que dispõe de toda a informação sobre o pacote, a receber pela aplicação, é igualmente utilizada

nos sistemas de monitorização em nível utilizador. A monitorização da rede é efectuada directamente no controlador da interface de rede, utilizando para isso um *socket AF_PACKET*.

A biblioteca dinâmica *LibPcap* é utilizada pela generalidade das ferramentas de captura de pacotes. Esta biblioteca, como é multi-plataforma, permite ao programador utilizá-la nos principais sistemas de operação, porque a *API* da *LibPCap* é transparante relativamente à implementação, esta sim, específica de cada plataforma. Para ser possível efectuar a captura de pacotes, o programa tem de especificar qual a interface de rede que quer monitorizar. Se necessitar de filtrar pacotes, especificará o filtro a utilizar, recorrendo às instruções presentes na *LibPCap*, que através da função *pcap_compile* o traduzirá e optimizará para o conjunto de instruções do *BPF*, sendo posteriormente aplicado no núcleo através da função *setfilter*. Se o filtro for demasiado complexo, não poderá ser aplicado no núcleo, sendo apenas possível aplicá-lo em nível utilizador, perdendo algum desempenho. Antes da introdução do novo filtro no canal, este tem de ser objecto de drenagem, para receber apenas os pacotes do novo filtro. Após esta inicialização, o programa poderá utilizar *Polling*, de forma a obter os pacotes e analisá-los.

O módulo *AF_PACKET* permite activar o modo promiscuo da interface a monitorizar, ou seja permite que sejam capturados pacotes que não são destinados à máquina utilizada. Na versão actual do núcleo de sistema do *Linux* (2.6.39), o *socket AF_PACKET* pode utilizar um sistema de partilha de *buffer* (*MMAP*), com o espaço de utilizador. Neste caso, cabe ao utilizador definir explicitamente a forma como pretende utilizar o *socket*. Para atingir este objectivo o utilizador tem de pedir ao núcleo, uma região de memória, sendo esta, posteriormente indicada como argumento de configuração do *socket*.

Como o referido anteriormente, o *AF_PACKET*, permite a comunicação directa com o controlador da interface de rede, possibilitando capturar os diversos pacotes ainda antes destes poderem ser filtrados pela *firewall*, presente no núcleo de sistema, neste caso o *netfilter*. Podem ser aplicados filtros aos *sockets* de modo a aumentar a eficiência na captura dos pacotes, excluindo aqueles que são irrelevantes para a monitorização.

2.4.1.1 Linux Socket Filtering

O *Linux Socket Filtering* é derivado do *BPF (Berkeley Packet Filtering)* que é o standard de *de facto* para a criação de filtros dentro do núcleo de sistema. Este sistema de captura e filtros permite aos os utilizadores com permissões de *Super User*, definir filtros e afectá-los aos *sockets*. Os filtros definidos são descritos numa linguagem simples, de forma a efectuarem com rapidez a selecção dos pacotes a capturar, rejeitando todos

os outros. A linguagem utilizada para a criação dos filtros é traduzida para o *Instruction Set*, definido no *BPF*. Este *Instruction Set* utiliza operadores lógicos de forma a combinar as regras definidas nos filtros, criando-se apenas um filtro a ser aplicado aos pacotes. [MJ92].

2.4.2 Limitações e optimizações

De modo geral, para a avaliar o desempenho da *LibPcap*, é frequente utilizar-se uma métrica que nos permita determinar a percentagem de pacotes que é possível capturar, sem que se verifiquem perdas. Esta métrica é calculada sabendo o número de pacotes que atravessam um determinado *router ou switch*, e compará-la com o número de pacotes capturados na interface. Uma variante desta medida de desempenho é combiná-la com o máximo número de regras que é possível aplicar, sem que se assista à perda de pacotes. É particularmente importante conhecer o número máximo de pacotes capturados ou regras aplicáveis aos filtros, afim de determinar a velocidade máxima expectável de captura. Quanto mais eficiente for a filtragem, menor o número de pacotes transferidos para as aplicações, o que se traduz numa menor sobrecarga para o sistema, aumentando o seu desempenho.

Captura de pacotes com *TimeStamp* Um dos pontos que pode influenciar negativamente o desempenho da obtenção de pacotes, é a necessidade de colocar nestes, uma estampilha temporal, com o tempo da sua chegada ao sistema. O *LibPcap* pode obter estes dados de duas formas distintas, dependo da interface de rede que esteja a ser utilizada. Se a interface de rede fornecer a estampilha temporal associada ao pacote, o *LibPcap* pode utilizar este valor, caso contrário será necessário obter a estampilha temporal da chegada do pacote ao sistema. A forma de obtenção da estampilha pela *LibPcap*, processa-se através da função *gettimeofday*, o que incorre numa maior sobrecarga do sistema face à solução anterior.

Apesar da *API* da *LibPcap* ser igual em diferentes plataformas (*Windows*, *Linux*, *FreeBSD*, etc), o desempenho desta pode variar. Quando exista pouco tráfego de rede estas diferenças são pouco notórias, mas logo que este se intensifica, estas acentuam-se bastante, como demonstra o estudo *Improving Passive Packet Capture: Beyond Device Polling* [Der04].

Diversos esforços no sentido de aumentar o desempenho da captura de pacotes têm sido efectuados. Relativamente ao *software*, são conhecidos esforços na utilização da técnica de *mmap*, de forma a reduzir o número de cópias de dados entre as aplicações e o núcleo de sistema. Igualmente no *hardware*, tem-se assistido a uma evolução

no sentido de reduzir as interrupções efectuadas ao *cpu*, adicionando nas interfaces de rede processadores dedicados às funcionalidades presentes no núcleo, de modo a libertá-lo da execução destas tarefas.

NAPI - New API - Foi criada uma nova *API* (*Application Program Interface*) de atendimento de interrupções do processador, oriundos das interfaces de rede, com o objectivo de aumentar o desempenho da utilização de rede de elevado débito. Esta nova *API*, permite desligar a atenção do processador a novas interrupções, oriundas da interface de rede, durante um certo período de tempo.

Este tempo foi definido, de forma a que não se verifiquem situações de elevada latência na chegada dos pacotes às aplicações. A forma anterior de recepção de pacotes, era efectuada de modo a atender uma interrupção por cada pacote que chegava à interface de rede, gerando assim demasiada sobrecarga no sistema. Actualmente a arquitectura de rede, está balanceada de forma a mudar o modo de atendimento de interrupções para NAPI, caso existam demasiadas interrupções por unidade de tempo. [Adm09].

Utilizando esta nova *API*, é possível explorar um escalonamento mais eficiente das interrupções, em situações de intensa actividade da interface de rede e do processador. O *NAPI* apenas pode ser aplicado nos caso em que os controladores das interfaces de rede estejam preparados para utilizar alguma forma de mitigação da interrupção, caso contrário esta *API* não é aplicada, resultando em interrupções por cada pacote, enviado ou recebido.

A utilização desta nova *API* permite diminuir o número de trocas de contexto entre o controlador da interface e o núcleo de sistema. Sempre que o cpu atende uma interrupção de rede, obtem um número maior de dados, que combinado com um sistema de memória partilhada aumenta substancialmente o desempenho.

PACKET_MMAP Com base na técnica *MMAP*, foi criado o *PACKET_MMAP*, disponível a partir da versão 1.0.0 da biblioteca do *LibPcap*. Este módulo permite algumas melhorias em termos de desempenho, visto que foi reduzido o número de cópias efectuadas e de trocas de contexto, face à anterior versão 0.9.8 da biblioteca *LibPcap*.

Para além destas modificações no *Packet_MMAP*, o núcleo de sistema de operação *Linux* na sua versão 2.6, passou a contar com a nova *API* de rede (*NAPI*).

Se as interfaces de rede suportarem um mecanismo de mitigação de interrupções, é possível obter melhores resultados, conforme [Der04].

PF_RING Este é um novo módulo para o núcleo de sistema, criado com base em duas técnicas *mmap* e *ring_buffers* anteriormente descritas.

Este módulo difere na abordagem utilizada no *Packet_MMAP*, porquanto nesta a memória é mapeada entre a ferramenta e o controlador da interface, enquanto que no *Packet_MMAP* a memória é mapeada entre a ferramenta e um *buffer* externo ao controlador da interface, mas interno ao núcleo de sistema. Esta abordagem permite que os dados fiquem disponíveis para a aplicação directamente, verificando-se a inexistência de cópia dos dados do *buffer* do controlador, para o *buffer* partilhado entre a ferramenta e o núcleo[PFR].

PF_RING com DNA (Direct NIC Access) Baseando-se na técnica anteriormente descrita de utilizar um *buffer* partilhado entre a ferramenta de monitorização e o controlador, assiste-se a uma evolução desta técnica, ao permitir que a interface de rede partilhe um *buffer* com a ferramenta de monitorização, possibilitando que os pacotes passem directamente para esta.[Int]. Esta partilha é efectuada utilizando *mmap*, *ring_buffers* e *DMA*. Para se utilizar esta técnica é necessário que a interface de rede, permita a utilização de memória partilhada e *DMA*.

Sistemas *multi-core* e multi-processador Com o aparecimento de sistemas *multi-core* e *multi-processador* a que a generalidade do público tem acesso, a paralelização de código ou a forma de tirar partido destas arquitecturas, que permitem um melhor aproveitamento dos recursos, assumem uma particular importância. De forma a tirar partido das arquitecturas *multi-core*, é necessário que o controlador e interfaces de rede, os *buffers*, e os controladores de *DMA* (*Direct Memory Access*), sejam modificados de forma a conhecerem esta estrutura. É pois, determinante um esforço conjunto envolvendo todas estas componentes, com vista à obtenção do máximo rendimento destas arquitecturas[Der10].

2.5 Captura de tráfego de um processo especifico

A captura do tráfego respeitante a um processo genérico, foi alvo de estudo em [LML09] ou e em [Far09], esta última objecto de uma dissertação de mestrado.

No primeiro trabalho, foi desenvolvido um sistema de captura de pacotes de um determinado processo, utilizando um módulo no núcleo de sistema que intercepta e captura os pacotes do processo.

Este sistema é constituido por três componentes essenciais, uma dentro do núcleo de sistema e duas em nível utilizador. Na realização do trabalho, foi utilizada a ferramenta *KProbes* para a monitorização de algumas funções do núcleo de sistema *Linux*,

de forma a conhecer quais os portos que vão ser utilizados por uma determinada aplicação. Logo que obtida, a informação é enviada para um processo em nível utilizador que tem o registo de todos os portos que estão a ser utilizados pela aplicação objecto de monitorização. Se esse porto não estiver a ser monitorizado, essa informação é passada a outro processo que utiliza a biblioteca *LibPcap* de forma a capturar o tráfego existente nesse porto.

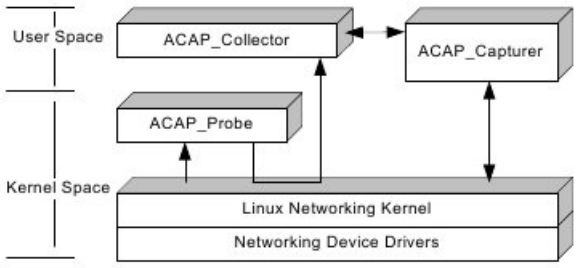

Figura 2.2: Arquitectura da monitorização de tráfego

Analisando a arquitectura de monitorização de tráfego representada na figura 2.2, o ACAP_Collector e o ACAP_Capturer, estão em nível utilizador, de onde se infere que o desempenho desta ferramenta é passível de ser aumentado, caso esta seja completamente implementada dentro do núcleo de sistema de operação.

Relativamente ao segundo trabalho, foram implementadas duas abordagens: uma com monitorização da aplicação e outra através de informações pertencentes ao núcleo do sistema de operação. No que à primeira diz respeito, a monitorização efectuada processou-se através da interceptação das chamadas à bibiloteca *LibC*, para a utilização de *sockets*, criando uma bibiloteca partilhada com a mesma sintaxe das chamadas que são utilizadas. A opção pela utilização deste método, implica definir a variável de ambiente *LD_PRELOAD*, de forma a operar esta intercepção. Como esta biblioteca está em nível utilizador, mostra-se necessário capturar todos os pacotes e apenas em nível utilizador visualizar o tráfego respeitante ao processo. Esta obrigatoriedade de capturar todos os pacotes pode constituir uma tarefa com elevado grau de perturbação do sistema.

No que à segunda abordagem se refere, o método de monitorização utilizado baseou-se na consulta efectuada de forma regular com intervalos reduzidos, dos dados referentes aos portos de comunicação, utilizados pela aplicação, que são exportados pelo núcleo através do *ProcFs*. Esta forma de monitorização consome demasiados recursos e não se mostra totalmente fiável, na medida em que quanto menor o intervalo de tempo

utilizado, maior é a perturbação apresentada pelo sistema.

O dinamismo das aplicações, nomeadamente das aplicações multimédia, deu origem a diversos estudos sobre a forma de monitorização de rede, que estas necessitam. Estas aplicações utilizam diversos fluxos de dados, designadamente de transmissão e de recepção. Em geral, as aplicações multimédia, com base na *internet* utilizam uma metodologia cliente/servidor, onde o servidor aguarda pedidos do cliente num determinado porto. O cliente, conhecendo antecipadamente este porto, liga-se. A partir deste ponto, iniciam-se trocas de informações, que irão originar a troca de portos dinâmicos e posteriormente o processo de transmissão/recepção de dados multimédia.

As aplicações multimédia assentes na *internet* são apenas um exemplo de aplicações com diversos fluxos, em que as portas de comunicação entre as aplicações, são negociadas dinamicamente.

