Análise de Desempenho de Interfaces de Rede Virtualizadas com NAPI

Eduardo Hideo Kuroda

Dissertação de mestrado apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciência da Computação

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Daniel Macêdo Batista

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da Comissão Européia e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

São Paulo, setembro de 2013

Análise de Desempenho de Interfaces de Rede Virtualizadas com NAPI

Esta é a versão original da dissertação elaborada pelo
candidato Eduardo Hideo Kuroda, tal como
submetida à Comissão Julgadora.

Resumo

Em ambientes virtualizados, como nuvens computacionais, a capacidade efetiva de transmissão de dados via rede tende a ser inferior do que em ambientes não virtualizados quando aplicações que fazem uso intensivo da rede são executadas. Uma das principais causas para essa diferença na capacidade de transmissão é a arquitetura da virtualização de rede, que adiciona passos para o sistema operacional transmitir e receber um pacote. Esses passos adicionais acarretam em mais utilização de memória e de processamento. Em ambientes virtualizados com o sistema operacional GNU/Linux, a *New Application Programming Interface* (NAPI) é utilizada para reduzir os impactos negativos da virtualização por meio de agregação de interrupções. Nesta dissertação de mestrado, estudamos mecanismos que modificam a configuração da NAPI. Experimentos mostraram que esses mecanismos afetam o desempenho de máquinas virtuais e tem consequências diretas nas aplicações que fazem uso intensivo de rede e que são executadas em ambientes com os softwares de virtualização Xen, VMware e VirtualBox.

Palavras-chave: computação em nuvem, virtualização de rede, agregação de interrupções, NAPI

Abstract

Cloud infrastructures shows inferior performance if compared to infrastructures without virtualization in scenarios where there is intensive use of network. One of the main reasons is the network virtualization's architecture. Differently from the standard network architecture, there are some extra steps to transmit and receive a packet of information within a virtual machine, which implies an additional cost in both memory and processing. To reduce the processing cost, a modern solution is the NAPI. NAPI is a set of interfaces provided by the Linux kernel that the drivers of network devices use to coalesce interruptions. To control the amount of interruptions coalesced, there is a param called budget. In this master's thesis, we study a mechanism to improve the bandwidth and CPU usage performance choosing a value to budget. Results showed that budgets with high values have better performance than low values on Xen, VMware, VirtualBox not requiring changes in the budget's value default.

Keywords: cloud computing, network virtualization, interrupt coalescence

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A computação em nuvem refere-se tanto a aplicações fornecidas como serviços por meio da Internet como também a sistemas de *hardware* e *software* dos CPDs (Centro de Processamento de Dados) que fornecem os serviços [AFG+09].

Quando fornecemos uma infraestrutura computacional como serviço, recursos computacionais devem ser alocados a qualquer momento de acordo com os requisitos do usuário [GED+11]. Para que a infraestrutura suporte esse tipo de recurso, ela adota uma tecnologia chamada virtualização.

A virtualização divide um recurso com grande capacidade de processamento em recursos menores chamados de máquinas virtuais [BDF+03]. Com recursos menores, é possível fornecer ao consumidor uma quantidade menor de recursos computacionais que ainda satisfaçam seus requisitos e também alocar mais sob demanda [AFG+09].

Por exemplo, uma empresa de comércio eletrônico pode alugar uma quantidade x de processadores de uma máquina física na véspera do natal, afim de garantir que todos os clientes terão sucesso nas compras, e reduzir esse aluguel para $x/16$ após a data festiva, já que a quantidade esperada de acessos tende a cair. Apesar das máquinas virtuais ajudarem a aumentar a flexibilidade, elas ainda têm um desempenho abaixo das máquinas físicas quando são executadas aplicações que usam muito a rede [WCC+08] [EF10] [Liu10] [WR12] [Rix08].

Uma das principais causas é a forma com que o *hypervisor*, módulo que gerencia as máquinas virtuais, intermedeia a comunicação das máquinas virtuais com os dispositivos de rede. Diferentemente dos computadores físicos, as máquinas virtuais necessitam executar alguns passos adicionais para transmitir ou receber um pacote de informação que implicam em um custo adicional no processamento [WCC+08] [EF10] [Liu10] [WR12] [Rix08].

Uma proposta para reduzir a carga no processamento da rede é adotar estratégias que agreguem as interrupções geradas para a transmissão/recepção de pacotes [Sal07], [DXZL11]. Uma Estratégia simples é agregar as interrupções por intervalo de tempo ou por pacotes. Nela, o dispositivo deixa de gerar interrupções a cada pacote que chega ou é transmitido e passa a gerar uma única interrupção apenas depois de um intervalo de tempo que um pacote é recebido ou transmitido, ou quando uma quantidade de pacotes determinada é transmitida/recebida. Como apenas uma interrupção é gerada para um agrupamento de pacotes, isso reduz a quantidade de interrupções por pacote, porém, também pode atrasa o processamento de pacote já que cada pacote poderá ser processado somente quando a interrupção for gerada.

Atualmente, uma estratégia de agregação muito utilizada é a *NAPI* (*New API*) [CRKH05], um conjunto de interfaces oferecido pelo núcleo do *Linux* que os *drivers* dos dispositivos de rede implementam para agregar interrupções. Nela, quando uma grande quantidade de pacotes for enviada para o dispositivo de rede, ao invés do dispositivo enviar uma interrupção ao *driver* para cada pacote que chega, o *driver* desabilita as interrupções na chegada do primeiro pacote e processa continuamente os próximos pacotes. Esse processamento continuo dos pacotes é chamado varredura.

Atualmente a *NAPI* é a proposta mais implementada pelos *drivers* de dispositivos de rede para agregar interrupções, porém, há poucos estudos na área, principalmente quando consideramos

dispositivos virtuais. Neste trabalho de mestrado realizamos um estudo aprofundado da *NAPI* e identificamos quais parâmetros influenciam o desempenho em função da banda de rede e uso de *CPU* (do inglês *central processing unit* – unidade central de processamento).

1.1 Objetivos

A *NAPI* possui um parâmetro chamado limite. Esse parâmetro limita a quantidade de pacotes processada pelo sistema a cada ciclo de varredura, podendo controlar o momento em que o sistema deve iniciar uma varredura. O limite é estudado no artigo de Corbet [Cor05] e Salah [SEB05], porém, apenas considerando dispositivos físicos. Assim, o objetivo desse trabalho é elaborar um mecanismo que escolha o melhor valor para o parâmetro limite da *NAPI*, no qual o sistema tenha um bom desempenho tanto no uso de *CPU* como na largura de banda.

1.2 Contribuições

As principais contribuições desse trabalho são:

- Um estudo aprofundado sobre o comportamento dos dispositivos de rede virtuais usando *NAPI*. Atualmente, estudos foram feitos apenas com dispositivos de rede reais.
- Um mecanismo que define o melhor valor para o parâmetro limite da *NAPI*, considerando a largura de banda e uso de *CPU* para os seguintes *hypervisors*: *VirtualBox*, *Xen* e *Vmware*.

1.3 Organização do Trabalho

Este texto está organizado da seguinte forma: no Capítulo 2, são apresentados os conceitos de dispositivos de E/S, interrupções de *hardware* e *software*, virtualização de computadores, virtualização de rede, agregação de interrupções e computação em nuvem. No Capítulo 3 é apresentada uma revisão bibliográfica na área de virtualização de rede. No Capítulo **??** é apresentada a proposta de pesquisa. No Capítulo 4 são apresentados os experimentos com *NAPI*. Por fim, no Capítulo 5 são apresentadas as conclusões do trabalho.

Capítulo 2

Conceitos

Este trabalho avalia aspectos tanto de *hardware*, como de *software* de sistemas operacionais em ambientes virtualizados. A seguir, descreveremos os mecanismos de E/S (Entrada e Saída) de computadores que seguem o modelo de Von Neumann e, em seguida, mostraremos como isso se aplica em plataformas de computação virtualizada como as de computação em nuvem.

2.1 Arquitetura de E/S (Entrada e Saída) em Computadores

Um computador, segundo o modelo de Von Neumann [Sta10], é formado por uma memória principal, uma unidade central de processamento e dispositivos de E/S como mostra a Figura 2.1. Cada dispositivo de E/S do computador é controlado por um módulo para E/S. Este módulo de E/S é necessário para que o processador possa se comunicar com um ou mais dispositivos de E/S. Os dispositivos de E/S possuem vários métodos de operação, diferentes formatos, comprimento de palavras e velocidade de transferência, o que faz cada módulo ter uma lógica específica para um tipo de dispositivo.

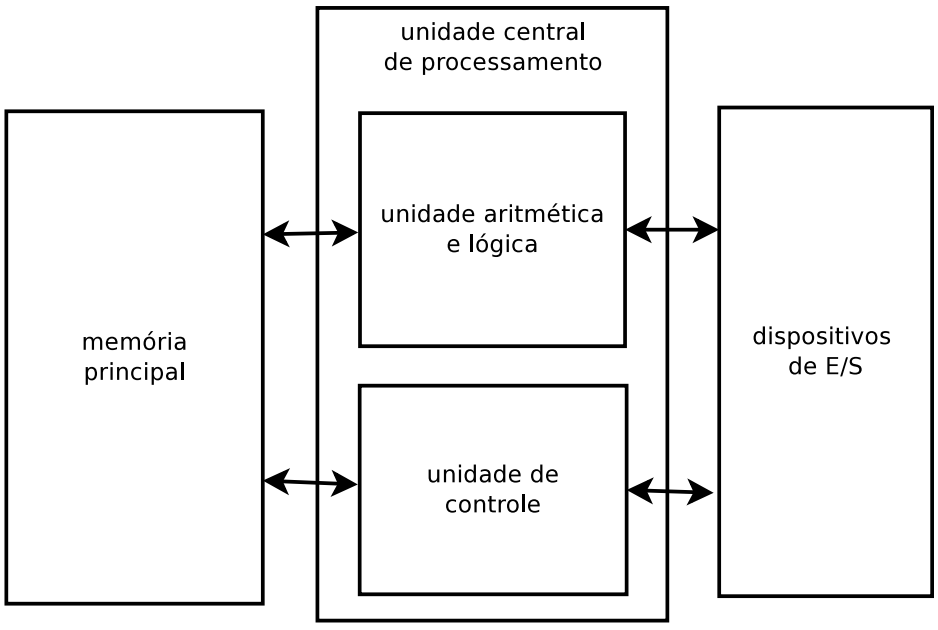

Figura 2.1: *Estrutura de um computador segundo o modelo de Von Neumann traduzida de [Sta10]*

Existem três técnicas possíveis para operações de E/S: E/S programada (varredura), interrupção dirigida à E/S e *DMA* (do inglês *direct memory access* – acesso direto a memória).