Como o referido em 7.1, a captura de pacotes é definida em filtros estáticos e para capturar este tipo de tráfego, é necessário modificar os filtros definidos, de modo a acompanhar o protocolo. Esta forma de captura mostra-se bastante ineficaz, o que motivou a que fossem estudadas algumas alternativas, tendo em vista a correcção desta situação. Os projectos *mmdump*[vdMChCS00], e *Swift*[WXW08] são dois destes casos estudados, que merecem especial relevância e que adiante se analisam:

MMDump - É uma ferramenta de monitorização de protocolos multimédia com suporte na rede. Esta aplicação tem como base o *tcpdump*, sendo a captura de pacotes efectuada através da utilização de filtros. Para determinar os portos a obter, é necessário analisar o conteúdo dos pacotes direccionados a portos específicos, e a partir destes é possível identificar os novos portos negociados dinamicamente pela aplicação, e proceder à alteração dos filtros a aplicar. Como a alteração, que é constituída pela cópia do novo filtro para o núcleo e verificação de segurança, de forma a validá-lo, é um processo demorado, é necessário reduzir este tempo, com o objectivo de ter uma aplicação que consiga minimizar o grau de perturbação no sistema.

Na alteração do filtro, foi verificado que existe um certo padrão, que consiste em pré-estabelecer uma parte comum a ser adicionada ao filtro, e apenas alterar a parte referente aos portos.

Esta forma de monitorização é muito específica, pelo que é imprescindível todo o protocolo interno de comunicação. Assim, para cada novo protocolo a monitorizar, é necessário acrescentar um novo módulo com a interpretação desse protocolo.

Swift - É uma ferramenta de criação de filtros, cujo principal objectivo é a melhoria do desempenho da utilização destes, na captura do tráfego de rede. Nesta ferramenta, foi avaliado o tempo de alteração dos filtros, utilizando o *Linux Socket Filtering*, pois

estes são os filtros de referência no sistema de operação *Linux*. A aplicação destes filtros a partir da biblioteca *LibPcap* compreende três fases: cópia do filtro definido em nível utilizador para o núcleo de sistema, verificação de segurança, e aplicação do filtro. De forma a diminuir a latência de actualização dos filtros, foi criada uma especificação de modo a que estes não necessitassem de ser analisados relativamente à segurança, visto que a própria linguagem garante as propriedades de segurança necessárias. Com este novo *instruction set*, e sem necessidade de verificação, foi reduzida a latência de actualização dos filtros, o que conduziu a um aumento de desempenho na utilização de filtros dinâmicos.

3

Sistema Proposto

De modo a conceber a arquitectura do sistema de Monitorização de Rede orientado ao Processo (*MRoP*), foi necessário conhecer, como os processos em nível utilizador interagem com o exterior, via rede. Uma vez conhecida a arquitectura de rede do *Linux*, tornou-se imprescindível compreender a forma, como se processa a comunicação e a monitorização de rede.

Assim, tendo em conta os factores anteriormente mencionados, neste capítulo irá ser apresentado funcionamento dos processos e a sua estruturação, assim como, a constituição do sistema de rede do *Linux*, desde o momento do envio dos dados pelo processo até que estes cheguem à interface de rede, tendo em vista desenhar o sistema *MRoP*. Considerando a abrangência desta visão, apenas serão focados os componentes essenciais da arquitectura, nomeadamente o descritor de processo, as familias de *sockets*, a *firewall*, o sistema de escalonamento de rede e a monitorização de rede.

Após esta visão geral dos processos e do sistema de rede do *Linux*, será apresentada a arquitectura proposta para o sistema de Monitorização de Rede orientado ao Processo (*MRoP*).

3.1 Estrutura de um processo

Um processo é uma instânciação de um programa em execução, sendo igualmente uma unidade de escalonamento de execução no sistema operativo (*task*). Este consome recursos (físicos e lógicos), partilhados com outros processos num sistema multiprogramado. O núcleo de sistema gere uma estrutura para cada processo (*struct_task*), onde estão identificados todos os recursos a este atribuídos. Esta estrutura contém apontadores para diversos recursos nomeadamente: zonas de memória requisitadas, canais abertos (ficheiros, *sockets*, entre outros), contabilização de utilização de *cpu*, apontadores para a sua árvore genealógica (pai, irmãos e filhos), etc.[BDK+97, BDKV02]

A árvore genealógica anteriormente indicada contém o apontador para o processo pai (o processo que efectuou um *fork*, o qual lhe deu origem), a lista de irmãos e dos descententes. O processo pai, tal como cada um dos elementos presentes na lista, são também estruturas do tipo *task_struct*, possibilitando, desta forma, a navegação ao longo da árvore genealógica do processo.

Um dos recursos partilhados pelos diferentes processos é o acesso à rede, ou seja, o recurso que permite a comunicação com o exterior. De modo a que as comunicações tenham lugar, é necessário a alocação de canais de comunicação, cuja criação, utilização e destruição é efectuada pela *API* definida no sistema de operação. Quando um processo cria um canal, o núcleo devolve-lhe um identificador, permitindo ao processo referenciar o canal dentro do núcleo e, deste modo, efectivar o seu controlo e realizar futuras comunicações. No núcleo, este identificador é gerido na estrutura que identifica os canais abertos, pelo processo.

Para reconhecer qual o tipo do canal aberto, existem algumas funções específicas que o analisam e lhe permitem efectuar operações. Assim, ao manipular um ficheiro, existe a possibilidade de avançar e recuar, tendo como referência um determinado ponto. Todavia, esta situação deixa de fazer sentido, quando se pretende controlar um *socket* ou um *pipe*, na medida em que as comunicações nestes são destruídas quando consumidas (situação essa, que não se verifica aquando da manipulação de um ficheiro).

Apesar destas informações estarem disponíveis no descritor de processo, não existe suporte neste descritor de modo a obter as interacções do processo com as interfaces de rede. Desta forma, apenas através da instrumentação das chamadas ao sistema, é possível obter as interacções anteriormente referidas. Assim quando um processo comunica com a rede, executa uma chamada ao sistema e por intermédio da instrumentação destas, é possível obter algumas informações relevantes que, combinadas

com os dados do descritor do processo, permite que se obtenha as informações relativas aos portos, endereços e protocolos utilizados na comunicação com a rede. Com estes dados (protocolos, portos e endereços) é possível filtrar e capturar eficientemente os dados, através da extensão efectuada ao *LSF*.

3.2 Arquitectura de rede em *Linux*

Os processos necessitam de comunicar para obter dados, de modo a completar as suas execuções. Estas comunicações podem ser efectuadas tanto internamente, como externamente ao sistema. Em geral, as comunicações externas processam-se através de uma interface de rede, sendo necessário, para tal, criar canais de comunicação. A(s) interface(s) de rede são recursos do sistema, as quais são partilhadas pelos diversos processos. Esta partilha é realizada pelo núcleo, pois apenas este é capaz de a efectuar correctamente.

A chamada ao sistema *socket*, é efectuada para a criação de um canal de comunicação e este varia em função dos parâmetros: *família*, t*tipo* e *protocolo*. Existe um agrupamento em família de endereços (*Address Family*), consoante se destina à comunicação remota ou local. Em termos gerais, as comunicações locais podem incidir na comunicação entre processos (*AF_UNIX*), ou entre estes e o núcleo (*AF_NETLINK*), enquanto as comunicações remotas podem ter lugar sobre protocolos da Internet, dispondo igualmente de uma família própria para o efeito (*AF_INET*). Estes canais possuem um conjunto de funções (*API POSIX*) bem definidas, para utilização e controlo das operações, permitindo estruturar a comunicação de um modo eficiente.

Hoje em dia, os administradores de sistemas necessitam filtrar o fluxo de informação que circula nas suas redes. Para tal, tem-se assistido ao desenvolvimento de sistemas de filtragem de comunicações, conhecidas por *firewalls*. Sendo o *Linux* um sistema de operação frequentemente utilizado em ambientes de servidores, tornou-se essencial a aplicação deste tipo controlo. Assim, a solução encontrada resultou do desenvolvido de uma *firewall* (a qual foi denominada por *NetFilter*) capaz de filtrar o fluxo de dados que circulam na rede, entre o sistema e a periferia.

Para além da existência da filtragem do fluxo de dados, o sistema de operação *Linux* é constituido por um sistema de controlo do escalonamento do fluxo de tráfego, (*Traffic Control*), que permite ao administrador indicar quais os fluxos de dados prioritários ou que necessitam de determinada largura de banda, possibilitando-lhe efectuar uma reserva antecipada da mesma.

Adiante apresentar-se-ão diferentes constituintes da estrutura de rede do *Linux*, iniciando-se pelas famílias de *sockets* e terminando nos controladores de interfaces de

rede.

Por último será apresentado o sistema de monitorização genérica de rede do *Linux*.

3.2.1 Sockets e as suas famílias

Como o anteriormente referido na secção 3.2, um processo para comunicar via rede, tem de criar um canal. Estes são criados para efectuar comunicações locais ou remotas e, embora as acções realizadas sejam relativamente comuns, o meio de transmissão é diferenciado, assim como os canais e as formas utilizadas.

No núcleo do sistema, encontram-se implementadas diferentes famílias de endereços tal como apresentado na figura 3.1. No entanto desse conjunto destancam-se as famílias de endereços *UNIX*, *NETLINK*, *INET* e *PACKET*.[Ben05, WR05]

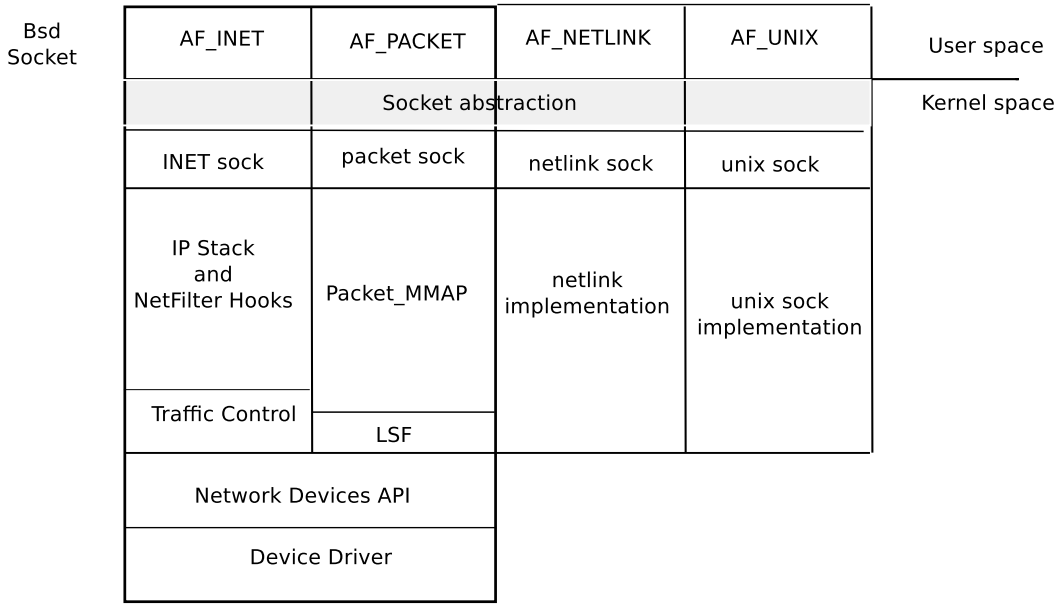

Figura 3.1: Arquitectura parcial de sockets do Linux

3.2.1.1 AF_UNIX

A *AF_UNIX* consiste numa família de *sockets* utilizada para efectuar a comunicação entre processos dentro da mesma máquina. Para tal, é um dos sistemas de *Inter Process Communication (IPC)*, utilizado em sistemas *Unix*, ao qual permite utilizar um ficheiro, contido no sistema de ficheiros, como um canal de comunicações.

3.2.1.2 AF_NETLINK

A família *NETLINK* é utilizada pelos processos, em nível utilizador, para comunicar com o núcleo. É possível efectuar comunicações ponto-a-ponto ou multi-ponto, isto é, é possível que um ou mais processos comuniquem com o núcleo, designadamente o *netfilter*, como o sistema de encaminhamento de pacotes de rede, como o sistema de configuração de interfaces de rede, etc., através de um *socket* nele criado. Cabe aos processos, em nível utilizador, conectarem-se ao *socket* definido no núcleo, o qual não é completamente passivo, na medida em que este pode iniciar comunicações assíncronas com os processos.

Embora as interfaces de rede sejam configuradas através do programa *ifconfig* em nível utilizador, o qual utiliza *ioctls* para efectuar essa configuração, uma outra ferramenta foi desenvolvida (*ethtool*), que tira partido de *socket netlink* para efectuar estas configurações, dado que as *ioctls* não permitem especificar correctamente os parâmetros (contrariamente ao que sucede nos *socket netlink*).

3.2.1.3 AF_INET

A *AF_INET* representa a família de protocolos utilizada na comunicação através da *Internet* e que faz uso do protocolo *IP versão 4*.

A estruturação por camadas permite que a implementação de protocolos seja efectuada de forma simples e rápida, sendo que ao utilizar as camadas inferiores como suporte para as superiores, aumenta a abstracção e complexidade dos protocolos.[SV08] Existem vários protocolos de nível transporte sobre *IPv4*, sendo os mais utilizados o *TCP* e *UDP*, como evidenciado nas figuras 3.2 e 3.3. A criação de canais sobre o nível de transporte é efectuado utilizando a chamada ao sistema *socket*, indicando para o parâmetro tipo, os valores *SOCK_STREAM* ou *SOCK_DGRAM* para os protocolos *TCP* e *UDP*, respectivamente.

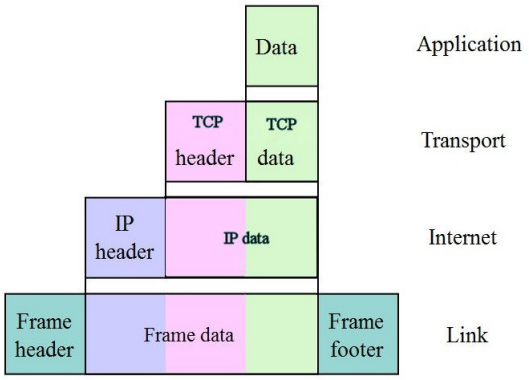

Figura 3.2: Protocolo TCP

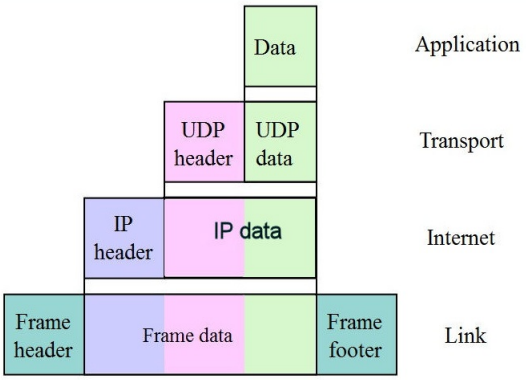

Figura 3.3: Protocolo UDP

O *TCP* é utilizado para comunicações com controlo de fluxo de dados, que se adaptam ao canal existente, tolerando a perda momentânea de pacotes, e procedendo à sua retransmissão logo que possível. O controlo das retransmissões é efectuado através do envio de pacotes (*acknowledges*), que informam o transmissor, dos que foram correctamente recebidos, permitindo-lhe reenviar os que se encontram em falta.

O *UDP* é um protocolo mais leve que o *TCP*, pois não utiliza o controlo anteriormente referido. Para tal, este protocolo é habitualmente utilizado quando se está perante a possibilidade de perda de pacotes durante a transmissão ou quando o controlo da retransmissão destes, é efectuado em camadas superiores.

Para além destes protocolos anteriormente mencionados, existe ainda um protocolo denominado por *SOCK_RAW*. Este protocolo permite o acesso directo ao nível rede, através da criação de um canal que comunica directamente com o nível rede da pilha de protocolos *TCP/IP*, permitindo executar protocolos de comunicação em nível utilizador. Para utilizar *sockets* através do modo *RAW*, é necessário que o utilizador tenha permissões de *super user*, uma vez que neste modo, é permitido ao utilizador específicar parte dos cabeçalhos dos pacotes a enviar, sem que o núcleo os valide. Contudo, a não validação, pode proporcionar que pacotes maliciosos, isto é, com falhas deliberadas, possam ser introduzidos na rede.

3.2.1.4 AF_PACKET

A família *AF_PACKET* comunica directamente com o controlador da interface de rede, possibilitando assim efectuar a monitorização de rede. A utilização destes *sockets* requer uma autorização de *CAP_NET_RAW*, apenas disponível ao *super user*. Tal verificação previne que utilizadores não autorizados possam monitorizar ou injectar pacotes na rede, influenciando o seu comportamento. Dado que dispõe da possibilidade de enviar pacotes directamente para o controlador de rede, uma das suas utilizações

é a implementação de protocolos de rede em nível utilizador. Os canais da família *AF_PACKET* são frequentemente utilizados em sistemas de detecção de intrusos, uma vez que permitem analisar os pacotes e detectar falhas na sua formatação.

Como a obtenção dos pacotes por estes canais, é efectuada antes da recepção destes pelos processos, existindo a possibilidade de o *NetFilter* bloquear a recepção de alguns pacotes, impedindo as aplicações de os receberem, não obstante do canal já os ter obtido para análise.

Como todos os pacotes recebidos pela(s) interface(s) de rede são obtidos por este canal, é necessário efectuar uma cópia destes e fornecê-los à aplicação em nível utilizador. Esta monitorização implica um peso extra na computação, pelo que diversas técnicas (designadamente *lazy cloning*, *mmap*, etc.) têm sido desenvolvidas no sentido de minimizá-lo. A utilização de *lazy cloning*, destina-se a apenas efectuar a cópia, caso esta seja absolutamente necessária, permitindo reduzir a perda de eficiência do sistema, aquando da monitorização dos pacotes de rede.

3.2.2 NetFilter

Este sistema, está implementado no núcleo do sistema de operação do *Linux*, o qual controla o fluxo de dados dos processos de e para as interfaces de rede. O *NetFilter* está presente na pilha de protocolos *TCP/IP* em apenas cinco pontos (*PRE ROUTING*, *LOCAL IN*, *FORWARD LOCAL OUT* e *POST ROUTING*) onde, em cada um dos quais, existe uma lista de funções a ser executada sobre o pacote, que foi recebido ou transmitido. Após a finalização destas funções e dependendo do valor retornado, o pacote irá ou não, prosseguir para as restantes camadas, até atingir o *Traffic Control*(apresentado na subseccção 3.2.3) ou na aplicação, efectuando deste modo, o controlo sobre o fluxo de rede. O *NetFilter* pode ser controlado pelo *iptables*, uma ferramenta em nível utilizador, que através de regras, controla o acesso dos dados das comunicações de rede.

Um dos componentes do *netfilter* designado por *connection tracking*, cuja sua funcionalidade permite que se efectue o acompanhamento das ligações desde a sua iniciação até ao seu término. De modo a realizar este acompanhamento, este componente necessita de conhecer os diferentes protocolos utilizados. Do mesmo modo que o *nefilter* pode ser controlado por um programa em nível utilizador, esta componente *connection tracking*, pode igualmente ser controlada pelo administrador, através do *conntrack*.

Este componente, *connection tracking*, permite efectuar uma gestão mais eficiente e inteligente das conexões, sendo geralmente referido como *firewall* com estado. Por outro lado, é também utilizado para efectuar *NAT* (*Network Address Translation*) sobre as interfaces de rede.

Na transmissão de dados, quando estes passam no *netfilter*, existe a possibilidade

de identificá-los de forma a serem reconhecidos pelo *traffic control*, permitindo assim efectuar uma gestão conjunta e inteligente do tráfego.

3.2.3 Traffic Control

No núcleo, para além da existência do *NetFilter*, que controla o fluxo de dados na rede, existe o *Traffic Control* que efectua o escalonamento da transferência de dados para o exterior. Na rede, o escalonamento do tráfego é particularmente importante para os seus administradores e, dado que a largura de banda é um recurso limitado, a sua gestão rigorosa é criteriosamente observada.

A gestão é efectuada através de regras definidas em função da largura de banda, ritmo de envio, ou outros parâmetros. O tráfego baseado nas regras anteriormente referidas, é agrupado em classes, que podem ter diferentes níveis de prioridade. Para além desta classificação, é ainda possível definir os algoritmos de escalonamento a utilizar, de modo a potenciar a eficiência da rede.

Quando se verifica a iminência do envio de um pacote pela função *ip_output*, este é transmitido para o *Traffic Control*, onde o pacote é marcado com a *tag* da respectiva classe. Posteriormente, esse pacote irá ser adicionado ao *QDisk* correspondente. O *QDisk* é a estrutura de suporte à transmissão de dados efectuada pelo *Traffic Control*, onde cada elemento desta estrutura é um *sk_buff* (que será abordado em 3.2.4.1).

3.2.4 Interfaces de rede

A interface de rede é o dispositivo que efectua a transmissão e recepção de dados na rede. Do ponto de vista do *cpu*, a interface de rede apenas efectua pedidos de interrupção, sendo que o restante trabalho é realizado pelo controlador de rede, o qual efectua a ponte entre as funcionalidades de rede do núcleo e a interface de rede.

O núcleo mantém a informação sobre as interfaces de rede, inicializadas pelos seus controladores. Estas, independentemente de estarem ou não em utilização, constam de uma lista duplamente ligada de *net_devices*. Em cada posição desta lista existem informações referente a uma interface de rede (nomeadamente a informação sobre o seu endereço *IP*, o seu endereço *MAC*, etc.), bem como outras configurações.

No registo de um novo controlador de interface de rede definem-se as operações a executar em determinados eventos (tais como a recepção e/ou transmissão de frames, a remoção da interface, etc.). Estas operações são definidas através de uma interface comum (*struct netdev_ops*), que permite afectar a esta estrutura, funções presentes nos módulos dos controladores, de modo a serem efectuadas as acções necessárias à utilização da interface de rede.

3.2.4.1 Estrutura de um sk_buff

Os dados envolvidos no fluxo de comunicação e utilizados pelos controladores correspondem aos *socket buffers*, estruturas do tipo *sk_buff*. Esta estrutura contém diversos elementos, destacando-se os apontadores para a estrutura de rede do controlador, apontadores para secções do pacote (os diversos cabeçalhos pertencentes ao nível de rede e transporte), bem como a dimensão dos espaços alocados para estes e outros apontadores.

3.2.5 Captura dos fluxos de dados das interfaces de rede

O sistema de monitorização de rede, opera de forma transparente à normal utilização da rede, por parte das aplicações. Desta forma, quando um pacote chega à interface de rede, esta envia ao *cpu* um pedido de interrupção da computação. Neste ponto, é desligada a atenção do processador a novas interrupções, passando a computação para o controlador da interface de rede. Com a atenção do processador a novas interrupções desligada, a computação deve ser breve e restabelecer-se, logo de imediato, a atenção do processador a novas interrupções, de modo a que o normal funcionamento seja retomado. Apenas a computação crítica é efectuada, diferindo a restante execução através da invocação de um *softirq*, para que seja terminado o tratamento dos pacotes que chegaram à interface. O escalonamento do *softirq* potencia o aproveitamento dos recursos ao equilibrar a execução de interrupções com as restantes tarefas.

Os pacotes recebidos são entregues aos *sniffers* registados no sistema, antes de o serem aos *packet handlers* respectivos. A monitorização de rede é efectuada pelos *sniffers*, sendo os pacotes entregues a cada um deles, para que possam proceder à sua análise. Cada *sniffer* tem um filtro associado escrito em linguagem *bpf*, a ser executado na máquina virtual, implementada para o efeito.

Ao modificar a função que executa o filtro, de modo a incluir a chamada de um novo sistema de filtragem, permitirá a extensão do *LSF*, sem que contudo, se assista a um elevado aumento da sobrecarga quando esse sistema de filtragem não esteja activo.

3.3 Arquitectura do MRoP

O sistema proposto foi desenvolvido procurando cumprir os seguintes requisitos:

- seleccionar as comunicações que envolvem apenas um processo (ou um conjunto de processos);
- manter a compatibilidade com o sistema já existente, incrementando a sua funcionalidade;

- minimizar eventuais perdas de desempenho;
- a implementação deve envolver poucas alterações ao código do sistema, de modo a facilitar a sua manutenção e evolução aquando das novas versões do sistema *Linux*.

Assim, o sistema criado está dividido em quatro componentes principais: filtragem dos pacotes de rede, instrumentação das chamadas ao sistema, repositório do estado das interacções via rede, e controlo e informação sobre o estado da monitorização (ver figura 3.4). A função de filtragem, invocada por um *hook* que estende o *LSF*, permite que apenas o tráfego do processo alvo seja analisado pelo restante sistema de fitragem do *LSF*. Por outro lado, relativamente à componente instrumentação das chamadas ao sistema (ou outras funções contidas no sistema de rede), esta actualiza a componente repositório de dados, onde é mantido o estado das interacções via rede do(s) processo(s) alvo, com as efectuadas pelo processo. Existe ainda um sistema para controlo/configuração, destinado a obter informação sobre o estado da monitorização.

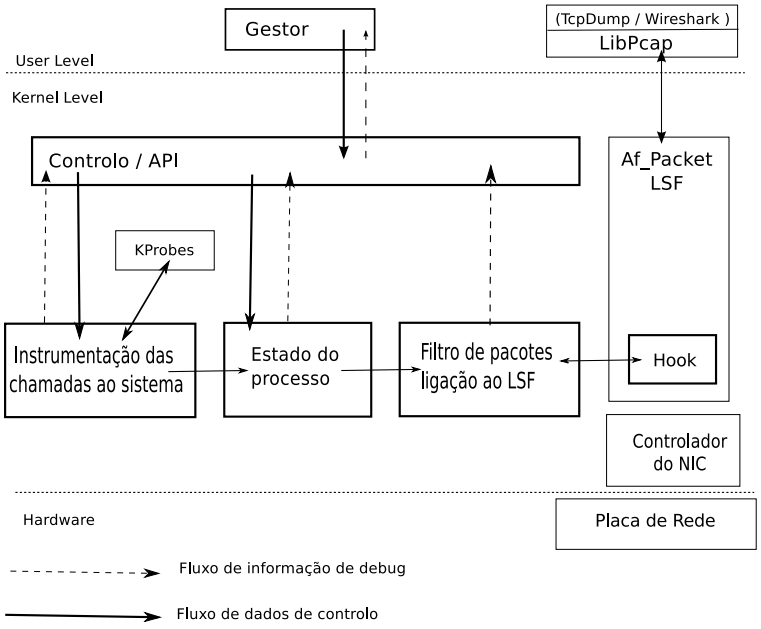

Figura 3.4: Arquitectura do *MRoP*

Este sistema, *MRoP*, captura os pacotes de rede de um processo, sem que exista um conhecimento prévio sobre o(s) protocolo(s) ou portas utilizadas. A utilização de um sistema de instrumentação do núcleo foi necessária apenas para monitorizar as chamadas envolvendo *sockets*, e identificar o processo responsável, permitindo desta forma obter e manter, permanentemente actualizada, a informação relativa ao estado

do processo alvo. De modo a minimizar a redução de desempenho, todo o sistema foi desenvolvido no núcleo do *Linux*, sem alterações nas interfaces já existentes. Assim, ferramentas que façam uso da biblioteca *LibPCap*, como o programa *tcpdump* ou as suas variantes, podem beneficiar desta extensão sem qualquer alteração e sem impacto relevante no seu desempenho.

Este módulo do núcleo foi desenvolvido em quatro componentes, que em seguida serão apresentados:

3.3.1 Instrumentação das chamadas ao sistema de rede

A principal relevância deste sistema, assenta no pressuposto de que todas as interacções desencadeadas por um processo com o exterior são detectadas. Para atingir esse objectivo, foi necessário recorrer à monitorização das chamadas ao sistema de rede ao nível do núcleo, para obter as interacções dos processos, de modo não intrusivo, com as interfaces de rede, possibilitando assim reduzir as cópias de dados e as trocas de contexto. Fazendo uso do sistema de monitorização *KProbes*, foi possível realizar a monitorização sob um limitado conjunto de chamadas ao sistema, nomeadamente: *sendto*, *recvfrom*, *bind*, *accept*, *connect* e *close*. Ao efectuar esta monitorização é possível obter os portos, endereços e protocolos utilizados, de modo a detectar todas as modificações ao estado dos *sockets* do processo. Com os dados relevantes obtidos da monitorização, afectados ao repositório, este mantem-se permanentemente actualizado relativamente aos portos, endereços e protocolos, utilizados pela aplicação. O filtro de pacotes ao aceder ao repositório e ao decidir quais os pacotes a capturar, origina a filtragem dinâmica orientada ao processo.