- Na E/S programada também chamada de varredura, dados são transferidos entre o processador e o módulo de E/S. O processador executa um programa e fornece a este, controle

direto das operações de E/S. Um problema com essa estratégia é o intervalo de tempo que o processador precisa esperar para o dispositivo de E/S estar pronto. Dentro desse intervalo, muitas instruções poderiam ser processadas;

- Na interrupção dirigida à E/S, um programa emite um comando requisitando E/S e continua executando outras instruções. O modulo E/S então gera uma interrupção de *hardware* quando o dispositivo estiver preparado, o processador então trata essa interrupção. Como o processador não espera ociosamente o dispositivo estar pronto, é possível processar uma quantidade de instruções maior quando algum dispositivo de E/S é acessado;
- No *DMA*, um módulo chamado *DMA* é incluído no sistema. O módulo é um processador especializado em E/S que recebe o controle das operações de E/S para mover um grande bloco de dados usando a memória principal. Quando o módulo conclui uma operação, uma interrupção de hardware é gerada para ser tratada pelo processador. Nota-se que o processador não participa ativamente nessa técnica como nas anteriores, o que reduz o custo de processamento em relação as outras.

Na Figura 2.2, é possível observar o fluxogramas das três técnicas sendo aplicadas para receber um bloco de dados de um dispositivo de E/S. Tanto na varredura como na interrupção dirigida a E/S, percebe-se que o processador participa de todo processo, enquanto que no *DMA*, como o dispositivo consegue acessar diretamente a memória, a *CPU* participa apenas na requisição de leitura e na recepção da interrupção do módulo de *DMA*, avisando que o bloco de dados foi copiado. Atualmente, a técnica de varredura é pouco usada, pois é desperdiçado muito tempo de processamento e sempre podem existir aplicações que necessitam de processamento. Já a interrupção dirigida à E/S é normalmente usada para dispositivos de E/S que transferem quantidades pequenas de informação e necessitam de pouco atraso como o teclado e o *mouse*. Por fim, o *DMA* é normalmente usado para dispositivos que transferem quantidades grandes de informações como o disco rígido e o leitor de CD/DVD.

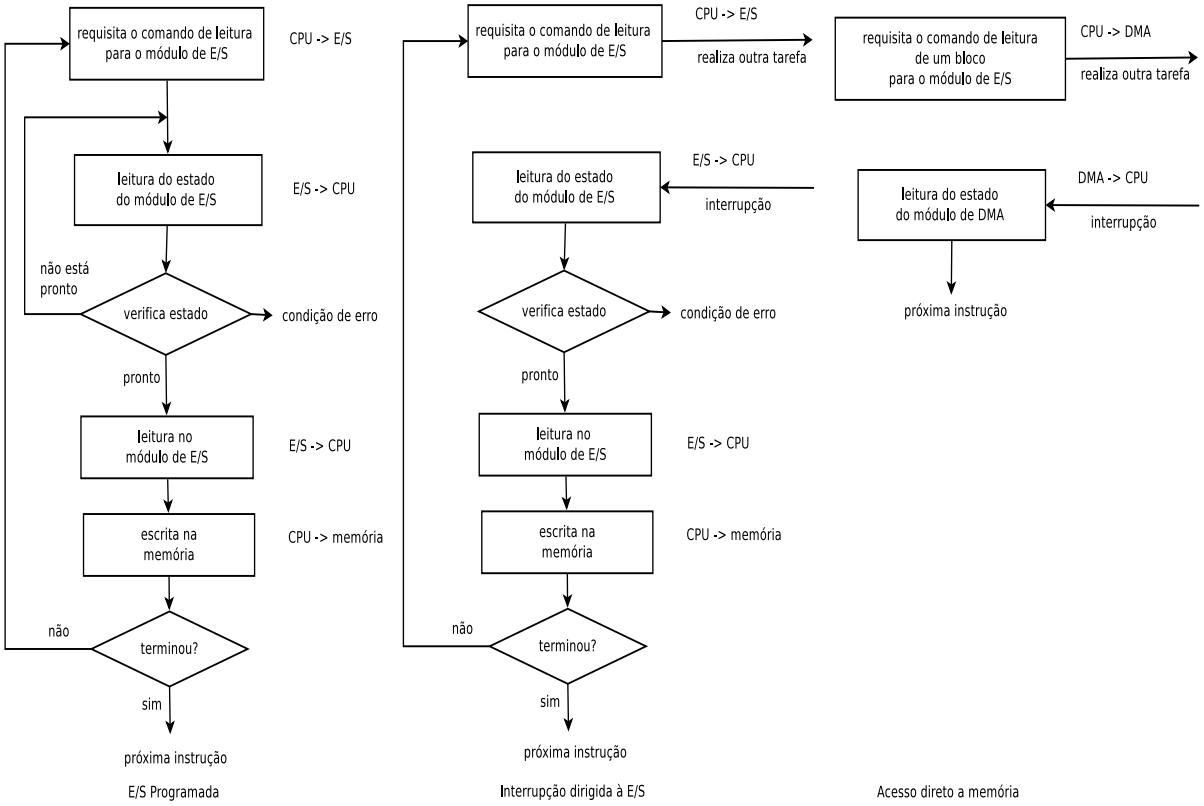

Figura 2.2: *Três técnicas para operações de E/S traduzida de [Sta10]*

2.2 Interrupções de Hardware X Interrupções de Software

As interrupções de *hardware* são como o próprio nome diz, interrupções geradas por dispositivos externos como o disco rígido, a placa de rede, mouse entre vários outros. Essas interrupções são usadas quando a estratégia para operações de E/S é a interrupção dirigida à E/S ou *DMA*. Quando geradas, devem ser tratadas pelo núcleo do sistema o mais rápido possível, ignorando interrupções durante o processo e sem ceder o tempo para outros processos ou seja, o tratamento das interrupções de *hardware* é não preemptivo e não reentrante. Nesse caso, se o tratamento for demorado, outros processos terão um atraso de execução por falta de espaço de tempo no processador. Já se o tratamento da interrupção entrar em um laço infinito, o sistema entrará em *livelock*, ou seja, o sistema não estará bloqueado, porém, não será capaz de prosseguir porque todo o processador estará sendo usado para o tratamento da interrupção [SEB05]. Como cada dispositivo transfere informações em taxas diferentes, alguns geram mais interrupções que outras. As redes gigabits por exemplo transferem com um taxa 10x maior que os discos rígidos e 1000x maior que os discos flexíveis como é possível ver na Figura 2.3 [Sta10].

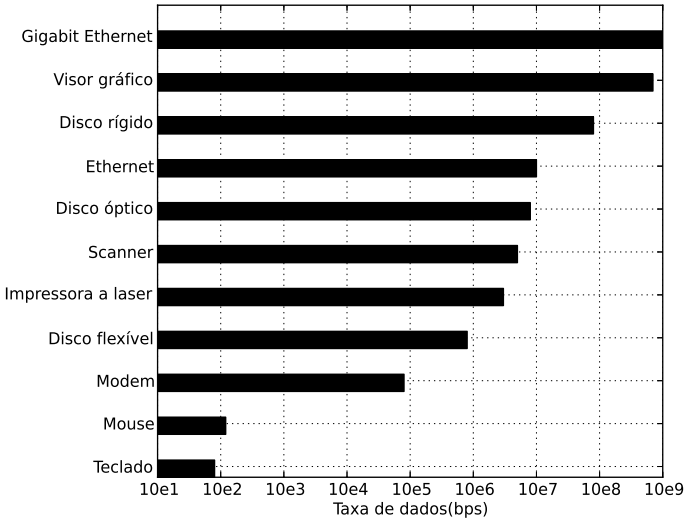

Figura 2.3: *Taxa de transferência para diferentes dispositivos de E/S [Sta10]*

Já as interrupções de *software* são semelhantes com as interrupções de hardware, porém, são geradas por software. Assim, qualquer software no espaço do núcleo do sistema é capaz de gerar interrupções de *software*. Elas são reentrantes, mas não são preemptivas, ou seja não podem ser interrompidas tanto por interrupções de *software* como de *hardware*, porém, podem ceder seu tempo para outros processos tornando-as mais flexíveis em relação ao tempo gasto no tratamento delas [Cor12]. O uso delas é normalmente reservado para processos do sistema que exigem tempo real e que são importantes. Também são usadas pelo *driver* no fim do tratamento de uma interrupção de *hardware* para reativá-la.

O *ksoftirqd* é um processo do núcleo do sistema que ajuda no processamento das interrupções de *software*. Quando muitas interrupções de *software* são geradas, o sistema pode sobrecarregar. Nessa situação, o *ksoftirqd* passa a processar essas interrupções, e a controlar a reativação delas. Caso algum núcleo do processador esteja desocupado, o *ksoftirqd* será rapidamente escalonado.

2.3 Computação em Nuvem

A computação em nuvem refere-se tanto a aplicações fornecidas como serviços por meio da Internet como também a sistemas de *hardware* e *software* dos CPDs (Centro de Processamento de

Dados) que fornecem os serviços [AFG+09].

Como o termo computação em nuvem é muito abrangente, ele foi dividido em várias classificações [AFG+09], entre elas, o tipo de serviço o qual fornece. Seguindo essa classificação existem as nuvens que fornecem *software* como serviço (SaaS – *Software as a Service*), plataformas como serviço (PaaS – *Platform as a Service*), e infraestruturas como serviço (IaaS – *Infrastructure as a Service*).

Exemplificando:

- *SaaS* oferecem aplicações como serviço para o cliente. Como exemplo temos o *Google Docs*[1] que fornece um *software* para edição de documentos como serviço;
- *PaaS* oferecem plataformas nas quais o cliente pode criar e implantar suas aplicações. O *Google App Engine*[2] e o *Windows Azure*[3] são exemplos desse tipo de nuvem;
- *IaaS* oferecem infraestruturas computacionais ao cliente. Um exemplo desse tipo de nuvem é o *Amazon EC2*[4] que fornece uma infraestrutura a qual emula um computador.

Nesse texto, iremos nos focar em fornecer infraestruturas, em específico, computadores como serviço (IaaS). Cada serviço pode receber várias requisições para hospedar programas de desenvolvedores e, nesse caso, terá que implantá-los na infraestrutura. Quando um cliente, em algum momento, faz uma requisição para executar esse programa, a nuvem executa o programa internamente e repassa o resultado ao cliente. Para que isso seja possível, a infraestrutura de nuvem contém vários nós, os quais são recursos físicos, como computadores ou mesmo CPDs inteiros, que contêm e controlam várias máquinas virtuais usando alguma técnica de virtualização. Cada requisição para implantar ou executar um programa é feita oferecendo as máquinas virtuais as quais estão contidas na infraestrutura.