3.3.2 Estado do processo

O estado dos portos dos protocolos *TCP* e *UDP* em uso no processo alvo, é mantido num repositório de dados e permanentemente actualizado pelo componente anteriormente referido em 3.3.1. A árvore *Red and Black* já disponível no núcleo do sistema, foi a estrutura dados escolhida para produzir o repositório pretendido.

O conteúdo de cada folha da árvore é uma estrutura com duas listas de elementos, contendo cada uma endereços *IP* utilizados pela aplicação, sendo a chave de indexação das folhas, o número do porto. Desta forma, a árvore poderá conter no máximo 65535 elementos, por ser este o número máximo de portos em utilização por um endereço *IP*. No pior caso, a procura de um porto na árvore necessitará de efectuar dezasseis iterações, uma vez que $\log_2 65536 = 16$.

O uso deste tipo de estrutura, permite obter um bom compromisso entre o tempo

de acesso aos dados e a quantidade de memória utilizada.

3.3.3 Filtro de pacotes

A função de filtragem implementada neste sistema assenta no estado do processo alvo, mantido pelos módulos anteriormente descritos. Através da extensão do *LSF* com um *hook*, este quando ligado, invoca esta filtragem que devolve uma resposta ao *LSF*, informando-o se deve ou não capturar o pacote em causa. Caso seja capturado indica que este pertence ao processo alvo, caso contrário irá ser sujeito a um processo de avaliação, baseado nas restantes regras de filtragem definidas. Mantêm-se assim, a compatibilidade e os benefícios da utilização do *Linux Socket Filter*. Quando não existe uma ligação (*hook*) activa, assiste-se a um ligeiro decréscimo no desempenho na utilização da filtragem estática. Tal facto deve-se apenas à necessidade de constatar se a ligação (*hook*) está ou não activa.

Como se verifica, este sistema interage com o filtro estático do *LSF*, efectuando uma conjunção entre o filtro definido pelo utilizador e a captura do tráfego da aplicação a ser monitorizada.

3.3.4 Controlo e Informação

Para facilmente controlar e configurar o sistema desenvolvido, foi definida uma interface baseada em ficheiros virtuais, numa directoria do (*DebugFS*). Esta interface que contém ficheiros, de modo a indicar ao *MRoP* quais os identificadores do processo a monitorizar, permite ao administrador obter informações estatísticas da monitorização, bem como controlar o repositório de dados do estado do processo. É através desta interface, que os actuais sistemas de monitorização de rede, podem usufruir da funcionalidade disponível através do *MRoP*.

Estes ficheiros, com permissões apenas acessíveis ao utilizador *root*, impedem o acesso por parte dos restantes utilizadores da máquina ao sistema de monitorização.

4

Implementação do sistema proposto

O objectivo principal desta dissertação consiste no desenvolvimento de um módulo do núcleo que consiga estender as funcionalidades do *LSF*, de forma a introduzir a funcionalidade de monitorização do tráfego realizado por um processo, tirando partido da instrumentação de código do núcleo. O facto desta instrumentação ser efectuada no núcleo permite que qualquer aplicação possa ser monitorizada, sem que o seu código tenha de ser alterado. Como se pode compreender, esta instrumentação é não instrusiva para as aplicações, havendo, deste modo a necessidade de criar uma método de verificação que identifique o processo que irá cada função instrumentada. Esta adição ao *LSF* irá conter um sistema externo, possibilitando a sua utilização sem a necessidade de modificar qualquer aplicação que tire partido da utilização do *LSF*. As modificações necessárias às extensões das funcionalidades do *LSF* no núcleo estão confinadas ao ficheiro *filter.c*, presente no directório *net/core* do código do *Linux*.

4.1 MRoP e a sua implementação

A implementação da solução *MRoP* teve em consideração o desenvolvimento de um código minimalista, que implicasse ligeiras alterações ao código do núcleo e, ao mesmo tempo, tirasse partido de *APIs* internas. Por outro lado, é transparente à implementação da biblioteca *PCap*, podendo ser integrado na mesma. A implementação está autocontida num módulo do núcleo, de modo a ser carregada e libertada, do mesmo, pelo

administrador. A modularização das diversas componentes, permite um desenvolvimento autónomo de cada subcomponente.

No capítulo 2 foram apresentados, a título exemplificativo (secção 2.2), alguns sistemas que permitem efectuar monitorização ou filtragem de pacotes, com a indicação de um processo. O *MRoP*, embora utilize o sistema de instrumentação do núcleo *KProbes*, apenas o utiliza para instrumentar as interacções com o sistema de rede, diferenciando-se assim das soluções apresentadas, o que lhe permite encontrar-se totalmente implementado no núcleo, com reduzida utilização de memória e perturbação do sistema.

Como apresentado na secção 3.2.5, o sistema de monitorização de rede é utilizado para cada pacote que passa pelo controlador de rede.

Assim quando uma aplicação efectua uma chamada ao sistema (*connect*, *accept*, *bind*, *sendto*, *recvfrom*), o *handler* da função instrumentada é executado. Na execução deste *handler*, obtido do identicador do processo que efectuou a chamada ao sistema e comparado com os identificadores, *pid*, *ppid* e *tgid* internos do *MRoP*, caso um destes seja igual, é obtido o canal através dos parâmetros passados à chamada ao sistema e obtidos os dados sobre o *socket* (porto, endereço e protocolo), para serem adicionados ao repositório. Deste modo quando o *LSF* executar o novo sistema de filtragem irá verificar se os dados do pacote, recebido ou enviado, existe no repositório *Estado do processo* e, caso exista, retorna à função de filtragem do *LSF* que é para o capturar, sendo posteriormente passado para o monitor em nível utilizador.

Nas subsecções seguintes irão ser apresentados os quatro componentes do *MRoP* como se constanta na figura 4.1.

4.2 Instrumentação de funções do núcleo

A metodologia aplicada à resolução do problema de desempenho, baseia-se no desenvolvimento de uma componente para efectuar uma análise aos canais de comunicação de rede, utilizados por um processo, que insira a informação necessária no repositório. Assim que um pacote chega ao sistema de monitorização, o *LSF* tira partido do repositório para decidir a captura dos pacotes.

De entre os sistemas analisados, o *KProbes* foi aquele que permitiu obter menor sobrecarga, apesar do seu carácter dinâmico. Os restantes sistemas contêm componentes de *logging* que, para a realização deste estudo, apresentam uma sobrecarga desnecessária, afectando negativamente o desempenho.

Para além do sistema de instrumentação utilizado foi igualmente considerado o número de funções a instrumentar. A sobrecarga total exercida pela instrumentação de

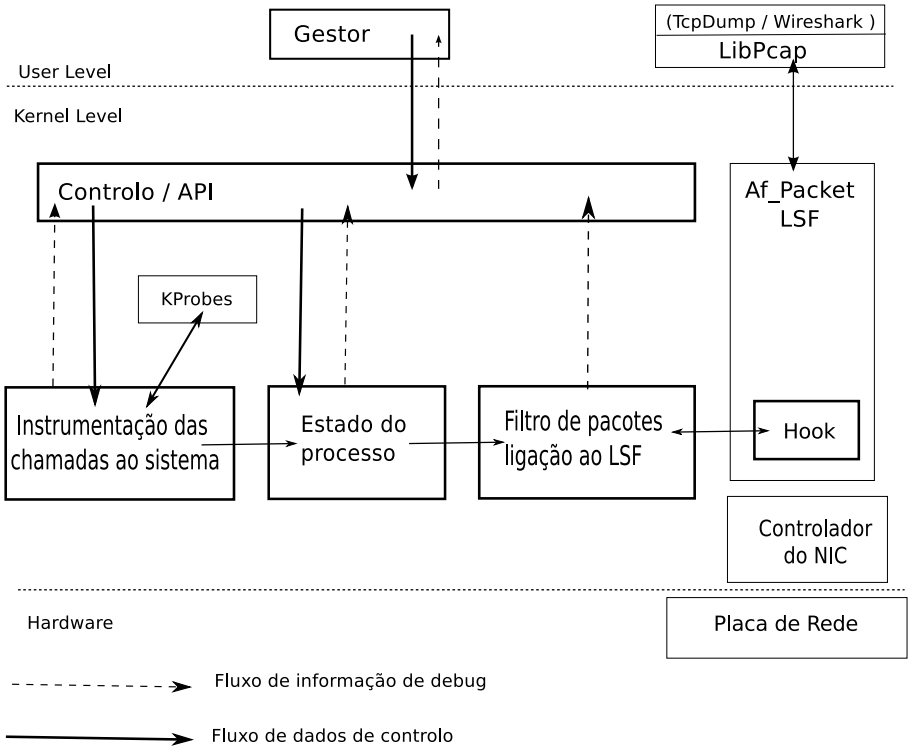

Figura 4.1: Arquitectura geral do MRoP

funções do núcleo, tem em conta não só o número de funções que são instrumentadas, como o número de vezes que estas são executadas.

O conhecimento adquirido com a análise efectuada no capítulo 3, sobre a *Arquitectura de rede em Linux*, afectou positivamente a opção sobre as funções a instrumentar.

O *MRoP*, foi desenvolvido para monitorizar a utilização de canais da familia *INET*, tendo como particularidade, a utilização de canais baseados nos protocolos *TCP* e *UDP*, e instrumentar um número reduzido de funções do núcleo, pertencentes ao subsistema de rede.

O *TCP* e o *UDP* apresentam algumas funções em comum com protocolos de outras famílias, principalmente quando são executadas chamadas ao sistema, onde o nível de abstracção sobre estes protocolos é elevado. Com o objectivo de diminuir o número de funções a instrumentar, optou-se pelas chamadas ao sistema, obtendo assim um maior nível de abstração e um controlo directo dos processos de nível utilizador.

As chamadas ao sistema instrumentadas correspondem: *sendto*, *recvfrom*, *connect*, *bind*, *accept* e *close*. As análises inicialmente efectuadas demonstraram que apenas a chamada ao sistema *close*, era demasiadas vezes executada, penalizando o desempenho global do sistema. Esta situação radica no facto da chamada ao sistema *close*, ser utilizada extensivamente para fechar canais, independentemente destes serem ficheiros, *sockets*, *pipes*, etc. Para contornar esta dificuldade, foi necessário encontrar uma

função que lidasse exclusivamente com o fecho de *sockets*, de modo a reduzir a sobrecarga imposta pela instrumentação.

4.2.1 Filtro de processos

O *KProbes*, é um sistema de instrumentação do núcleo que não distingue entre que funções, não *inline*, está a efectuar a instrumentação. Não existindo suporte no *KProbes* para filtrar o processo, que efectuou a chamada ao sistema, foi necessário desenvolver uma metodologia que permitisse reduzir a sobrecarga, quando a monitorização é realizada sobre um processo, que não o desejado.

No intuito de ultrapassar esta dificuldade, poder-se-ia ter recorrido à criação de um repositório com a informação sobre os identificadores dos processos a monitorizar, sendo necessário, uma vez mais, uma estrutura de suporte a este repositório, bem como funcionalidades de adição, remoção, actualização e consulta. Este repositório teria de conter a estrutura genealógica do processo a monitorizar, assim como uma componente que actualizasse essa informação. A actualização desta estrutura poderia ser efectuada através da instrumentação da chamada ao sistema *fork* ou *clone*. Sempre que fossem invocadas as funções já referidas, seria efectuada uma consulta ao repositório e, a partir da informação obtida, este seria ou não, actualizado. Esta possibilidade, ao contemplar a remoção de dados do repositório, necessitaria também de instrumentar a função de término de processos.

Considerando que a actualização dinâmica deste repositório, sem aplicar alterações no código do núcleo, afectaria negativamente o desempenho do sistema na sua totalidade. Assim, optou-se por excluir esta alternativa em virtude da criação e destruição de processos que podessem apresentar custos elevados.

Face a esta situação, optou-se por efectuar uma análise aos campos da estrutura (*task_struct*), o que permitiu compreender o modo como os identificadores dos processos se relacionam com os identificadores dos membros da árvore genealógica do processo. Desta análise concluí-se que, os identificadores *pid*, *tid* e *ppid* permitem, na grande maioria das aplicações, identificar toda a árvore genealógica.

4.3 Estado dos *sockets* do processo

De modo a manter a informação relativa aos endereços e portos em utilização por uma aplicação, sem requer a consulta sobre os canais de rede, foi necessário criar um repositório de dados, que contivesse informações relevantes para as decisões do filtro de captura. Neste repositório, existe a necessidade de ter funções de inserção, remoção e consulta, sendo que qualquer uma destas, deverá ser efectuada com celeridade,

procurando estruturas que tenham uma complexidade temporal *BigO*, *O(1)* ou *O(log n)*. A estrutura de dados necessária para suportar o *Estado dos sockets do processo*, será uma estrutura que tenha, no máximo, uma complexidade temporal de *O(log n)* sobre as pesquisas, dado que as estas irão ser muito superiores às inserções e remoções.

Assim, as estruturas de dados com suporte no núcleo, que permitem a criação de um repositório de dados, como o requerido, são:

BitMap - O recurso a um mapa de *bits*, permite, de um modo bastante rápido e com uma reduzida utilização de memória, conhecer se um determinado porto está em utilização, por um determinado processo. O núcleo do sistema possui suporte para o tratamento de mapas de *bits*, permitindo representar cada porto usado pela aplicação por um *bit*. Embora este processo seja extremamente rápido, carece de modularidade, na medida em que apenas controla os portos, não dispondo de suporte para protocolos ou múltiplos endereços de rede.

Listas - No núcleo existe uma implementação bastante eficiente da estrutura de dados *lista* (*list*), contendo apenas dois apontadores, destinando-se um ao elemento que o precede e outro ao que se lhe segue.

Embora não seja necessário definir uma lista com o número máximo de portos, dado que estes podem ser adicionados dinamicamente, a complexidade temporal de pesquisa, no pior caso, é de *O(n)* (traduzindo-se num mau indicador de desempenho para o estudo pretendido nesta dissertação). No entanto, quando o número de elementos não é elevado, a utilização de uma lista apresenta-se como uma boa solução.

Árvore Balanceada - *Red-black Tree*

No núcleo existe uma implementação parcial de uma árvore *Red-black* genérica, a fim de permitir o acesso aos dados através de chaves, ou seja, trata-se de uma estrutura associativa. A árvore *Red-black* é semi-balanceada, isto é, a diferença de alturas entre o ramo mais profundo e o mais curto é de apenas de 1 nível. Esta propriedade é mantida através do rebalanceamento da árvore em inserções e remoções, o que provoca um custo na sua utilização. Contudo, no caso esperado, existe maior número de consultas do que inserções e remoções, fazendo com que o custo associado ao rebalanciamento da árvore seja amortizado. De modo a tirar partido da utilização desta estrutura de dados é necessário definir três funções da manipulação da árvore (inserção, remoção e consulta). O suporte disponibilizado pelo núcleo apenas permite manusear a árvore, sendo necessário definir as funções que utilizam a chave de acesso para aceder ao conteúdo dos dados, que devido à sua especificidade, não foram incluídas no suporte.

De referir igualmente que, para o objecto deste estudo, o número do porto dos protocolos (*TCP* e *UDP*), é considerada a chave mais adequada. Considerando o número máximo de portos possíveis nos protocolos (*TCP* e *UDP*), a árvore terá de conter 65535 elementos, com uma altura máxima de 16, ou seja, para pesquisar um dos elementos nos extremos (máximo ou mínimo) é necessário efectuar sobre ela, 16 iterações. Embora não constitua um requisito, é possível obter de forma ordenada todas as chaves, bem como os valores que lhe estão associados.

Outra estrutura de dados passível de ser considerada foi a tabela de dispersão, apesar de não possuir suporte no núcleo.

Tabela de Dispersão No núcleo existe uma implementação de tabelas de dispersão que efectuam a dispersão e o controlo sobre as suas chaves. O controlo sobre as chaves e a forma de dispersão é efectuada pela implementação, ou seja, não existe controlo do programador sobre as chaves nem sobre a forma de dispersar estas.

No núcleo existem subsistemas que implementaram tabelas de dispersão com base em *arrays* e listas, permitindo assim utilizar as vantagens desta estrutura de dados. Estas implementações tiram partido do conhecimento do dominio do problema que resolvem, assim, os *arrays* são criados com dimensões fixas, dado que não necessitarão de efectuar redispersão dos elementos nela contidos.

Como existe a necessidade de uma estrutura que se adapte ao comportamento dinâmico das interacções das aplicações com as interfaces de rede, seria necessário um estudo aprofundado sobre estatísticas do número de portos utilizados pelas aplicações e, caso este estudo fosse realizado seria necessário continuar a efectuar uma redispersão dos elementos, a não ser que, fosse utilizado um *array* com 65535 posições, valor este que representa o valor máximo de portos possíveis através dos protocolos *TCP* e *UDP*, desperdiçando assim memória do sistema.

A opção pela estrutura de dados *Red-black tree* deve-se principalmente ao desempenho e à existência de implementação da estrutura no núcleo do *Linux*.

O ciclo de desenvolvimento do *MRoP*, foi efectuado mais rapidamente, tendo em consideração a confiança na validação da estrutura de dados *Red-black tree*, uma vez que esta está em utilização no núcleo do sistema *Linux* e tem sido sujeita a uma análise extensiva ao longo dos anos.

4.3.1 Estrutura utilizada

Os elementos do repositório criado, através de uma árvore *Red and Black*, têm uma estrutura bem definida, contendo obrigatoriamente um *rb_node*, para possibilitar a manipulação da árvore e um outro elemento, de carácter comparativo, utilizado como chave.

Além dos referidos, esta estrutura contempla outros elementos que seguidamente se descrevem:

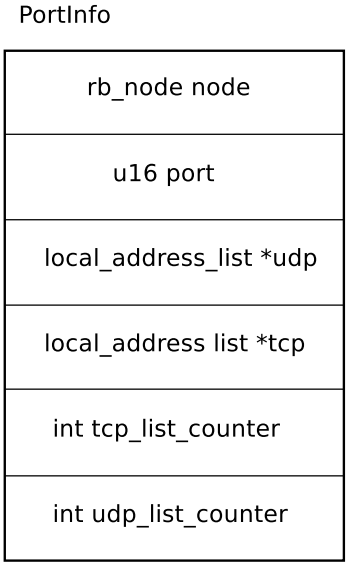

Figura 4.2: Elemento da árvore

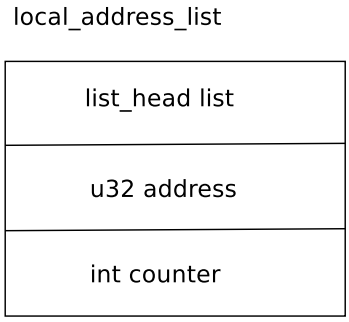

Figura 4.3: Lista de endereços

A figura 4.2 apresenta a disposição dos elementos da estrutura *PortInfo*, sendo que as listas de endereços *IP* das interfaces de rede, são adicionadas através na estrutura apresentada na figura 4.3. Os elementos do repositório, correspondem a instâncias da estrutura *PortInfo*, aos quais são adicionados através dos *handlers* das funções instrumentadas, em conformidade com o esquema apresentado na figura 4.4.

4.3.2 *API* de comunicação interna do MRoP

Com o objectivo de efectuar inserções, consultas e remoções dos dados do repositório, foi desenvolvida uma *API* interna ao *MRoP*, que permite validar os parâmetros passados às funções do repositório de dados. Através desta *API* foi possível realizar a separação das componentes do *MRoP* beneficiando, deste modo, a modularidade do código.

Esta *API* permite aos *handlers* das funções instrumentadas, efectuar as operações de

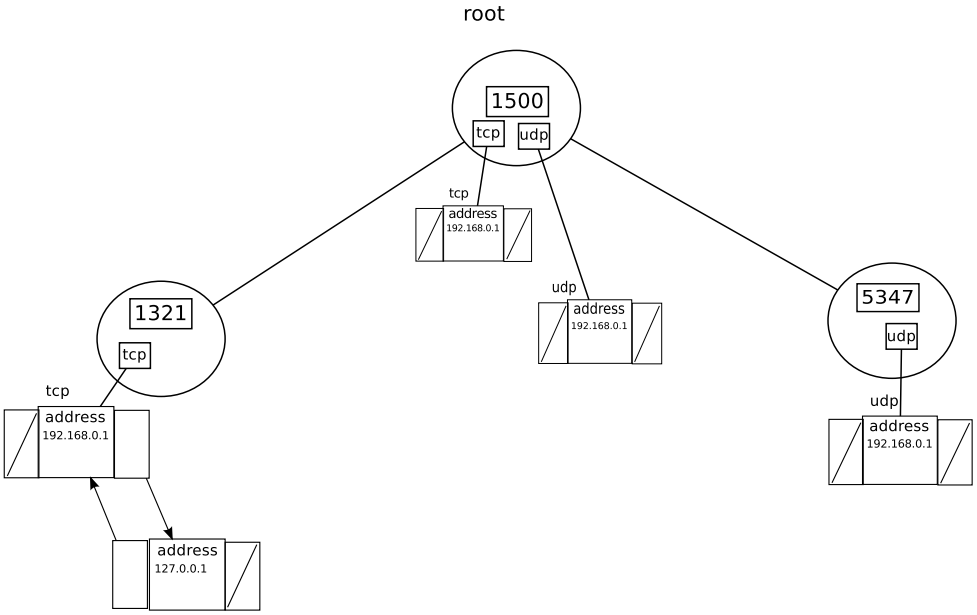

Figura 4.4: exemplo do repositório *Estado do Processo*, com 3 portos

inserção e remoção sobre a componente *Estado dos Sockets*. Relativamente à operação de consulta, esta é particularmente importante, na medida em que executa a filtragem de pacotes das interfaces de rede.

A *API*, não obstante de, reduzir o desempenho, devido à necessidade de chamar os métodos específicos ao repositório, permite a substituição deste, sem que se verifique alterações do código.

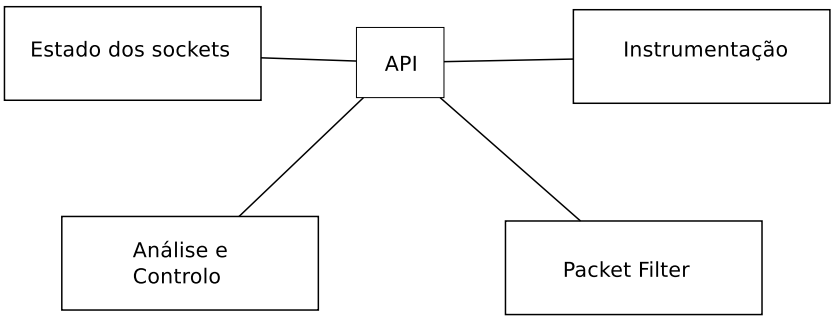

Figura 4.5: API interna do MRoP

Apesar dos processos monitorizados apresentarem elevado dinamismo nas interacções das comunicações, a maior percentagem de operações sobre o repositório concentra-se ao nível da consulta. As restantes operações (inserção e remoção) repartem entre si igual percentagem. Assim, é esperado que para cada inserção exista uma remoção. No final da monitorização de um processo, a componente *Estado dos Sockets* deverá apresentar um número de elementos idêntico ao que antecedeu a monitorização.

4.4 Filtro de pacotes, extensão ao LSF

No sistema de monitorização de rede, um dos componentes corresponde a uma função que serve de máquina virtual às instruções do *LSF*. Esta função itera sobre o filtro, executando instrução a instrução, sobre o pacote (recebido ou enviado), até ao momento em que identifica uma das instrução de retorno (*BPF_RET* ou *BPF_RET_A*). Consoante o valor retornado nestas instruções, o pacote analisado é, ou não, capturado. Caso seja capturado, é efectuado um *clone* do pacote e colocado num *ring buffer*, partilhado com a aplicação monitora de rede em nível utilizador. Caso o valor retornado não corresponda a uma captura, a computação sobre esse pacote termina, reduzindo a sobrecarga da monitorização. Assim, o facto de se tirar partido da utilização de um filtro, que identifique de forma célere a rejeição de um pacote, diminui consideravelmente a sobrecarga imposta ao sistema por parte da monitorização.

Face aos benefícios obtidos pela utilização do filtro, considera-se que o sistema de instrumentação do núcleo (*KProbes*), poderia ser utilizado para invocar o novo sistema de filtragem criado e modificar o valor de retorno, caso se tornasse necessário. Todavia, face ao número de vezes que a função de filtragem é invocada (uma para cada pacote, recebido ou enviado), a sobrecarga da utilização do *KProbes* é de todo desaconselhável.

Assim, foi necessário modificar o código do *Linux* de modo a inserir um *hook*, ou seja, um apontador para uma função. Esta função será invocada quando o filtro estático for avaliado para captura, permitindo que a função de filtragem do *MRoP* analize o pacote com base no estado do processo, possibilitando assim, efectuar uma conjunção entre o filtro estático definido pelo administrador e a filtragem dinâmica efectuada pelo *MRoP*, como pode ser observado na figura 4.6.

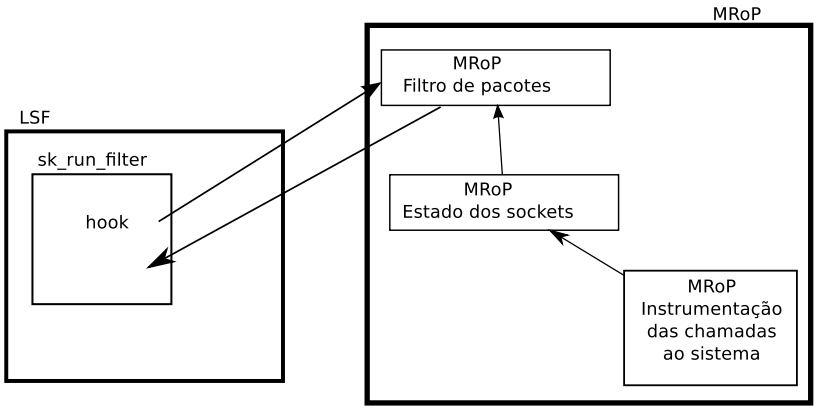

Figura 4.6: Execução da nova filtragem de pacotes pelo LSF

Assim, além da definição de um *hook* para activação e desactivação deste novo sistema, foi efectuada uma alteração ao código do *Linux* na função *sk_run_filter*, que se

traduziu no retorno da decisão conjunta do filtro estático com o dinâmico, ambas realizadas no ficheiro *filter.c*.

Esta alteração permitiu adicionar uma nova funcionalidade, não obstante se assistir a uma ligeira sobrecarga no sistema de monitorização, independentemente da mesma estar ou não activa.

4.5 Informação de análise e controlo

O *MRoP* foi desenhado e implementado de modo a ser relativamente autónomo, apenas necessitando da configuração de alguns parâmetros, essenciais ao seu funcionamento.

Com o propósito de obter informações sobre o estado da computação dos diversos componentes da ferramenta e de invocar a monitorização, procedeu-se à criação de alguns ficheiros no *DebugFs*. Assim, no que se refere à componente *Informação de análise e controlo*, esta fica responsável pelos diversos aspectos da instrumentação e do repositório.

4.5.1 Informação de controlo

Os ficheiros de controlo criados foram: *pid*, *ppid*, *tgid* e *option*. Estes ficheiros à excepção do último, são ficheiros que podem ser lidos e escritos pelo administrador, que caso já tenham sido escritos, contêm o identificador referente ao processo a monitorizar. Caso não tenham sido escritos o valor por omissão é 0. O ficheiro *option*, tem apenas a permissão de escrita e, dependendo do valor escrito, pode activar a procura de portos no processo indicado em *pid*, apagar todos os dados do repositório e activar ou desactivar o filtro dinâmico.