2.4 Virtualização

Na computação em nuvem, em particular quando se é fornecida uma infraestrutura para implantar aplicações (*IaaS*), a adoção da virtualização melhora a utilização dos recursos e protege o computador de problemas que os software dos clientes possam causar em relação a computadores que não adotam virtualização [WCC+08]. Por exemplo, em um cenário com máquinas sem virtualização, um erro de programação que cause um laço infinito pode consumir toda a *CPU* do computador, atrapalhando todos os usuários daquela máquina. Em um cenário virtualizado, temos um isolamento entre os recursos das máquinas virtuais o qual impede uma máquina virtual de usar recursos de outra máquina. Assim, a única máquina afetada no cenário é aquela utilizada pelo programador. Além disso, não é possível expandir a quantidade dos recursos sem permissão do administrador da nuvem, dando mais segurança na infraestrutura em relação a infraestruturas sem virtualização. Além da segurança, outra consequência da virtualização, é o surgimento de um novo modelo de negócio chamado "pague somente quando usa", onde o cliente paga somente pelo tempo que o recurso é usado. Além disso, o cliente tem a impressão de estar utilizando um ambiente com recursos infinitos, já que podemos aumentar os recursos de uma máquina virtual sem interrupção do serviço e mais máquinas podem ser agregadas para prover o serviço [AFG+09].

Essas características beneficiam o lado do servidor, que não precisará fornecer um recurso físico inteiro para cada cliente e terá maior segurança e tolerância a falhas, já que cada sistema é independente. Do lado do cliente, ele irá economizar dinheiro pelo novo modelo de negócio e terá recursos sob demanda.

Nos dispositivos de E/S, a virtualização permite a emulação de *hardware*. Em relação a flexibilidade, é possível mapear os dispositivos lógicos com as implementações físicas, garantindo uma

[1] http://docs.google.com
[2] http://developers.google.com/appengine/
[3] http://www.windowsazure.com/en-us/
[4] http://aws.amazon.com/ec2/

maior portabilidade. Esse mapeamento pode também trazer novas funcionalidades ao recurso como: balanceamento da carga de trabalho e mascaramento das falhas.

2.4.1 Virtualização de Computadores

As nuvens normalmente são constituídas de computadores que estão ligados de alguma forma por uma rede. A virtualização de computadores divide um computador, geralmente com grande capacidade de processamento, em recursos menores chamadas de máquinas virtuais de modo que cada uma age como se fosse um computador separado, podendo ter inclusive, diferentes sistemas operacionais [BDF+03]. Segundo Walters, Chaudhary, Cha, Jr. e Gallo [WCC+08], as estratégias de virtualização podem ser divididas em 4 grandes categorias: virtualização completa, para-virtualização, virtualização em nível de sistema operacional e virtualização nativa.

Um *hypervisor* fornece uma plataforma para as máquinas virtuais e gerencia a execução delas. No *hypervisor* da virtualização completa, é feita a interceptação, tradução e execução das instruções sob demanda dos sistemas operacionais das máquinas virtuais. Nessa estratégia, o núcleo do sistema operacional que executa o *hypervisor* não necessita de modificações. Dentro dessa categoria de *hypervisors* estão o *KVM*[1], o *Xen*[2], o *VMWare*[3] e o *VirtualBox*[4].

A para-virtualização também utiliza um *hypervisor* para gerenciar as máquinas virtuais, porém, ao invés desse ser executado num sistema operacional como na virtualização completa, ele é implantado diretamente no *hardware* da máquina exigindo que o núcleo do sistema operacional da máquina física seja modificado. Assim, caso não exista o código-fonte do sistema, não é possível usar essa estratégia. Na para-virtualização, o hardware virtual consegue conversar diretamente com o dispositivo emulado. Isso garante uma sobrecarga mínima em relação a tentar emular o dispositivo real. Nessa categoria estão incluídos o *Xen* e o *VMWare*.

A virtualização em nível de sistema operacional não utiliza um *hypervisor*. Ela modifica o núcleo do sistema isolando múltiplas instâncias do sistema operacional dentro de uma mesma máquina física. Nesse caso, como é feito apenas um isolamento entre as instâncias, estas ficam limitadas a usarem o mesmo sistema operacional. Uma vantagem e ao mesmo tempo desvantagem dessa categoria é o compartilhamento do núcleo entre as instâncias do sistema operacional. O compartilhamento do núcleo simplifica os mecanismos de compartilhamento de memória e o acesso a dispositivos físicos que em outras virtualizações necessitam de implementar um canal de comunicação entre núcleos adicionando um carga extra no sistema. Por outro lado, como todas instâncias dependem de um único núcleo, este se torna um ponto único de falha. Está incluído nessa categoria o *OpenVZ*[5].

Por fim, a virtualização nativa é uma virtualização completa "melhorada". Ela aproveita o *hardware* do processador que implementa mecanismos para otimizar a virtualização. Isto permite que múltiplos sistemas operacionais rodem sobre outro, sendo capazes de cada um acessar diretamente o processador do hospedeiro. Como exemplos temos o *Xen*, *KVM*, o *VMWare* e o *VirtualBox*.

As virtualizações completa e nativa têm uma grande vantagem em relação às outras: não é necessário alterar o núcleo do sistema operacional da máquina hospedeira. Isto as tornam mais simples e mais portáveis já que sistemas operacionais com código fechados podem ser utilizados. A para-virtualização e a virtualização em nível de sistema operacional exigem uma modificação no núcleo da máquina hospedeira, porém, são as que tem um melhor desempenho pois elas têm acesso ao hardware físico. Comparando as duas, a virtualização em nível de sistema operacional é bem mais intrusiva e não permite a mudança do sistema operacional das máquinas virtuais [PZW+07] [WCC+08] [SBdSC] [CSYL10].

[1] http://www.linux-kvm.org/
[2] http://xen.org/
[3] http://www.vmware.com/
[4] http://www.virtualbox.org/
[5] http://wiki.openvz.org/

A história do XEN

O projeto *Xen* foi criado na Universidade de *Cambridge* com o objetivo de aprimorar a técnica de virtualização completa através da para-virtualização, permitindo que a máquina virtual tenha conhecimento e acesso ao hardware da máquina física. Assim, foi desenvolvida uma nova interface de virtualização com a ajuda de pesquisadores da *Intel* e *Microsoft* usando o *Linux* e o *Windows XP* [BDF+03]. O *Xen* para *Windows* nunca saiu dos laboratórios, entretanto, para *Linux* é mantido e atualizado até hoje por uma comunidade global de desenvolvedores [1].

Enquanto a equipe do *Xen* focava na para-virtualização, os engenheiros da *Intel* e *AMD* trabalhavam para aperfeiçoar a virtualização completa [NSL+06] [AMD08]. O resultado é o que hoje chamamos de virtualização nativa. Nela, extensões para o processador foram criadas para simplificar a virtualização do processador. O *Xen* acompanhou e passou a manter tanto máquinas com para-virtualização como com virtualização nativa [Ros05].

Com a chegada dos processadores de 64 bits, a para-virtualização passa até o momento a ter problemas no desempenho. Isso ocorre porque os processadores de 64 bits não conseguem manter no mesmo espaço de endereço a memória do usuário, do núcleo de sistema e do núcleo de sistema do hóspede. Essa divisão era feita com páginas de memória (separando o usuário do núcleo do sistema) e com limite de segmentação (separando o núcleo do sistema do hóspede dos outros dois). Porém, como nenhum sistema operacional usava limite de segmentação, ele foi removido dos processadores de 64 bits [Dun12].

Atualmente, na para-virtualização, o *Xen* separa o espaço de endereço entre o núcleo do hóspede, o núcleo de sistema e o usuário, fazendo com que cada chamada de sistema do hóspede faça o *Xen* trocar de contexto para o espaço de endereço do hóspede reduzindo o acerto do cache e, como consequência, adicionando uma carga extra ao processador. Na virtualização nativa, isso não ocorre pois as extensões do processador para virtualização nativa facilitam a divisão. Isso faz a máquina com virtualização nativa tão rápida quanto a máquina pura.

Apesar da virtualização nativa ter um bom desempenho, as interrupções, os contadores, *boot* legado e a virtualização de rede, de disco e de placa-mãe são melhores na para-virtualização. Assim, sugiram técnicas de virtualizações híbridas misturando técnicas da virtualização nativa e para-virtualização [Dun12]. A virtualização nativa com *driver* para-virtualizado (*PV-on-HVM*), como o próprio nome diz, é a virtualização nativa usando *drivers* de rede e de disco para-virtualizados. *PVHVM* é um aperfeiçoamento do *PV-on-HVM*, além dos *drivers*, as interrupções e contadores são para-virtualizados. Por fim, *PVH* será um futuro aprimoramento da para-virtualização usando instruções privilegiadas e tabela de páginas da virtualização nativas.

2.4.2 Virtualização de Dispositivos de E/S

Os dispositivos de E/S são instrumentos que recebem ou enviam dados para o sistema computacional como o *mouse*, o teclado e o monitor. Quando falamos em dispositivos físicos nos referimos ao dispositivo de E/S como *hardware*, enquanto que dispositivos lógicos se referem ao dispositivo em forma lógica. Com a virtualização de computadores, os dispositivos de E/S sofreram modificações já que em um computador não há apenas um único sistema operacional, mas sim, várias máquinas virtuais com um sistema dentro de cada uma.

No trabalho de Rixner [Rix08], foi separada a virtualização de E/S em duas categorias: privada ou compartilhada. Na virtualização de E/S privada, cada dispositivo físico é associado a apenas uma única máquina virtual enquanto que na virtualização de E/S compartilhada, o dispositivo é compartilhado por várias máquina virtual.

Comparando a virtualização de E/S privada com a compartilhada há uma subutilização na virtualização privada, pois enquanto uma máquina virtual não utiliza o dispositivo, outra poderia necessitar do seu uso. Por outro lado, o desempenho da virtualização compartilhada é pior já que divide o recurso com outras máquinas. Quando pensamos em aumentar o número de máquinas virtuais, o custo da virtualização privada cresce absurdamente (com 10 máquinas virtuais teríamos

[1] http://xen.org/

que ter 10 dispositivos físicos enquanto que na virtualização compartilhada, talvez até um dispositivo poderia ser o suficiente para resolver o problema).

Normalmente, a opção mais comum é que o dispositivo físico seja compartilhado entre as máquinas, tanto pela possibilidade de escalar como pelo custo. Porém, disponibilizar de maneira compartilhada o acesso a dispositivos físicos pode trazer muitos problemas de segurança e dificultar o monitoramento das informações [STJP08]. Problemas de segurança surgem porque o usuário de uma máquina virtual pode tentar o acesso a uma outra máquina virtual justamente através do recurso que está sendo compartilhado [Rix08]. O monitoramento é dificultado porque ferramentas comuns só fazem medições do dispositivo físico e como este está associado a várias máquinas virtuais, ficaria complexo separar as informações específicas de cada uma dentro do agregado [GED+11].