4.5.2 Informação de análise

A informação de análise apenas é adicionada, caso o seu suporte seja activado na compilação, permitindo assim obter estatísticas sobre os diferentes componentes internos ao *MRoP*.

Os ficheiros de análise apenas estão apenas disponíveis para leitura, dado que quando lidos, devolvem os valores presentes nos contadores internos do *MRoP*. Sobre os filtros estes contadores têm o número de pacotes que passaram no filtro dinâmico, bem como o número de pacotes que foram declarados para captura. No que se refere à monitorização do processo, existe também um ficheiro que devolve, em relação a cada uma das funções instrumentadas, o número de vezes que foi executado o *handler*

e quantas destas execuções pertenciam ao processo alvo. Relativamente ao repositório de dados, foi também criado um ficheiro com estatísticas, onde incidiram sobre o número de portos em utilização e, para cada porto, a indicação de estar estar em utilização, através do protocolo *TCP*, *UDP* ou em ambos, e por qual ou quais endereço(s) de rede.

5

Avaliação

Antes de se utilizar um novo sistema, é conveniente proceder-se a testes alargados de verificação, para que se tome conhecimento do seu correcto funcionamento e das sobrecargas introduzidas que de modo a este possa ser utilizada como uma mais–valia. O sistema implementado (*MRoP*) foi avaliado funcionalmente através da utilização de comandos específicos para a transferência de dados, baseados nos protocolos *ftp*, *http* e da aplicação *iperf*. Para esse efeito recorreu-se a um conjunto alargado de testes, tendo como principal objectivo verificar o correcto funcionamento da aplicação, a capacidade de capturar todos os pacotes envolvidos nas comunicações do processo alvo (e apenas estes), bem como o seu desempenho e a sobrecarga introduzida. A verificação da captura dos pacotes envolvidos nas comunicações do processo alvo foi realizada, recorrendo à ferramenta *WireShark* (processo esse, descrito na secção 5.1). Tendo em vista a realização de testes que avaliam o desempenho da aplicação, foram utilizadas duas máquinas, conectadas directamente, conforme o descrito na secção 5.2. Por fim abordar-se-à, detalhadamente, o desempenho efectivo da aplicação.

5.1 Avaliação Funcional

A análise funcional foi efectuada recorrendo a programas simples, que desencadeiam chamadas sucessivas de criação de *sockets* e canais de comunicação, verificando-se o estado dos mesmos (portos e endereços), dentro do módulo do núcleo. Estes dados foram confirmados no sistema de ficheiros *DebugFs*, através de consultas aos ficheiros

existentes, para o efeito. O ficheiro responsável destes dados, contém toda a informação relativa aos portos e endereços em utilização, por parte da aplicação monitorizada. Deste modo, efectua a comparação dos dados produzidos e valida esta análise utilizando, para além de comportar com o próprio programa, foi a ferramenta (*netstat*), que indicou os portos e endereços utilizados pelos processos no sistema. A ferramenta anteriormente referida, utiliza o sistema de ficheiros virtual *ProcFs*, para obter os dados relativos aos portos e endereços utilizados, de modo a estabelecer uma análise comparativa com os dados produzidos pelo *MRoP*. Para além desta verificação, foi efectuada uma confirmação de que todos os pacotes pertencentes às comunicações foram, de facto, correctamente capturadas. Para tal, recorreu-se à captura de pacotes, por intermédio do *TcpDump* com o módulo *MRoP* activo. Através deste processo constatou-se que todo o tráfego, referente aos protocolos (*ftp* e *http*), estava de facto completo e correcto, desde da abertura ao fecho das conexões, não existindo outros pacotes em outros processos na captura. Esta validação foi ainda conseguida, por intermédio da utilização do programa *WireShark*, no qual permitiu identificar os fluxos de dados referentes aos protocolos das novas aplicações alvo.

5.2 Avaliação do desempenho

Tendo presente a avaliação do desempenho, foram efectuados diversos testes com o objectivo de avaliar a sobrecarga gerada pela introdução do *MRoP*. Estes testes basearam-se na recepção ou transmissão de *1 GigaByte* de dados, utilizando diferentes programas e protocolos. Ambas as máquinas, que se optou por designar de máquina 1 e máquina 2, procederam à transmissão/recepção de dados, utilizando cada uma, apenas, um processador activo de 2 e de 2.6 Ghz, respectivamente. As máquinas anteriormente descritas encontravam-se conectadas directamente, por interfaces de rede a 100 MBit/s, ficando uma das máquinas responsável pela execução dos servidores *ftp*, *http* e *iperf*, e a outra pelos respectivos clientes. A versão do sistema de operação utilizado, em ambas as máquinas, correspondeu ao 2.6.39, sendo que na máquina 1 foram introduzidas as modificações, para incluir o *hook* do *MRoP* e as suas funções auxiliares, enquanto na máquina 2 se executou o sistema original.

5.2.1 Desempenho do MRoP

Na execução destes testes, foram efectuadas dez iterações, isto é, cada teste foi executado dez vezes, para cada experiência considerada, de modo a obter um valor médio e um desvio padrão considerado aceitável. os testes efectuados, em particular os primeiros, ilustram situações em que não há grande vantagem em ter o sistema o *MRoP*

activo, com vista a medir a sobrecarga do *MRoP*. Os resultados obtidos constam nas tabelas 5.1 e 5.2:

Tabela 5.1: Tempos médios em segundos (s)

Teste	Original	Com TcpDump	Com TcpDump e MRoP
1GB - FTP[1]	91.8508	91.8500	91.8854
1GB - HTTP[2]	91.6391	91.6472	91.6674
IPerf - 1GB TCP[3]	91.3790	91.2535	91.2672
IPerf - 1GB UDP[4]	89.7975	89.8007	89.8464
1GB HTTP - 2 conexões[5]	182.1573	188.7156	182.0161
IPerf - 1GB UDP 2 conexões[6]	179.4930	179.6280	179.6369

Tabela 5.2: Sobrecarga das transferências (valores em percentagem)

Teste	TcpDump	TcpDump com MRoP
1GB - FTP[1]	-0.0009	0.0377
1GB - HTTP[2]	0.0088	0.0309
IPerf - 1GB TCP[3]	-0.1373	-0.1223
IPerf - 1GB UDP[4]	0.0036	0.0545
1GB HTTP - 2 conexões[5]	3.6003	-0.0775
IPerf - 1GB UDP 2 conexões[6]	0.0752	0.0802

Os primeiros 4 testes foram efectuados utilizando apenas uma conexão ao servidor, enquanto o 5º e o 6º teste utilizaram mais uma comunicação, de modo a aumentar o peso sobre o processador e o número de pacotes a circular entre as máquinas. Desta forma, foi possível identificar a sobrecarga exercida enquando o *tcpdump* executava e capturava todos os pacotes ou apenas um subconjunto destes, ou seja, os pacotes relativos aos processos alvo no novo sistema. A coluna "Original"corresponde aos valores resultantes dos tempos médios das execuções das transferências na ausência de monitorização. A coluna "Com *TcpDump*"apresenta a média dos tempos de transferência com a captura total do tráfego utilizando a *LibPCap/LSF* original, enquanto que a coluna identificada com "Com *TcpDump* e *MRoP*"regista a média dos tempos para a transferência com captura pelo tcpdump e o módulo *MRoP* desenvolvido no núcleo, de forma a capturar, apenas, o tráfego da transferência do processo alvo. Nos primeiros quatro testes é possível verificar que a utilização do *MRoP*, aumentou de forma insignificativa o tempo de execução (figura 5.1). É igualmente possível observar que no

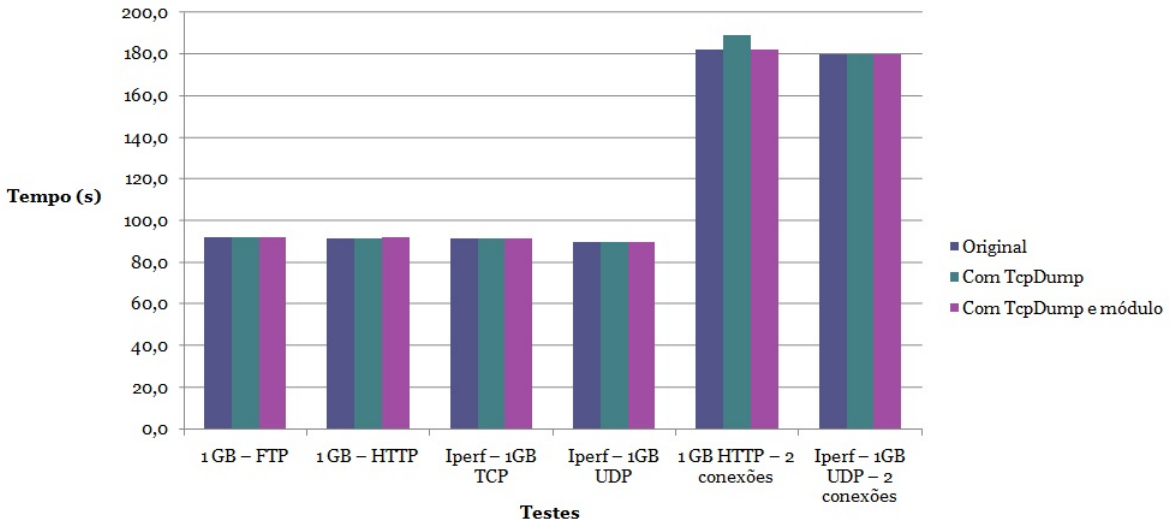

Figura 5.1: Testes de desempenho efectuados ao MRoP

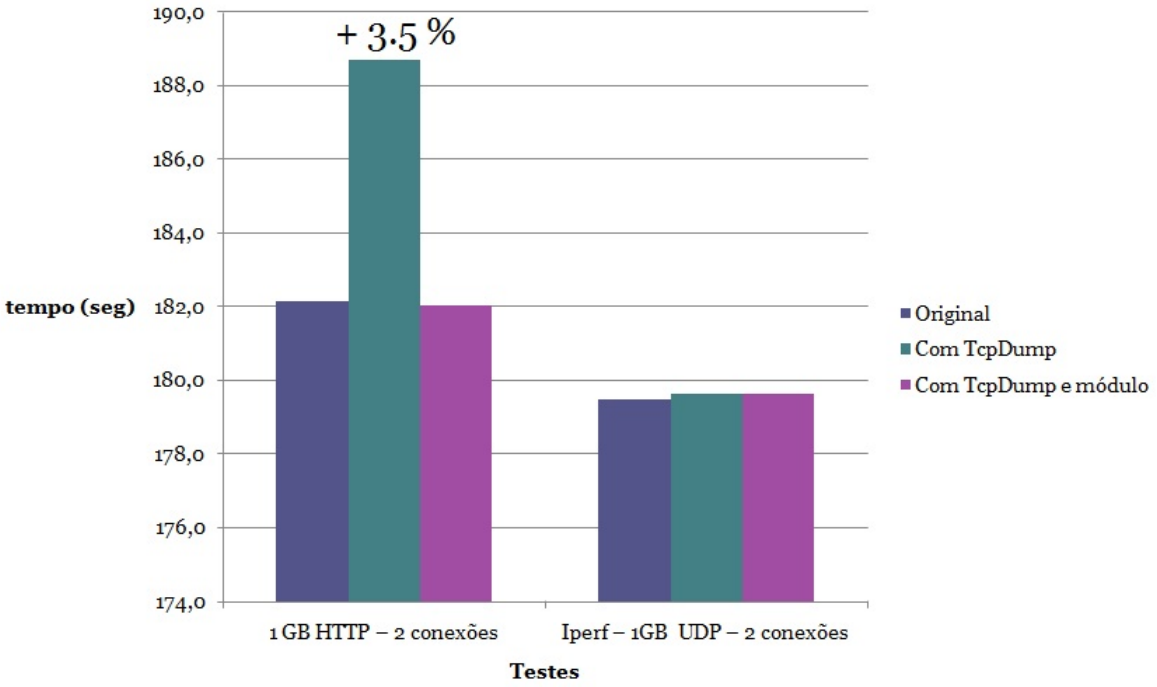

Figura 5.2: Sobrecarga nos testes 5 e 6

1º e 3º testes, aquando da utilização do *tcpdump*, a execução sem o *MRoP*, mostrou-se muito ligeiramente mais rápida, como se pode verificar na tabela 5.1 e 5.2.

Esta situação pode dever-se ao facto de, quando a máquina se encontra em sobrecarga, leva ao aumento do tamanho médio dos pacotes, reduzindo o seu número e o

volume de dados transferidos, em virtude da diminuição dos seus cabeçalhos.

Nos testes 5º e 6º testes, como o tráfego na interface é duplicado e o *tcpdump* tem que capturar todos os pacotes, é possível evidenciar a sobrecarga exercida por estas cópias de dados e consequentes transferências (para nível utilizador) face ao novo sistema onde apenas captura um fluxo de dados. Na tabela 5.2 e na figura 5.2 é possível observar que, para o teste 5, a sobrecarga do *tcpdump* atinge os 3.6% face ao original, enquanto que a sobrecarga do *tcpdump* com o *MRoP*, permitiu uma ligeira melhoria face ao original (-0.0775%). Conclui-se, portanto, que quando o fluxo de dados que não pretendemos capturar aumenta consideravelmente, torna-se mais vatajoso utilizar o *MRoP*, do que capturar todos os pacotes, na medida em que minimiza-se a sobrecarga, captura-se apenas os dados relevantes e evita-se a eventual análise em nível utilizador, ou seja, na identificação e filtragem dos pacotes pertencentes ao processo alvo (o que acarretaria numa sobrecarga adicional).

5.2.2 Desempenho da estrutura de dados

Para além das avaliações anteriormente descritas, tornou-se essencial analisar o comportamento da estrutura de dados utilizada para manter o "estado do processo", de modo a verificar o seu desempenho. Assim, para esta análise, foi elaborado um teste que utiliza o sistema de alta resolução de temporizadores (*HRTimer*)[GM], contido no núcleo do sistema de operação. O teste consistiu em obter o tempo anterior e posterior à inserção dos 1024 elementos, representando outros tantos portos/endereços, afim de determinar o tempo decorrido. De igual modo, foi calculado o tempo de remoção dos referidos elementos. Os resultados gerados estão reproduzidos na tabela 5.3.