Para lidar com o problema de segurança causado pelo compartilhamento do dispositivo, na virtualização em nível de sistema operacional como o *OpenVZ*, o núcleo do sistema gerencia a utilização do dispositivos entre as máquinas virtuais como mostra a Figura 2.4. Já em *hypervisors* como *XEN*, *KVM* e *VMWare*, como cada máquina possui seu próprio núcleo, é muito complexo fazer o conjunto de núcleos gerenciar o dispositivo entre as máquinas. A solução foi restringir o acesso ao dispositivo físico para apenas uma máquina virtual ou para máquina hospedeira e o acesso a esse dispositivo pelas outras máquinas virtuais é feito através dessa máquina. Como é possível ver na Figura 2.5, a máquina virtual 0 gerencia o dispositivo de E/S enquanto que as máquinas virtuais 1 e 2 acessam o dispositivo se comunicando com o núcleo do sistema 0. Essa restrição traz uma perda de desempenho em relação a ambientes que não usam virtualização quando o uso da rede é intensa, porém, garante segurança já que é possível monitorar o tráfego de todas as máquinas através da máquina que gerencia o dispositivo [WCC+08] [EF10] [Liu10].
Em relação a monitoração, cada *hypervisor* implementa seu próprio programa para monitorar os recursos de cada máquina virtual.

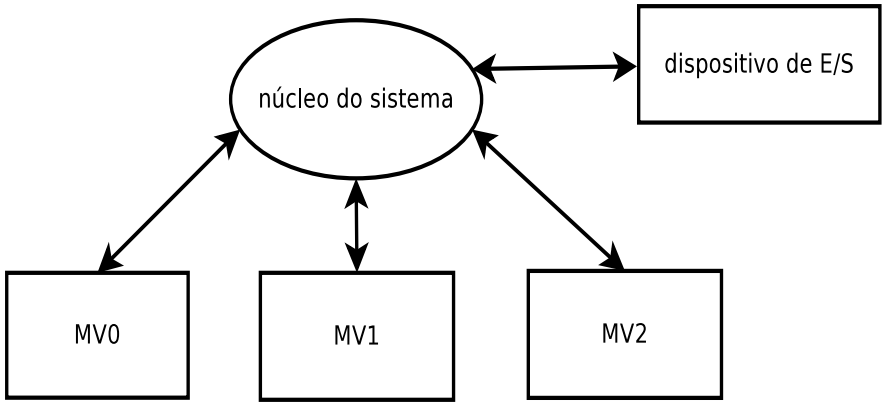

Figura 2.4: *Compartilhamento de um dispositivo de E/S na virtualização em nível de sistema operacional*

No trabalho de Waldspurger e Rosenblum [WR12], foram feitas algumas menções sobre o uso de técnicas de virtualização de E/S que desacoplam o dispositivo físico da sua implementação lógica. Dentre as vantagens, ele cita a melhor utilização dos recursos e a economia de custos em relação a sistemas que estão com o dispositivo físico acoplado com a sua implementação lógica, pois vários sistemas podem aproveitar o mesmo recurso. Em relação a flexibilidade, é possível mapear os dispositivos físicos com as implementações lógicas, garantindo uma maior portabilidade. Esse mapeamento pode também trazer novas funcionalidades ao recurso como: balanceamento da carga de trabalho e mascaramento das falhas. A funcionalidade de suspender, migrar e continuar uma máquina virtual também é possível, pois com o dispositivo físico desacoplado da implementação lógica, é possível reconectar a máquina virtual em outra máquina física com uma configuração diferente. Outra funcionalidade trazida com a virtualização é a interposição e transformação das requisições virtuais de E/S. Isso permite que as requisições que passam pelo dispositivo lógico sejam transformadas. Em um exemplo de leitura e escrita no disco, além de simplesmente ler/escrever no

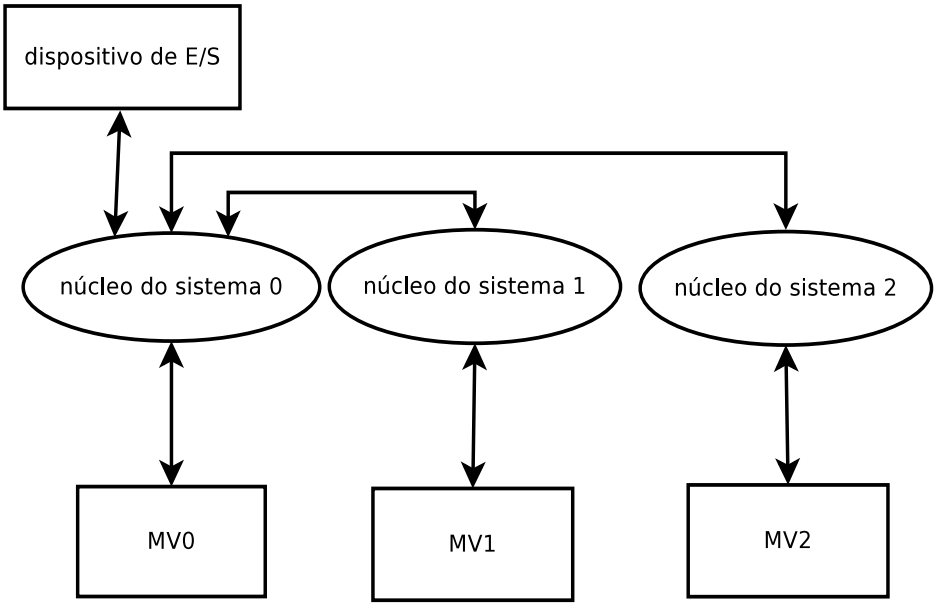

Figura 2.5: *Compartilhamento de um dispositivo de E/S na virtualização que utiliza hypervisors*

disco, torna-se possível guardar uma cópia da informação antiga como cópia de segurança. Outra possibilidade é criptografar a informação quando alguém escrever no disco, dificultando outras pessoas de acessarem o seu conteúdo escrito.

2.4.3 Virtualização do Dispositivo de Rede

A virtualização do dispositivo de rede, que também é um dispositivo de E/S, tem algumas particularidades em relação a outros dispositivos. Segundo Rixner [Rix08], a complexidade de virtualizar a rede é muito maior pelo fato de muitas vezes não se conhecer o destino de uma informação, pois esse está fora do sistema, diferente por exemplo do disco rígido que só se comunica com o sistema. Outra dificuldade é necessidade de estar preparada a qualquer momento para receber e responder ao tráfego da rede, diferentemente da virtualização de disco em que a leitura e escrita só ocorre quando requisitada pela máquina virtual.

Outra diferença é a taxa de transferência em relação a outros dispositivos de E/S, as redes gigabits comuns transferem com um taxa 10x maior que os discos rígidos e 1000x maior que os discos flexíveis como é possível ver na Figura 2.3 [Sta10].

Virtualização da Rede no XEN

Apesar de estarmos descrevendo arquitetura de virtualização de rede específica do *XEN*, esta é utilizada por outros *hypervisors* como o *KVM* e o *VMWare* [STJP08]. O *XEN* é um *hypervisor* de código aberto disponível para arquiteturas de máquina física x86, x86_64, IA64, ARM. Ele permite a virtualização nativa e para-virtualizada de sistemas operacionais *Windows*, *Linux*, *Solaris* e diversos outros sistemas baseados no BSD [Spe10].

No *XEN*, o *dom0* ou domínio zero é a primeira máquina virtual iniciada. Ela tem certos privilégios que as outras máquinas virtuais não têm, como iniciar novas máquinas e acessar o hardware diretamente. Os `domUs` ou domínios do usuário são máquinas virtuais que, por padrão, não tem alguns privilégios que o `dom0` tem como o acesso direto ao *hardware*. Assim, é necessário um mecanismo para conseguir acessar o dispositivo de rede [Spe10].

No *XEN*, para todas as máquinas conseguirem acessar o dispositivo de rede ao mesmo tempo, existem dois tipos de configuração: ponte (*bridge*) e roteador. Ambas as configurações seguem os conceitos dos equipamentos de mesmo nome que existem na interconexão de redes de computadores. Todos os dois tipos encaminham pacotes entre domínios baseados nas informações que os próprios

pacotes contêm, porém a ponte se fundamenta nos dados da camada de enlace enquanto que o roteador se fundamenta nos dados da camada de rede [BM99]. Podendo trafegar pacotes entre domínios, os `domUs` conseguem enviar e receber pacotes do dispositivo de rede com o `dom0` como intermediário.

No artigo de Eastep [Eas07], foi descrita a implementação da configuração de ponte na qual uma ponte virtual (`xenbr0`) é criada dentro do `dom0` como é possível ver na Figura 2.6. Essa ponte está ligada na interface de rede física `peth0`. A interface `vif0.0` está sendo usada para tráfegos de/para `dom0` e as interfaces `vifX.0`, onde X é um valor maior que 0, estão sendo usadas para tráfegos de/para algum `domU`. Como é possível observar, todo pacote que é recebido ou transmitido para alguma máquina virtual tem que passa pela ponte dentro do `dom0`. A configuração de roteador é muito semelhante à configuração da ponte, porém, ao invés de existir uma ponte virtual, o *dom0* possui um roteador virtual que é configurado para encaminhar pacotes *IP* (do inglês *Internet Protocol* – Protocolo Internet) entre os domínios e os *domUs*.

No experimento de James [Jam04], foi comparado a ponte virtual e o roteador virtual. Os resultados foram semelhantes tanto na largura de banda como na latência e no uso do processador. Nessa pesquisa focaremos na configuração de ponte devido ao fato dessa configuração trabalhar numa camada de rede mais baixa.

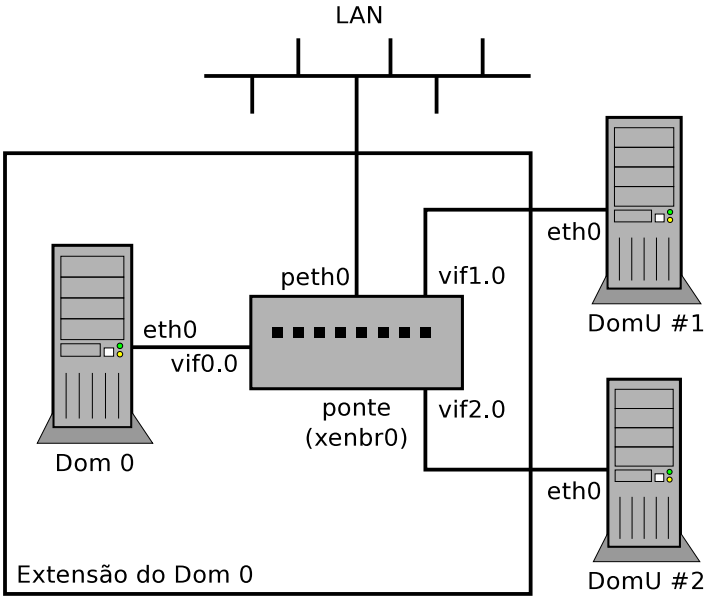

Figura 2.6: *Ponte virtual criada no XEN traduzida de [Eas07]*

Na Figura 2.7 vemos a arquitetura da virtualização da rede usando ponte no *XEN* [STJP08]. Para transmitir/receber um pacote no `domU` é usado o canal de E/S (*I/O channel*). Esse canal evita que cada pacote tenha que ser copiado de um domínio a outro. Para tal, o `domU` compartilha algumas páginas de sua memória e informa a referência delas por esse canal para o outro domínio mapeá-las em seu espaço de endereço. Quando algum domínio envia algum pacote para essas páginas, uma notificação é enviada para o outro domínio.