Tabela 5.3: Custo das operações (tempos em nanosegundos)

Teste	Duração	Média por elemento
Adição de 1024 elementos	869 244	848.8711
Remoção de 1024 elementos	675 086	659.2637

Como se pode verificar, pela tabela 5.3, a inserção de um elemento na árvore é inferior a 1 microsegundo, possivelmente demonstrando que a estrutura utilizada foi a correcta. Para além de estabelecer um bom compromisso de desempenho e utilização de memória, a sua disponibilidade de utilização no núcleo do sistema, possibilitou obter ter um elevado grau de confiança na sua utilização. O tempo médio despendido na procura do elemento com o menor valor de chave, nos 1024 elementos adicionados, foi de 1327 nanosegundos. Com este valor é possível verificar que para efectuar 10

iterações de procura na árvore, incorre-se numa penalização de 1.3 microsegundos. Verifica-se assim, que o tempo médio de procura de elementos na estrutura, é menor ou igual a 1.3 microsegundos. Considerando que a maioria das aplicações não utiliza tantos portos em simultâneo, são expectáveis tempos inferiores em aplicações reais.

5.2.3 Desempenho do Sistema de instrumentação

Sendo a instrumentação das chamadas ao sistema um ponto fundamental na execução da monitorização de uma aplicação, a análise ao seu comportamento é bastante importante, na medida em que é necessário verificar se a introdução deste tipo de sistema irá produzir uma elevada penalização sobre o sistema de operação. A análise efectuada consistiu em colocar um *KRetProbe* na chamada ao sistema *getpid* e avaliar o tempo decorrido entre o início e o fim do total das chamadas, com e sem o *KRetProbe*, de forma a avaliar a sobrecarga e verificar se coincide com o indicado pelos criadores do sistema. A chamada ao sistema escolhida foi o *getpid*, pois esta é uma função muito simples onde apenas devolve o identificador do processo que a invocou.

Tabela 5.4: Duração das chamadas em segundos

Teste	Original	Com *KRetProbe*	Sobrecarga por chamada
100 000 000 chamadas	12.65	73.6600	610.10×10^{-9}
1 000 000 000 chamadas	126.85	737.2100	610.36×10^{-9}

Os valores de referência nos manuais do *KRetProbe* são de 0.7 microsegundos[KPH], sendo que os valores obtidos foram de 0.61 microsegundos, ou seja, ligeiramente inferiores, visto que a máquina de referência apresenta uma velocidade inferior à máquina onde foram realizados estes testes.

Consideram-se estes valores bastante aceitáveis e espera-se que tenham ainda pouco impacto no desempenho normal do sistema. Note-se ainda que esta instrumentação só é introduzida aquando do carregamento do módulo para executar a monitorização com esta nova funcionalidade.

6

Conclusões e Trabalho Futuro

Esta dissertação teve como objectivo a criação de uma extensão ao actual *PCap/LSF*, utilizado no *Linux*, de modo a restringir a captura dos pacotes referentes a uma determinada aplicação, contribuindo assim para uma nova funcionalidade e a redução da sobrecarga no sistema de monitorização e identificar os fluxos de rede de uma forma não intrusiva.

Para a realização da *Monitorização de Rede orientada ao Processo* (*MRoP*), foi necessário, para além de identificar os pontos e conhecer as razões que estiveram na génese do insucesso de anteriores trabalhos, criar alternativas permitissem ultrapassá-los. Para atingir tal objectivo, foram estudados os mecanismos de monitorização internos ao núcleo e os sistemas de comunicação entre o núcleo e as aplicações em nível utilizador.

O estudo centrou-se, principalmente, nos mecanismos de monitorização ao nível do núcleo, pois estes permitem efectuar análises não intrusivas e, devido à sua localização, dispensam a permuta de dados entre o nível utilizador e o núcleo, o que a não acontecer contribuiria para o aumento da sobrecarga.

Para executar a extensão ao *LibPCap*, foi arquitectada uma solução utilizando o sistema de instrumentação dinâmica do núcleo (*KProbes*), para a análise das interacções do processo alvo com o exterior. Os dados relevantes desta interacção, são adicionados a um repositório, de modo a que o sistema *LSF* os possa consultar e decidir quais os pacotes a capturar, com base nestas informações.

Esta extensão foi analisada funcionalmente através de programas que criavam, comunicavam e destruiam canais de comunicação, onde todas estas interacções eram

registadas e verificadas. Na avaliação funcional, foram igualmente executados programas que efectuavam transferências utilizando os protocolos *HTTP* e *FTP*, enquanto decorriam outros fluxos na rede. O *MRoP* foi aplicado a estes programas, de modo a capturar todo o trafego respeitante às transferências destes dois protocolos, a compatibilidade está garantida, continuando a poder utilizar-se as ferramentas existentes como o *tcpdump* e o *Wireshark*. Foi possível constatar a correcção dos protocolos e verificar que apenas as interacções da aplicação alvo com o exterior, foram capturadas. Para além desta análise funcional, foi efectuada uma outra, onde foram comparados os desempenhos do *PCap*, com e sem esta nova extensão, a fim de determinar qual a sobrecarga introduzida pelo *MRoP* na monitorização já existente. Os resultados apurados evidenciam que a sobrecarga é praticamente inexistente nos piores casos, e que trará algumas vantagens perante vários fluxos de dados irrelevantes.

Nas secções seguintes são apresentados as principais conclusões da realização desta dissertação, assim como as possíveis evoluções e extensões ao *MRoP*.

6.1 Conclusões

O objectivo proposto de criar uma extensão ao actual sistemas de monitorização genérico de rede, que inclua a monitorização das interacções de rede de um processo com base no identificador deste e efectuar a captura dos pacotes da aplicação, foi atingido, constatando-se ainda, que a sobrecarga imposta sobre o actual sistema de monitorização é praticamente irrelevante.

O *MRoP* é um módulo do núcleo que permite monitorizar as interacções de rede de um processo, sem que exista um conhecimento prévio dos portos utilizados pela aplicação. Esta monitorização é inócua para a aplicação, porquanto esta desconhece que está a ser monitorizada, possibilitando ao administrador monitorizar aplicações sem acesso ao código fonte.

Se analisarmos a aplicação, do ponto de vista de segurança na rede, é possível verificar se está, ou não, a enviar indevidamente informação para o exterior. Esta situação, não é possível de observar com o recurso ao normal funcionamento da *LibPCap*, sem um conhecimento detalhado do seu funcionamente e protocolos, ou a uma monitorização da aplicação de forma intrusiva, o que pode originar comportamento errático e afectar negativamente o desempenho da aplicação e do sistema. O recurso à introdução desta funcionalidade não intrusiva, para a captura e análise do tráfego de um processo, constitui um avanço relativamente à monitorização de rede efectuada através da *LibPCap*.

O *MRoP* ao oferecer a possibilidade de capturar exclusivamente os pacotes de um

determinado processo ou de uma família de processos, facilita as análises a efectuar e reduz a sobrecarga neste tipo de sistemas, dispensando a *LibPCap* de capturar o tráfego não pretendido, e de conhecer os protocolos e portos utilizados pela aplicação.

Esta funcionalidade, é transparente para todas as ferramentas desenvolvidas com base no *PCap*, pelo que todas podem dela usufruir.

Relativamente à sobrecarga introduzida, tendo como referência a monitorização de rede já existente, esta revelou-se insignificante mesmo nos piores casos, melhorando substancialmente aqueles em que incide sobre o tráfego de um único processo, reduzindo igualmente o trabalho realizado pelo *LSF* e pela *LibPCap*.

As vantagens do sistema criado assumem maior notoriedade, quando a máquina se encontra perante uma carga mais elevada de trabalho, ou um grande volume de tráfego de rede e/ou muitos fluxos irrelevantes, dado manter o uso dos recursos na proporção aproximada apenas do tráfego do processo alvo.

6.2 Trabalho Futuro

Como trabalho futuro existe a possiblidade de expandir e melhorar o suporte para multiplos processos a serem monitorizados. Acresce ainda a possibilidade de verificar problemas de concorrência na presença de multiplos *cores*/*cpus*, e garantir a impossibilidade de ocorrência de *race conditions*.

Considerando que o sistema implementado se limita a monitorizar protocolos acentes em *TCP* e *UDP*, poderia ver a sua contribuição alargada se abrangesse outros protocolos como *icmp, arp, stp*, etc.

Outra possibilidade será procurar optimizar a instrumentação e evitar a instrumentação das funções *sendto* e *recvfrom*, restringindo a sua aplicação a funções internas especificas dos protocolos monitorizados.

Pretende-se partilhar o uso destas funcionalidades submetendo este sistema a analise da comunidade utilizador do sistema *Linux* com vista à sua implementação na versão principal do núcleo do *Linux*. Considera-se ainda a integração deste trabalho com o anterior [DF10, Far09], com vista à obtenção de uma ferramenta de monitorização distribuída com baixa sobrecarga.

7

LibPCap e Linux Socket Filtering

7.1 LibPcap e Linux Socket Filtering

A monitorização de rede existente em muitos dos sistemas tipo *Unix* e mesmo no *Windows* permite capturar os pacotes de rede logo que eles chegam ao controlador de rede. Assim a biblioteca *LibPCap*[Lib] permite às ferramentas de monitorização de rede fazerem uso deste mecanismo. Uma das principais características desta biblioteca, é a sua *API* de alto nível para a captura de pacotes, que é igual em todas as plataformas. Para filtrar os pacotes indesejados a *LibPCap*, utiliza filtros baseados no *BPF (Berkeley Packet Filtering)*, normalmente implementados no núcleo de sistema, de modo a que a monitorização se torne mais eficiente e menos intrusiva.

Em seguida é apresentada a arquitectura da biblioteca *PCap* e o seu suporte no *Linux*, *Linux Socket Filtering*.

7.1.1 Arquitectura

A arquitectura do *Packet Capture* (*PCap*) divide-se em duas partes: a biblioteca a nível utilizador, e a implementação do *Linux Socket Filtering* no núcleo de sistema. Como anteriormente referido a bibilioteca em nível utilizador é constituida por uma *API* homogénea, que oferece às ferramentas de monitorização de rede uma forma de efectuar as suas análises ao tráfego. As funções auxiliares da biblioteca efectuam chamadas ao sistema de modo a interagirem com o mecanismo de monitorização de rede do *Linux*,

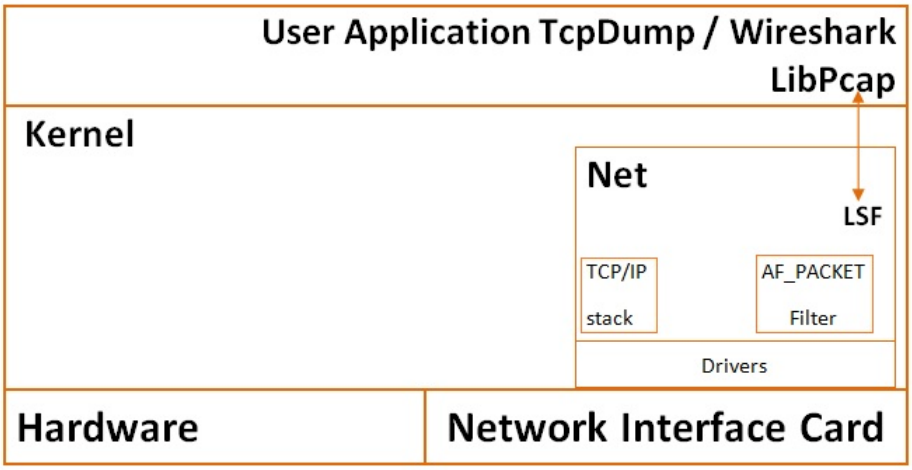

Figura 7.1: Arquitectura do LibPcap

Linux Socket Filtering, presente no núcleo.

7.1.2 Biblioteca

A biblioteca em nível utilizador ao ter uma *API* homogénea entre plataformas, permite aos programadores desenvolver mais facilmente ferramentas de monitorização de rede multiplataforma, tais como o *TcpDump*, o *Wireshark*, *Snort*, etc.

A biblioteca dinâmica *LibPCap* é utilizada pela generalidade das ferramentas monitorização de rede. Esta biblioteca, como é multi-plataforma, permite ao programador utilizá-la nos principais sistemas de operação, porque a *API* da *LibPCap* é transparante relativamente à implementação, esta sim, específica de cada plataforma. Para ser possível efectuar a captura de pacotes, o programa tem de especificar qual a interface de rede a monitorizar.

Necessário criar um canal de comunicação com o núcleo de modo a proceder à monitorização de rede Utilização de *ioctl's* para configurar o canal de recepção dos pacotes obtidos pela monitorização.

Como verificado anteriormente, quando o número de dados da monitorização é elevado, e nem todos os dados capturados são relevantes para as análises, estes são filtrados, de modo a apenas serem utilizados para análise os dados relevantes. Daí que a biblioteca *PCap* utilize uma linguagem própria para a especificação de filtros, sendo posteriormente traduzidos para a linguagem *BPF* para ser aplicado ao canal, obtendo assim apenas os pacotes relevantes para as análise a efectuar.

Se necessitar de filtrar pacotes, especificará o filtro a utilizar, recorrendo às instruções presentes na *LibPCap*, que através da função *pcap_compile* o traduzirá e optimizará para o conjunto de instruções do *BPF*, sendo posteriormente aplicado no núcleo através da função *setfilter*. Se o filtro for demasiado complexo, não poderá ser aplicado no núcleo, sendo apenas possível aplicá-lo em nível utilizador, perdendo algum desempenho. Antes da introdução do novo filtro no canal, este tem de ser objecto de drenagem, para receber apenas os pacotes do novo filtro. Após esta inicialização, o programa poderá utilizar *Polling*, de forma a obter os pacotes e analisá-los.