O canal de E/S consiste de notificações de evento e um *buffer* de descrição em anel. A notificação de evento avisa que algum usuário do domínio deseja enviar informações. O *buffer* de descrição em anel guarda os detalhes de requisições entre o *driver* de *frontend* (*netfront*) que fica no interior do *dom0* e o *driver* de *backend* (*netback*) que fica dentro de um `domU`.

Para o *dom0* ter acesso às páginas da memória do *domU* é necessário um mecanismo de permissão. Neste, o *domU* fornece páginas vazias da sua memória para serem usadas como *buffer* de E/S. Essas páginas são passadas como referência na descrição da requisição.

Na transmissão de pacotes, o `domU` coloca o pacote no *buffer* de E/S, as referências de suas páginas de memória no *buffer* de descrição e notifica o *dom0* através de uma interrupção. O *dom0*

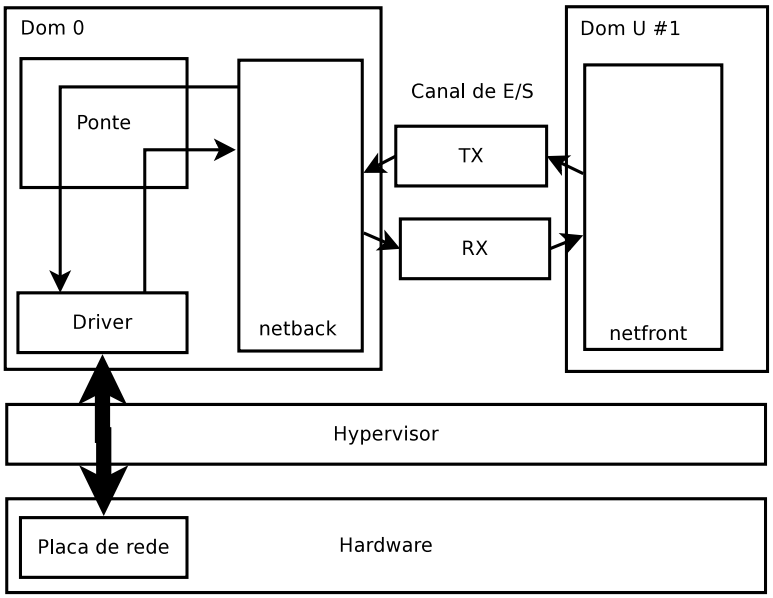

Figura 2.7: *Arquitetura da rede virtual no XEN simplificada e traduzida de [STJP08]*

por sua vez, lê o *buffer* de descrição, mapeia as páginas recebidas no seu espaço de endereços e pede para transmiti-las através da ponte. Quando o dispositivo físico confirmar a transmissão, o `domU` libera as páginas do *buffer* de E/S.

Na recepção, o *netfront* informa as possíveis páginas da memória que podem ser usadas como *buffer* de E/S ao *netback*. Quando algum pacote chega pelo dispositivo físico, este envia uma interrupção de chegada de pacote à ponte dentro do *dom0*. A ponte então avisa o *netback* correto sobre a chegada de pacotes. O `netback` o copia para uma página da memória que foi fornecida pelo *netfront* e envia uma interrupção para o mesmo. Quando o `netfront` recebe a interrupção, ele pega o conteúdo que está no *buffer*, envia para o seu sistema e libera as páginas fornecidas.

2.5 Agregação de Interrupções

Quando o tráfego de pacotes possui uma taxa de transmissão da ordem de Gbps no meio físico, a quantidade de interrupções devido a chegada de pacotes é muito grande podendo sobrecarregar o processamento [DXZL11]. Isso ocorre porque as interrupções têm prioridade absoluta sobre todas as outras tarefas e se a taxa de interrupções é suficientemente elevada, o sistema gastará todo seu tempo para respondê-la e o rendimento do sistema cairá para zero [Sal07].

A agregação de interrupções por intervalo de tempo ou número de pacotes é uma proposta da literatura para resolver esse problema [Sal07]. Ela pode ser implementada no *driver* do dispositivo e usada através do programa *ethtool*. O seu mecanismo reduz a quantidade de interrupções na transmissão/recepção de pacotes dentro de um intervalo de tempo ou número de pacotes em troca de aumentar a latência da rede. Para configurá-lo, existem 4 parâmetros: `tx-frames`, `rx-frames`, `tx-usecs` e `rx-usecs` (a descrição de cada parâmetro está na tabela 2.1).

Como pode-se notar na Tabela 2.1, a agregação de interrupções depende do tamanho do *buffer* de transmissão e recepção. O *buffer* pode ser tanto um espaço de memória da máquina (*DMA*) como uma memória interna da placa de rede. Caso este seja pequeno, vários pacotes serão descartados durante o tráfego de pacotes por falta de espaço, caso seja grande, pode aumentar a latência por ter muitos pacotes esperando serem lidos dentro dele. Um problema com essa proposta é que nem sempre a placa de rede virtual ou física implementa essa funcionalidade como o e1000 da Intel, o dispositivo de rede do *XEN* e o *Virtio* que são mais usados pelos *hypervisors*.

Tabela 2.1: *Parâmetros para agregação de interrupções*

nome	descrição
`tx-frame` N	gera uma interrupção quando a quantidade de pacotes transmitida chegar a N
`rx-frame` N	gera uma interrupção quando a quantidade de pacotes dentro do buffer de recepção chegar a N
`tx-usecs` N	gera uma interrupção N microssegundos depois que um pacote for transmitido
`rx-usecs` N	gera uma interrupção N microssegundos depois que um pacote for recebido

2.5.1 NAPI

A *NAPI* (*New API*) [CRKH05] é um conjunto de interfaces oferecido pelo núcleo do Linux que os *drivers* dos dispositivos de rede implementam para agregar interrupções. O objetivo dela é reduzir a carga extra do processamento na recepção de uma grande quantidade de pacotes em um ou mais dispositivos de rede. Para isso, no momento que uma grande quantidade de pacotes for enviada para o dispositivo de rede, ao invés do dispositivo enviar uma interrupção de *hardware* ao *driver* para cada pacote que chega, o *driver* desabilita a interrupção na chegada do primeiro pacote e processa continuamente os próximos pacotes numa estratégia de varredura.

No processo dos pacotes, o *driver* envia uma tarefa de recepção de pacotes na fila de varredura do núcleo do sistema[Cor05]. Para cada tarefa da fila de varredura, é gerada uma interrupção de *software* para o sistema processar a tarefa da fila. A quantidade de pacotes que essa tarefa poderá processar é definida por uma variável peso. Quanto maior o peso, mais pacotes poderão ser processados, mas não existe uma relação clara entre os pacotes e o peso, pois cada *driver* faz uma implementação diferente da recepção de pacotes. Durante o processo de recepção, é verificada a quantidade de pacotes recebida, se uma estimada quantidade não for recebida, a tarefa é retirada da fila de varredura, a interrupção de *hardware* é reativada e será necessária a chegada de outro pacote para a tarefa ser devolvida para a fila, já se essa quantidade é processada, a tarefa é encerrada e recolocada na fila [Cor05]. Caso o sistema fique sobrecarregado e não seja possível processar mais pacotes, o *driver* descarta os pacotes.

Comparando o uso da *NAPI* com a estratégia comum de interrupção dirigida à E/S e *DMA* e a estratégia de agregação de interrupções por intervalo de tempo e por pacote, temos prós e contras.

Prós do uso da *NAPI*:

- A quantidade reduzida de interrupções de *hardware* gerada pelo dispositivo, já que a interrupção é desabilitada durante o processo de pacotes;
- O processador fica menos ocupado tratando uma interrupção de *hardware* e, assim, reduz a chance do sistema entrar em *livelock*; Nesse caso é possível ainda, devido as interrupções de *software*, sobrecarregar a *CPU*, porém, o núcleo do sistema é capaz de controlar o tratamento delas através do `ksoftirqd`;
- Simplicidade, diferente da estratégia de agregação por intervalo de tempo e pacotes que necessita de ajustes manuais na configuração de acordo com o tipo de tráfego, a *NAPI* se ajusta ao tráfego, agregando mais interrupções quanto maior o tráfego de pacotes;
- A maioria dos *drivers* de dispositivos de rede virtuais atuais implementam *NAPI*. Foram vistos o *Virtio*, o driver do *Xen* e o e1000 da *Intel*.

Contras:

- Existe uma carga adicional com *NAPI*, já que não é gerada apenas uma interrupção de *hardware*, como também é gerada uma interrupção de *software*;
- O *driver* precisa implementar *NAPI*.

O peso, normalmente, na implementação do *driver* é a quantidade limite de pacotes que o *driver* consegue processar em cada ciclo de varredura, por isso nos experimentos iremos chamá-lo de limite.

2.6 Agregação de Interrupções na Transmissão x Recepção

Tanto a transmissão quanto a recepção de pacotes podem gerar interrupções com uma frequência grande [MCZ06]. A transmissão gera uma interrupção quando um pacote é transmitido com sucesso e a recepção gera uma interrupção quando um pacote é recebido [CRKH05]. A diferença entre elas é que enquanto a transmissão pode controlar os pacotes que são enviados pelo sistema, a recepção não consegue controlar os pacotes que chegam. Assim, na transmissão podemos reduzir de outras formas a quantidade de interrupções. Uma das principais propostas da literatura é o *GSO* (do inglês *generic segmentation offload*) [Cor09]. O *GSO* permite ao *driver* de rede segmentar os pacotes, uma tarefa que normalmente é feita pelo processador.

Atualmente, o tamanho do pacote é limitado pela *MTU* (do inglês *maximum transmission unit* – unidade máxima de transmissão). No protocolo *Ethernet* ela tem como valor padrão 1500 bytes. Esse valor acabou sendo adotado na época do crescimento da Internet pelos limites de *hardware* da época e infelizmente continua até hoje. Assim, não é possível enviar pacotes maiores que 1500 bytes pela Internet, o que força o sistema operacional a segmentar seus dados em pacotes pequenos para conseguir enviá-los. Isso sobrecarrega o processador tanto para segmentar os dados, como para enviar e receber esses pacotes.

Com a segmentação sendo feita apenas no momento da transmissão dos pacotes pelo *GSO*, pode-se configurar o *MTU* da interface de rede do sistema acima do limite do dispositivo físico. Com um *MTU* maior, o pacote é segmentado em pedaços grandes e em menor quantidade quando o sistema manda transmiti-lo. Com menos pacotes, a quantidade de interrupções por pacote é reduzida. Na recepção, o *LRO* (do inglês *large receive offload*) e o *GRO* (do inglês *generic receive offload*) [Cor09] são soluções baseadas no *GSO*, onde os pacotes são montados quando recebidos. O *LRO* monta cada pacote agregando os pacotes *TCP* que chegam, porém, se por exemplo, existir uma diferença nos cabeçalhos do pacote *TCP*, haverá perdas na montagem, pois o pacote será montado sem considerar essa diferença. Já o *GRO*, restringe a montagem dos pacotes pelos cabeçalhos, o que não gera perdas e, além disso, o *GRO* não é limitado ao protocolo *TCP*. Apesar da proposta permitir a montagem de pacotes, como já foi dito, não é possível controlar a chegada de pacotes, o que força a adoção de alguma técnica de agregação de interrupções como a *NAPI* para conseguir montar os pacotes.

Todas as otimizações dessa seção já foram implementadas nos *drivers* atuais. É possível configurá-las diretamente no código-fonte do *driver* ou pela ferramenta `ethtool` do *Linux*.

Capítulo 3

Revisão Bibliográfica

3.1 Objetivo

O objetivo dessa revisão foi analisar e estudar os problemas existentes na área de virtualização de dispositivos de rede. Também foi estudadas as estratégias de agregar interrupções tanto em dispositivos de redes virtuais como físicas.

3.1.1 Resumo Sintetizado

No artigo de Walters, Chaudhary, Cha, Jr. e Gallo [WCC+08], foi feito uma comparação entre o *XEN*, *VMWare* e *OpenVZ*. A partir dos experimentos foi concluído que o *hypervisor XEN* tem um desempenho baixo em termos de latência, porém alto em termos de largura de banda em relação a um ambiente com *OpenVZ* e um ambiente sem virtualização, enquanto que o *OpenVZ* tem uma perda em largura de banda, mas um atraso pequeno. Quanto ao *VMWare*, ele teve um desempenho baixo tanto em atraso quanto em largura de banda. Os autores não entram em detalhes sobre os motivos dos resultados terem sido esses.

O artigo de Ekanayake e Fox [EF10] estudou a relação entre o número de núcleos e o número de máquinas virtuais usando *XEN* e *Eucalyptus* como infraestrutura de nuvem. Foi concluído que a virtualização funciona bem para aplicações que não se comunicam muito, enquanto que em aplicações que são sensíveis a latência, houve uma perda de desempenho em relação a um ambiente não virtualizado. Outra conclusão foi que quanto maior o número de máquinas virtuais, maior a sobrecarga na *CPU*. A explicação para isso, segundo o autor, está na forma como é implementada a virtualização da rede. O hardware físico só pode ser controlado por um sistema (`dom0`), enquanto que os outros (`domUs`) para conseguirem fazer alguma operação de E/S pela rede, devem passar por esse sistema através de um canal. Isso forma um gargalo no `dom0`.

No artigo de Rixner [Rix08], foi feita uma revisão sobre a virtualização de rede. No texto o autor cita que a virtualização de rede impacta diretamente no número de servidores que podem ser diretamente consolidados dentro de uma única máquina física. Porém, as técnicas modernas de virtualização têm gargalos significantes, o que limita o desempenho da rede. Ele sugere um ganho de desempenho fazendo o dispositivo ter a capacidade de ler e escrever diretamente na memória da máquina virtual ao invés do processador da máquina virtual gerar interrupções cada vez que alguma informação entra ou sai pelo dispositivo. Essa funcionalidade é chamada acesso direto a memória (*DMA*). Apesar disso, o dispositivo pode escrever em uma posição da memória que não pertence a máquina virtual, podendo assim, causar problemas em outros processos da máquina física. Assim, foi criado a unidade de gerenciamento de E/S da memória (*IOMMU – input/output memory management unit*). No *IOMMU* a memória é restrita para o dispositivo de acordo com a máquina virtual que controla esse dispositivo. Atualmente os *hypervisors* modernos, como o *XEN*, utilizam essas técnicas [BDF+03] .

Como atualmente um processador possui vários núcleos, pode-se aproveitar esses núcleos para criar multi-filas nas interfaces de rede. O autor cita que pesquisadores do laboratório da HP e *Citrix* eliminaram a ponte no domínio de E/S para associar as máquinas virtuais diretamente com o *driver* de *backend* através das multi-filas, evitando a necessidade de sincronização das mensagens e multiplexação/demultiplexação da rede. Como benefícios do uso da multi-fila se teve: a redução da carga extra na fila e a eliminação de cópias entre o domínio de E/S e a máquina virtual, pois, a multiplexação não é feita. Por outro lado, é necessário que cada informação seja enviada para a fila correta e que a *CPU* consiga aguentar a carga extra gerada pelas múltiplas filas.

Ainda no mesmo artigo, na arquitetura de virtualização de rede CDNA (do inglês *concurrent direct network access* – acesso direto a rede concorrente) foi empregada a proposta de multi-filas, em adição foi removido o domínio de E/S. Sem o responsável por controlar as filas, o *hypervisor* passa a considerar cada conjunto de fila como uma interface de rede física e associa o controlador a uma máquina virtual. Assim, cada máquina virtual consegue enviar ou receber informações diretamente da rede sem nenhuma intervenção do domínio de E/S. Como consequência, a carga extra é reduzida pelo número reduzido de interrupções (antes era necessário interromper tanto o domínio de E/S como as máquinas virtuais em cada transmissão/recepção). Pela máquina virtual poder acessar diretamente a interface de rede, ela também pode acessar algum local indevido da memória por *DMA*. Para contornar esse problema o autor sugeriu o uso de *IOMMU*.

No artigo de Waldspurger e Rosenblum [WR12], são citados diversos desafios e problemas na área de virtualização de E/S: a carga extra no *hypervisor*, a complexidade em gerenciar recursos (escalonamento e prioridades) e a dificuldade de dar uma semântica ao hardware virtual.

No artigo de Liu [Liu10], foram feitos diversos experimentos com virtualização de E/S baseados em *software* usando *Virtio* e em hardware usando *SR-IOV* (do inglês *single root I/O virtualization* – virtualização de E/S de raiz única) usando o *hypervisor KVM*. O *Virtio* é um padrão do Linux para *drivers* de rede e disco que estão rodando em um ambiente virtual cooperado com um *hypervisor* para-virtualizado. Ele tem um padrão arquitetural similar aos *drivers* de rede do *Xen*, apesar de serem diferentes. Já o *SR-IOV* é uma especificação que permite a dispositivos *pci-express* fornecerem interfaces extras com funcionalidades reduzidas para serem usadas pelas máquinas virtuais diretamente.

Foram analisada as seguintes métricas nesse experimento: a largura de banda, a latência e uso do processador. Na latência, o *Virtio* teve um desempenho muito baixo. A explicação, provada desabilitando a função de agregação de interrupções na transmissão, é que o hospedeiro atrasa o envio do pacotes para ser enviado em rajadas, mas mesmo assim, seu desempenho sem mitigação ainda perdeu próximo de 20 microssegundo em relação a máquina não virtualizada. Quando a opção de agregação é desabilitada, isso provoca uma perda de desempenho pois cada pacote que é transmitido gera uma carga de trabalho no *CPU*. Com a mitigação a carga por pacote é reduzida. Já o *SR-IOV* teve um desempenho próximo da máquina pura perdendo apenas alguns microssegundos devido a virtualização da interrupção.

Na largura de banda, a transmissão em todas as configurações tiveram o mesmo desempenho. Já na recepção o *SR-IOV* se aproximou da máquina pura, mas o uso da sua *CPU* foi muito maior que as demais. No *Virtio*, ele não conseguiu um bom desempenho, mas o uso de sua *CPU* foi baixa. No experimento de uso da memória na recepção, o *SR-IOV* teve um uso muito menor que o *Virtio*, assim, o autor concluiu que o mal uso da largura de banda na recepção do *Virtio* foi pelo uso excessivo da memória, o que explica também o baixo uso da *CPU*.

No artigo de Santos, Turner, Janakiraman e Pratt [STJP08], foi proposto modificar a arquitetura do *driver* de E/S do *XEN* para conseguir melhorar o uso da *CPU*. Dentro dos problemas que o autor encontrou está o excesso de cópias de dados. Para reduzir o excesso de cópias, foram propostos otimizações nas operações de remapeamento de páginas tanto na transmissão como na recepção. No *hypervisor*, ele conseguiu uma economia de 56% no uso do processador.

No artigo de Oi e Nakajima [ON09], foi analisado o desempenho de um sistema virtualizado com *XEN* aplicando a estratégia *LRO (large receive offload)* onde ainda dentro do *driver* da placa de rede são recebidos e reunidos os pacotes de informações que tiveram que ser segmentados. Nesse experimento foram medidos a vazão da rede variando o tamanho da mensagem e o tamanho da *MTU*. Os resultados mostraram um ganho de 8% a 14% na vazão da rede.

No artigo de Apparao, Makineni e Newell [AMN06], foram pesquisadas as principais causas de carga extra na virtualização de E/S. No experimento eles estudaram dois modos de virtualização de E/S: o `domU` e o `dom0` na mesma *CPU* e em *CPUs* distintas. O resultado mostrou que nos dois métodos, tanto a transmissão como a recepção de pacotes apresentaram uma perda de desempenho de mais de 50% quando comparado com a máquina física. Também foi notado que executar o `domU` e o `dom0` em *CPUs* distintas é mais custoso que executar elas juntas na mesma *CPU*.

No artigo de Jang, Seo, Jo e Kim [JSJK11], foi estudado sacrificar o isolamento que existe entre as máquinas virtuais para conseguir reduzir a carga extra do processador. Os resultados mostram uma redução de 29% no uso do processador e 8% de ganho de largura de banda na transmissão de pacotes grandes.

No artigo de Fortuna e Adamczyk [FA12], foram feitos experimentos em torno do problema da carga extra na virtualização da rede. Para isso, os autores propuseram adequar o balanceamento de interrupções para demostrar a possibilidade de reduzir o número de pacotes perdidos. O resultado foi que um balanceamento adequado pode melhorar muito o desempenho, porém, o comportamento é difícil de ser previsto, dificultando a elaboração de um algoritmo. Uma proposta futura sugerida foi deixar o núcleo do sistema automatizar o processo de balanço e analisar os resultados. Quando aparecerem bons resultados, congelar a configuração de interrupção. Eles também, discutiram a possibilidade de usar a função de agregação existentes nos *drivers* das placas de rede modernas para um trabalho futuro.

Evaluating System Performance in Gigabit Networks [SEB05]

No artigo de Salah e El-Badawi [SEB05], é feita uma análise e simulação sobre o impacto da sobrecarga de interrupções no desempenho do sistema operacional em redes de alta velocidade. O principal problema que eles exploraram é a grande quantidade de interrupções gerada na recepção de pacotes. Como a interrupção tem prioridade máxima em relação a outras tarefas, em excesso, ela acaba consumindo todo tempo de processamento, impedindo outras tarefas de serem realizadas e, consequentemente, reduzindo a taxa de transferência do sistema a 0. Essa situação é conhecida como *"livelock"*.