Este sistema de captura, evita imediatamente a captura de pacotes irrelevantes para a captura e permite que aqueles que não são visíveis às aplicações, devido à *firewall* do *Linux*, possam ser capturados e analisados pelas ferramentas de monitorização de rede.

A arquitectura de rede do núcleo de sistema *Linux*, utiliza *socket buffers*, que contêm dados referentes às diferentes camadas da pilha *TCP/IP*. Esta estrutura que dispõe de toda a informação sobre o pacote, a receber pela aplicação, é igualmente utilizada nos sistemas de monitorização em nível utilizador. A monitorização da rede é efectuada directamente no controlador da interface de rede, utilizando para isso um *socket AF_PACKET*.

7.1.3 Linux Socket Filtering

O módulo *AF_PACKET* permite activar o modo promiscuo da interface a monitorizar, ou seja permite que sejam capturados pacotes que não são destinados à máquina utilizada. Na versão actual do núcleo de sistema do *Linux* (2.6.39), o *socket AF_PACKET* pode utilizar um sistema de partilha de *buffer* (*MMAP*), com o espaço de utilizador. Neste caso, cabe ao utilizador definir explicitamente a forma como pretende utilizar o *socket*. Para atingir este objectivo o utilizador tem de pedir ao núcleo, uma região de memória, sendo esta, posteriormente indicada como argumento de configuração do *socket*.

Como o referido anteriormente, o *AF_PACKET*, permite a comunicação directa com o controlador da interface de rede, possibilitando capturar os diversos pacotes ainda antes destes poderem ser filtrados pela *firewall*, presente no núcleo de sistema, neste caso o *netfilter*. Podem ser aplicados filtros aos *sockets* de modo a aumentar a eficiência na captura dos pacotes, excluindo aqueles que são irrelevantes para a monitorização.

O *Linux Socket Filtering* é derivado do *BPF (Berkeley Packet Filtering)* que é o *standard* de *de facto* para a criação de filtros dentro do núcleo de sistema. Este sistema de captura e filtros permite aos administradores, definir filtros e afectá-los ao *socket*. Os filtros definidos são descritos numa linguagem simples, de forma a efectuarem com rapidez a selecção dos pacotes a capturar, rejeitando todos os outros. A linguagem

utilizada para a criação dos filtros é traduzida para o *Instruction Set*, definido no *BPF*. Este *Instruction Set* utiliza operadores lógicos de forma a combinar as regras definidas nos filtros, criando-se apenas um filtro a ser aplicado aos pacotes. [MJ92].

7.1.4 Limitações e optimizações

Apesar da *API* da *LibPCap* ser igual em diferentes plataformas (*Windows*, *Linux*, *FreeBSD*, etc), o desempenho desta pode variar. Quando exista pouco tráfego de rede estas diferenças são pouco notórias, mas logo que este se intensifica, estas acentuam-se bastante, como demonstra o estudo *Improving Passive Packet Capture: Beyond Device Polling* [Der04].

De modo geral, para a avaliar o desempenho da *LibPcap*, é frequente utilizar-se uma métrica que nos permita determinar a percentagem de pacotes que é possível capturar, sem que se verifiquem perdas. Esta métrica é calculada sabendo o número de pacotes que atravessam um determinado *router ou switch*, e compará-la com o número de pacotes capturados na interface. Uma variante desta medida de desempenho é combiná-la com o máximo número de regras que é possível aplicar, sem que se assista à perda de pacotes. É particularmente importante conhecer o número máximo de pacotes capturados ou regras aplicáveis aos filtros, afim de determinar a velocidade máxima expectável de captura. Quanto mais eficiente for a filtragem, menor o número de pacotes transferidos para as aplicações, o que se traduz numa menor sobrecarga para o sistema, aumentando o seu desempenho.

Captura de pacotes com *TimeStamp* Um dos pontos que pode influenciar negativamente o desempenho da obtenção de pacotes, é a necessidade de colocar nestes, uma estampilha temporal, com o tempo da sua chegada ao sistema. O *LibPcap* pode obter estes dados de duas formas distintas, dependo da interface de rede que esteja a ser utilizada. Se a interface de rede fornecer a estampilha temporal associada ao pacote, o *LibPcap* pode utilizar este valor, caso contrário será necessário obter a estampilha temporal da chegada do pacote ao sistema. A forma de obtenção da estampilha pela *LibPcap*, processa-se através da função *gettimeofday*, o que incorre numa maior sobrecarga do sistema face à solução anterior.

Captura com dns activo tcpdump com parametro n de dns para obter os nomes de dominios sempre que possivel

Captura especificando tamanho máximo de 65535 para pacotes MTU de 1500 ethernet

Diversos esforços no sentido de aumentar o desempenho da captura de pacotes têm sido efectuados. Relativamente ao *software*, são conhecidos esforços na utilização da técnica de *mmap*, de forma a reduzir o número de cópias de dados entre as aplicações e o núcleo de sistema. Igualmente no *hardware*, tem-se assistido a uma evolução no sentido de reduzir as interrupções efectuadas ao *cpu*, adicionando nas interfaces de rede processadores dedicados às funcionalidades presentes no núcleo, de modo a libertá-lo da execução destas tarefas.

PACKET_MMAP Com base na técnica *MMAP*, foi criado o *PACKET_MMAP*, disponível a partir da versão 1.0.0 da biblioteca do *LibPcap*. Este módulo permite algumas melhorias em termos de desempenho, visto que foi reduzido o número de cópias efectuadas e de trocas de contexto, face à anterior versão 0.9.8 da biblioteca *LibPcap*.

Para além destas modificações no *PACKET_MMAP*, o núcleo de sistema de operação *Linux* na sua versão 2.6, passou a contar com a nova *API* de rede (*NAPI*).

Se as interfaces de rede suportarem um mecanismo de mitigação de interrupções, é possível obter melhores resultados, conforme [Der04].

PF_RING Este é um novo módulo para o núcleo de sistema, criado com base em duas técnicas *mmap* e *ring_buffers* anteriormente descritas.

Este módulo difere na abordagem utilizada no *PACKET_MMAP*, porquanto nesta a memória é mapeada entre a ferramenta e o controlador da interface, enquanto que no *Packet_MMAP* a memória é mapeada entre a ferramenta e um *buffer* externo ao controlador da interface, mas interno ao núcleo de sistema. Esta abordagem permite que os dados fiquem disponíveis para a aplicação directamente, verificando-se a inexistência de cópia dos dados do *buffer* do controlador, para o *buffer* partilhado entre a ferramenta e o núcleo[PFR].

PF_RING com DNA (Direct NIC Access) Baseando-se na técnica anteriormente descrita de utilizar um *buffer* partilhado entre a ferramenta de monitorização e o controlador, assiste-se a uma evolução desta técnica, ao permitir que a interface de rede partilhe um *buffer* com a ferramenta de monitorização, possibilitando que os pacotes passem directamente para esta.[Int]. Esta partilha é efectuada utilizando *mmap*, *ring_buffers* e *DMA*. Para se utilizar esta técnica é necessário que a interface de rede, permita a utilização de memória partilhada e *DMA*.

7.2 Optimizações

NAPI - New API - Foi criada uma nova *API* (*Application Program Interface*) de atendimento de interrupções do processador, oriundos das interfaces de rede, com o objectivo de aumentar o desempenho da utilização de rede de elevado débito. Esta nova *API*, permite desligar a atenção do processador a novas interrupções, oriundas da interface de rede, durante um certo período de tempo.

Este tempo foi definido, de forma a que não se verifiquem situações de elevada latência na chegada dos pacotes às aplicações. A forma anterior de recepção de pacotes, era efectuada de modo a atender uma interrupção por cada pacote que chegava à interface de rede, gerando assim demasiada sobrecarga no sistema. Actualmente a arquitectura de rede, está balanceada de forma a mudar o modo de atendimento de interrupções para NAPI, caso existam demasiadas interrupções por unidade de tempo. [Adm09].

Utilizando esta nova *API*, é possível explorar um escalonamento mais eficiente das interrupções, em situações de intensa actividade da interface de rede e do processador. O *NAPI* apenas pode ser aplicado nos caso em que os controladores das interfaces de rede estejam preparados para utilizar alguma forma de mitigação da interrupção, caso contrário esta *API* não é aplicada, resultando em interrupções por cada pacote, enviado ou recebido.

A utilização desta nova *API* permite diminuir o número de trocas de contexto entre o controlador da interface e o núcleo de sistema. Sempre que o cpu atende uma interrupção de rede, obtem um número maior de dados, que combinado com um sistema de memória partilhada aumenta substancialmente o desempenho.

Sistemas *multi-core* e multi-processador Com o aparecimento de sistemas *multi-core* e *multi-processador* a que a generalidade do público tem acesso, a paralelização de código ou a forma de tirar partido destas arquitecturas, que permitem um melhor aproveitamento dos recursos, assumem uma particular importância. De forma a tirar partido das arquitecturas *multi-core*, é necessário que o controlador e interfaces de rede, os *buffers*, e os controladores de *DMA* (*Direct Memory Access*), sejam modificados de forma a conhecerem esta estrutura. É pois, determinante um esforço conjunto envolvendo todas estas componentes, com vista à obtenção do máximo rendimento destas arquitecturas[Der10].

Bibliografia

[Adm09] Linux Foundation Administrator. napi. *The Linux Foundation*, Novembro 2009.

[BDK+97] M. Beck, M. Dziadzka, U. Kunitz, R. Magnus, e D. VerWorner. *Linux Kernel Internals*. Addison-Wesley, 1997.

[BDKV02] M. Beck, M. Dziadzka, U. Kunitz, e D. VerWorner. *Linux Kernel Programming*. Addison-Wesley, 2002.

[Ben05] Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly, 2005.

[Der04] Luca Deri. Improving passive packet capture: Beyond device polling. Proceedings of SANE, Oct. 2004.

[Der10] Luca Deri. *Exploiting Commodity Multi-core Systems for Network Traffic Analysis*, 2010.

[DF10] V. Duarte e N. Farruca. Using libpcap for monitoring distributed applications. In *International Conference on High Performance Computing & Simulation (HPCS 2010)*, pág. 92–97. IEEE, 06 2010.

[DIH+07] Stephanie Donovan, Gerrit Huizenga Ibm, Andrew J. Hutton, Andrew J. Hutton, C. Craig Ross, C. Craig Ross, Linux Symposium, Linux Symposium, Linux Symposium, Martin K. Petersen, Wild Open Source, Tom Zanussi, Karim Yaghmour, Robert Wisniewski, Richard Moore, e Michel Dagenais. relayfs: An efficient unified approach for transmitting data from kernel to user space. 2007.

[DPr] *DProbes*. http://dprobes.sourceforge.net/ Site visitado em Abril de 2010.

[Dua05] V. Duarte. *Uma Arquitectura para a Monitorização de Computações Paralelas e Distribuídas*. Tese de Doutoramento, Faculdade de Ciências e Tecnologia, May 2005.

[Far09] Nuno Miguel Galego Farruca. Wireshark para sistemas distribuídos. Tese de Mestrado, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2009.

[GM] Thomas Gleixner e Ingo Molnar. hrtimers - subsystem for high-resolution kernel timers. Site consultado em janeiro de 2011.

[HMC94] Jeffrey Hollingsworth, Barton P. Miller, e Jon Cargille. Dynamic program instrumentation for scalable performance tools. pág. 841–850, 1994.

[Int] *Introducing PF_RING DNA (Direct NIC Access)*. http://www.ntop.org/blog/?p=50 site consultado em junho de 2010.

[Jon09] M. Tim Jones. Linux introspection and systemtap an interface and language for dynamic kernel analysis. Relatório técnico, Nov. 2009.

[KPH] Jim Keniston, Prasanna S Panchamukhi, e Masami Hiramatsu. Kernel probes (kprobes). Consultado em abril de 2010.

[KPr] Kprobes. http://sourceware.org/systemtap/kprobes/ Site consultado em Março de 2010.

[Lib] *LibPcap*. http://www.tcpdump.org/ Site consultado em Abril de 2010.

[LML09] Byungjoon Lee, Seong Moon, e Youngseok Lee. Application-specific packet capturing using kernel probes. In *IM'09: Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, pág. 303–306, Piscataway, NJ, USA, 2009. IEEE Press.

[MD11] Nuno Martins e Vítor Duarte. Pcap com filtragem orientada ao processo. *Inforum 2011 Simpósio de Informática*, Setembro 2011.

[MJ92] Steven Mccanne e Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. pág. 259–269, 1992.

[MR09] Desnoyers Mathieu e Dagenais Michel R. Lttng, filling the gap between kernel instrumentation and a widely usable kernel tracer. 2009.

[Pan04] Prasanna S. Panchamukhi. Kernel debugging with kprobes insert printk's into the linux kernel on the fly. Relatório técnico, aug 2004.

[PFR] *PF_RING*. http://www.ntop.org/PF_RING.html Site consultado em junho de 2010.

[SV08] Sammer Seth e M. Ajaykumar Venkatesulu. *TCP/IP Architecture, Design, and implementation in Linux*. Wiley, 2008.

[TZYW03] From Kernel To, Tom Zanussi, Karim Yaghmour, e Robert Wisniewski. relayfs: An efficient unified approach for transmitting data. In *In Proceedings of the Ottawa Linux Symposium 2003*, pág. 494–507, 2003.

[vdMChCS00] Jacobus van der Merwe, Ramón Cáceres, Yang hua Chu, e Cormac Sreenan. mmdump: a tool for monitoring internet multimedia traffic. *SIGCOMM Comput. Commun. Rev.*, 30(5):48–59, 2000.

[Wil] E. Cohen. William. Tuning programs with oprofile. *WIDE OPEN MAGAZINE*.

[WR05] Klaus Wehrle e Harmut Ritter. *The Linux Networking Architecture, Design and Implementation of Network Protocols in the Linux Kernel*. Pearson - Prentice Hall, 2005.

[WXW08] Zhenyu Wu, Mengjun Xie, e Haining Wang. Swift: a fast dynamic packet filter. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pág. 279–292, Berkeley, CA, USA, 2008. USENIX Association.