Na Figura 3.1, é mostrado um gráfico de carga do sistema por taxa de transferência. Na curva ideal, conforme a carga do sistema aumenta, a taxa de transferência passa a aumentar proporcionalmente, ou seja, quanto maior a velocidade de chegada dos pacotes, maior a quantidade de pacotes processada. Porém, como praticamente todo sistema tem uma capacidade finita de processamento, ele não recebe e processa pacotes além de sua velocidade máxima. Essa velocidade é chamada de *MLFRR* (do inglês *Maximum Loss-Free Receive Rate* – máxima velocidade de recepção livre de perdas). Na curva aceitável, quando o sistema chega à *MLFRR*, ele passa a não ter ganho na taxa de transferência pelo limite de processamento, e depois disso, a curva se comporta praticamente como uma constante. Por outro lado, se a rede for sobrecarregada na entrada, as interrupções geradas na chegada dos pacotes irão impedir que o pacote seja processado e a taxa de transferência do sistema cairá para 0 como é possível ver na curva *livelock*.

Para analisar a situação de sobrecarga de interrupções, os autores modelaram o sistema como uma fila M/M/1/B com chegada de pacotes em *Poisson* de velocidade λ e média efetiva de tempo de serviço de $1/\mu$ que tem uma distribuição geral (os autores não justificam o motivo de terem

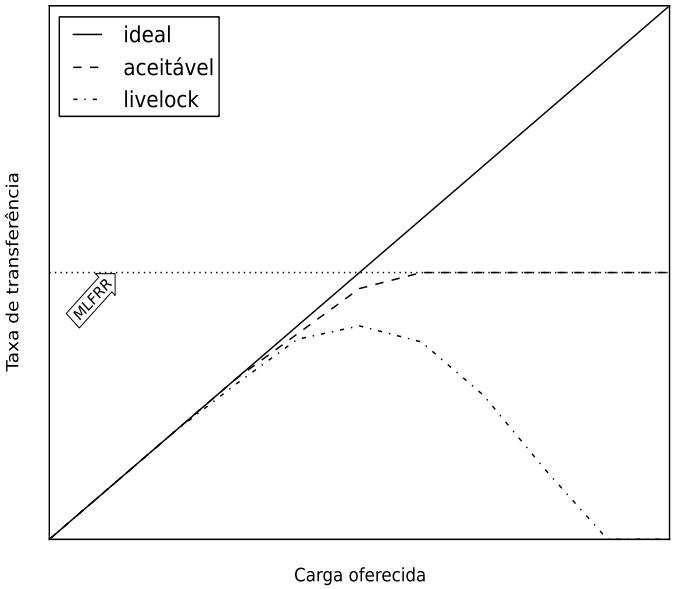

Figura 3.1: *Livelock na recepção de pacotes [SEB05]*

modelado segundo essa distribuição). O sistema pode usar ou não *DMA*. Sem *DMA*, o processador gerencia a recepção de pacotes. Quando o processador é interrompido enquanto está processando um pacote pela chegada de um outro pacote, o tempo para processar é estendido para realizar uma cópia individual do outro pacote que chegou para a memória do sistema. Com *DMA*, a placa de rede tem acesso direto a memória. Quando um ou mais pacotes chegam enquanto que o sistema está ainda processando um outro pacote, todos são processados sem estender o tempo de processar.

Foram feitos experimentos analisando o sistema ideal, *DMA* habilitado e *DMA* desabilitado. Com pouco tráfego de pacotes, a taxa de transferência de todos foi a mesma. Já com muito tráfego, a taxa de transferência do sistema com o *DMA* habilitado teve uma queda menor na taxa de transferência que o sistema sem o *DMA* habilitado e o sistema ideal se apresentou com taxa de transferência constante.

To Coalesce or Not To Coalesce [Sal07]

No artigo de Salah [Sal07], continuação do artigo anterior [SEB05], é analisado o desempenho de duas técnicas de agregação de interrupções: baseada em contagem e baseada em tempo. Na técnica baseada em tempo, o *driver* da placa de rede não gera interrupções no sistema quando um pacote é recebido. Ao invés disso, uma interrupção é gerada depois de um intervalo de tempo que um pacote é recebido. Já na técnica baseada em contagem, é gerada uma interrupção quando uma quantidade de pacotes é recebida.

A conclusão tirada nos modelos analíticos é que a agregação funciona melhor que o modelo de interrupção comum quando se tem um grande tráfego na rede. Porém, para um tráfego pequeno, a interrupção comum superou a agregação. Os autores sugerem monitorar o tráfego e fazer a troca entre a interrupção comum e a agregação de interrupções. Eles também citam um momento que pode ser usado para indicar a condição de sobrecarga. Este momento pode ser usado para a troca. Outras importantes conclusões são que na agregação, são necessários valores altos de parâmetros em tráfegos intensivos, que para tráfegos tolerantes a latência, o uso de agregação é interessante independente do tráfego e que para tráfegos de tempo não-real é interessante usar a agregação baseada em tempo ao invés da baseada em contagem por impor um limite de atraso na agregação.

No artigo de Salah e Qahtan [SQ09], foi apresentada uma estratégia de recepção de pacotes

chamada ativação-desativação de interrupções e implementada uma estratégia híbrida de recepção no *driver* de dispositivo de rede. A ativação-desativação de interrupções funciona como a estratégia comum de gerar uma interrupção a cada pacote que chega, porém, durante a recepção do pacote, as interrupções de recepção são desabilitadas impedindo os pacotes que chegam durante o processo de recepção de gerarem interrupções desnecessárias. A estratégia híbrida alterna entre a *NAPI* e a ativação-desativação de interrupções dependendo do tráfego. Em tráfegos moderados, o *driver* recebe pacotes ativando-desativando interrupções. Já em tráfegos intensivos o *driver* passa a usar *NAPI*. Nos experimentos, foram comparadas as estratégias de ativação-desativação de interrupções, *NAPI* e a proposta de estratégia hibrida. A estrátegia híbrida teve um desempenho melhor que a *NAPI* tanto para tráfegos intensivos como moderados, mas isso foi devido ao parâmetro limite da *NAPI* que foi configurado diferente nas duas estratégias.

Interrupt Moderation Using Intel GbE Controllers [Int07]

No artigo da Intel [Int07], é citado o problema da grande quantidade de interrupções gerada na transmissão e recepção de pacotes. Para resolvê-lo, os autores propuseram o uso de temporizadores internos da placa de rede para moderar a quantidade de interrupções geradas. Os temporizadores são divididos em temporizador absoluto, pacote e mestre. O temporizador absoluto inicia uma contagem regressiva quando o primeiro pacote chega ou é enviado. No momento que a contagem chega a zero, é gerada uma interrupção no sistema. Este temporizador é eficiente quando se tem muito tráfego de pacotes, pois muitos pacotes chegam/são enviados até o temporizador gerar a interrupção, reduzindo a quantidade de interrupções por pacote. Por outro lado, ele não é eficiente quando há pouco tráfego porque poucos pacotes chegam/são enviados até o temporizador gerar a interrupção e por atrasar as interrupções e, consequentemente, as informações que devem chegar ao sistema.

O temporizador de pacotes também inicia uma contagem regressiva quando o primeiro pacote chega ou é enviado e também gera uma interrupção quando a contagem chega a zero, mas ele é reiniciado sempre que um novo pacote chega. Isso reduz a latência quando há pouco tráfego no enlace, pois a interrupção é gerada quando o temporizador percebe que nenhum pacote será mais enviado/recebido, mas quando há muito tráfego, ele pode nunca gerar a interrupção, pois o temporizador estará sempre sendo reinicializado pelos pacotes que chegam/são enviados. O temporizador mestre é usado para otimizar os outros temporizadores. O temporizador absoluto e o temporizador de pacotes podem ser combinados para chegar a um bom resultado.

Além dos temporizadores já citados, existe um outro mecanismo chamado limitador de interrupções. Esse mecanismo também é um temporizador de contagem regressiva e limita o número de interrupções por segundo. Quando o temporizador inicia a contagem regressiva, este também começa a contar o número de interrupções que foi gerado. Quando a contagem chega a zero, o contador de interrupções também é zerado. Se o número de interrupções ultrapassar o limite estabelecido, as interrupções geradas são adiadas até o contador ser reinicializado.

Um algoritmo foi proposto para moderação de interrupções que ajusta dinamicamente o valor do limitador de interrupções. Dependendo do padrão de E/S, é usado um valor no limitador. O padrão de E/S é categorizado em: baixíssima latência, onde o tráfego é mínimo e predomina os pacotes pequenos, baixa latência, onde o tráfego também é minimo e há um significante percentual de pacotes pequenos, e intermediário, onde há muito tráfego de pacotes medianos. Não foi possível entender como os autores chegaram a esses valores e porque eles resolveram dividir o padrão de E/S dessa forma.

Placas de rede da Intel

Atualmente, a *Intel* é uma empresa que tem investido em pesquisas de dispositivos de rede. Suas placas de rede implementam várias funcionalidades exclusivas como a interrupção de baixa latência e a moderação de interrupções dita anteriormente em [Int11]. A interrupção de baixa latência permite que seja gerada uma interrupção imediatamente após a recepção de um pacote que segue algum

critério. Este critério pode ser o valor da porta de destino do pacote, o tamanho do pacote ou o tipo de protocolo *Ethernet*. Apesar dessas funcionalidades ajudarem muito na pesquisa de agregação de interrupções, uma placa de rede da Intel possui um custo muito alto, o que inviabiliza fazer experimentos com ela.

Linux

Linux é um clone do sistema operacional *Unix* escrito do zero por Linus Torvalds com ajuda de uma equipe de *hackers* espalhada por todo o mundo. Seu código-fonte pode ser obtido em http://github.com/torvalds/linux. É importante destacar que tanto o código quanto a documentação são de alta qualidade e atualizados diariamente sendo uma importante fonte de informação confiável para pesquisas. Muitos detalhes e dúvidas tanto de implementação quanto para entender melhor o funcionamento do *driver* de rede foram esclarecidos quando navegamos pelo seu repositório.

Estudando o código-fonte dos *drivers* encontrados no repositório do *Linux*, percebemos que a grande maioria implementava *NAPI* por padrão. Outras estratégias para reduzir interrupções como a agregação por intervalo de tempo e por pacotes dificilmente foram implementadas.

3.1.2 Resumo Conclusivo

Nessa revisão, foram encontrados diversos artigos com propostas que modificam diferentes partes da infraestrutura: *driver* de rede, placa de rede física, arquitetura da virtualização da rede e núcleo do sistema operacional. No artigo da Intel [Int07], foi proposto um algoritmo para a moderação de interrupções de acordo com o padrão de E/S. Não ficou claro como os autores criaram o algoritmo, e é necessário a funcionalidade de moderação de interrupção que apenas alguns *drivers* implementam. No segundo artigo de Salah [Sal07], foi criado um modelo analítico e simulador para analisar a analisar a estratégia de agregação de interrupções por intervalo de tempo e por quantidade de pacotes. Porém, atualmente, não foi possível encontrar *drivers* que implementem essas funcionalidades. No terceiro artigo de Salah [SQ09], foi apresentada uma estratégia híbrida de interrupções em dispositivos físicos. Uma desvantagem dessa estratégia é a necessidade de definir o momento em que se deve trocar entre *NAPI* e ativação-desativação de interrupções. Experimentos com placas que implementam *SR-IOV* e placas da *Intel* apresentaram progressos na virtualização de rede, porém, são muito caras atualmente. A *NAPI* pareceu ser a estratégia mais utilizada atualmente pelos fabricantes de dispositivos de rede. Nesse estudos vimos os *drivers*: e1000, r8169, virtio e tg3, todos implementavam *NAPI*. Não foi encontrado muitos artigos relacionados.

A revisão ajudou a entender melhor a área de virtualização de rede. Porém, foi notada também a falta de detalhes técnicos nos artigos o que dificulta demais a continuação deles. Outro problema é a rápida desatualização das pesquisas. O progresso tanto do *Linux* quanto dos *drivers* deixaram muitas propostas de agregação de interrupções obsoletas. Como a agregação por intervalo de tempo e por pacotes.

Capítulo 4

Experimentos

Para analisar o desempenho da *NAPI* em dispositivos de rede virtuais, foram feitos experimentos variando o parâmetro limite do *driver* de rede e1000 da *Intel* em vários *hypervisors* diferentes. Esse *driver* implementa *NAPI* e tem o código-fonte claro e bem escrito, sendo usado por vários *hypervisors* como *Xen*, *VirtualBox*, *VMWare* e *KVM* para processar a recepção e transmissão de dados pela rede na máquina virtual. Outras soluções que os *hypervisors* teriam são os *drivers* de para-virtualização do *Xen* e do *Virtio*. Ambos têm um desempenho superior ao e1000 por usarem técnicas de para-virtualização, porém, são mais complexos porque se comunicam diretamente com a máquina física.

Na implementação de recepção de pacotes do e1000, o limite define a quantidade limite de pacotes que a tarefa de recepção poderá coletar por ciclo de varredura. Se a quantidade de pacotes atingir esse limite ou esgotar o tempo limite de espera de pacotes, a tarefa é recolocada na fila de varredura, caso contrário, é removida da fila.

4.1 Banda X Limite

Nesse primeiro experimento estudamos a largura de banda em relação a variação de protocolo e limite. Como *hypervisor* foi usado o *VirtualBox*. A máquina física contém um processador i7-2620M de dois núcleos e quatro fluxos de execução, 8 Gigabytes de memória *RAM* e sistema operacional *Mac OS X* 10.6.8 enquanto que a máquina virtual usa dois fluxos de execução, 5 Gigabytes de memória *RAM* e sistema operacional *Ubuntu* 11.10 com núcleo *Linux* 3.0.43.

Foram analisadas a largura de banda na recepção do `iperf` e a quantidade total de pacotes processada pelo *driver*. O limite variou de 1 a 10 de um em um, de 10 a 100 de dez em dez e com o valor 200. A banda de transmissão foi a máxima que a máquina poderia transmitir, no *TCP* dependeria do seu mecanismo de controle de fluxo e no UDP configuramos para a transmitir 1 Gbit/s, mas a transmissão alcançou apenas 810 Mbits/s.

A escolha do intervalo reduzido quanto menor o limite foi feita considerando que as interrupções de *hardware* irão reduzir significantemente nos primeiros limites e muito pouco nos próximos, sendo assim, é esperado que variações na banda ocorram com valores de limite baixos. Também foram considerados o artigo de Corbet [Cor05] e de Salah [SQ09], que sugerem um melhor desempenho para valores baixos de limite. A banda foi medida usando o programa `iperf` com protocolo *TCP* e *UDP* durante 30 segundos e a quantidade de pacotes pelo `ifconfig`. Nos gráficos, são mostradas a média e intervalo de confiança de 95% dos resultados do experimento. Cada experimento foi realizado 10 vezes.

A Figura 4.1 mostra a largura de banda de recepção do `iperf` usando *TCP*. Percebe-se que com limites baixos, o desempenho da largura de banda de recepção é menor em relação aos experimentos com limites altos. Na Figura 4.2, é mostrada a quantidade de pacotes recebida pelo *driver*. Nota-se que existe uma semelhança grande entre a Figura 4.1 e 4.2. Uma correlação linear foi feita com as informações a largura de banda e a quantidade de pacotes recebida. O resultado foi de 0.9999, uma

forte correlação.

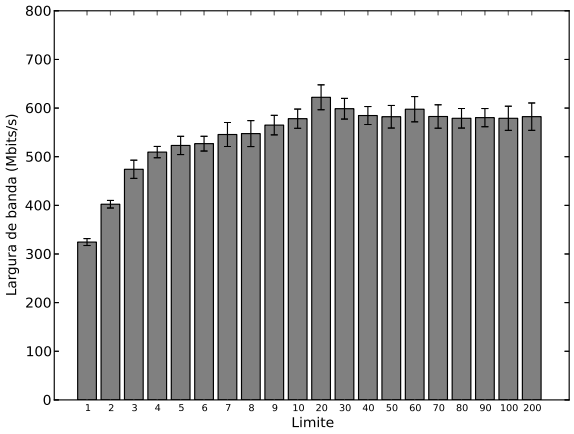

Figura 4.1: *Largura de banda na recepção de pacotes com protocolo TCP*

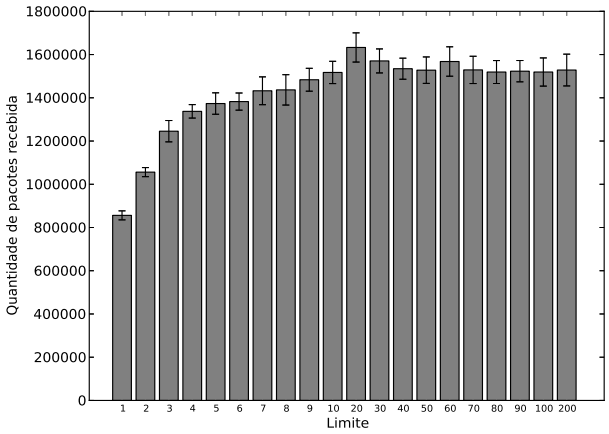

Figura 4.2: *Quantidade de pacotes recebida pelo driver com protocolo TCP*

A Figura 4.3 mostra a banda da recepção usando UDP. Percebe-se um comportamento diferente em relação ao *TCP* visto na Figura 4.1, enquanto no *TCP*, o aumento no limite elevaria a banda, o contrário parece ocorrer no UDP, quanto menor o limite maior a banda. Nota-se também um resultado incomum, a banda do UDP apresenta resultados muito inferiores em relação ao *TCP*.

Na Figura 4.4, é mostrada a quantidade de pacotes recebida usando *UDP*, nota-se que o gráfico se diferencia completamente em relação ao gráfico anterior da Figura 4.3. A correlação linear entre elas foi de -0.9999, um resultado contrário em relação ao experimento com *TCP*.

Existe uma possível resposta para esse resultado: no *UDP*, os pacotes são processados em grandes quantidades no *driver*, com exceção do experimento com limite igual a 1 que processa poucos pacotes. Entre a chegada no `iperf` e o processamento de pacote pelo *driver* pode ocorrer um transbordamento em uma fila de pacotes devido ao excesso de pacotes e isto faz o `iperf` perder muitos pacotes. No *TCP*, devido a característica do protocolo de controlar o fluxo, o envio de pacotes é mais lento e o sistema não consegue um fluxo o suficiente para transbordar a fila.

Sendo mais específico, esse transbordamento ocorreu no *buffer* de recepção do *socket* criado pelo

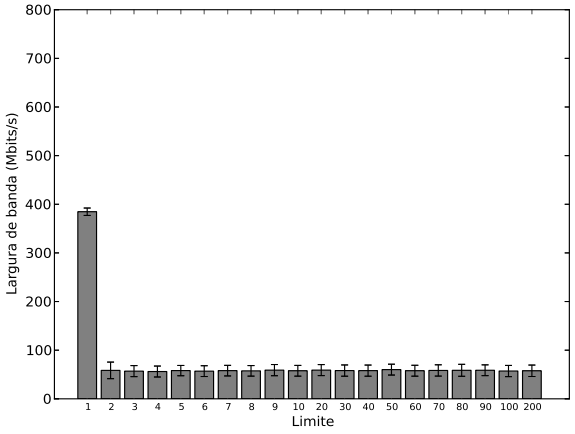

Figura 4.3: *Largura de banda na recepção de pacotes com protocolo UDP*

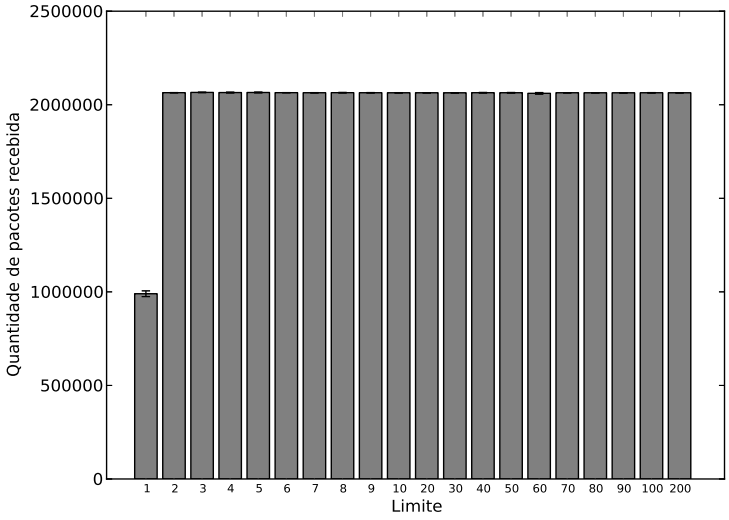

Figura 4.4: *quantidade de pacotes recebida pelo driver com protocolo UDP*

`iperf`. Aumentando esse *buffer*, percebemos um aumento expressivo da banda como é visto nas Figuras 4.5 e 4.6. A correlação linear foi de 0.9426. Talvez seja possível achar uma configuração de *buffer* que se correlacione mais analisando-a com mais detalhes, mas esse resultado já mostra que limites altos têm uma largura de banda maior que limites baixos se configurarmos o *buffer* corretamente.

Assim, concluímos tanto com *TCP* como com *UDP* que configurações de limite alto e ajustes no *buffer* resultam em uma largura de banda maior que configurações com limites baixos. No próximo passo, será estudada a causa dos *drivers* com limites altos terem uma banda de recepção maior que *drivers* com limites baixos.

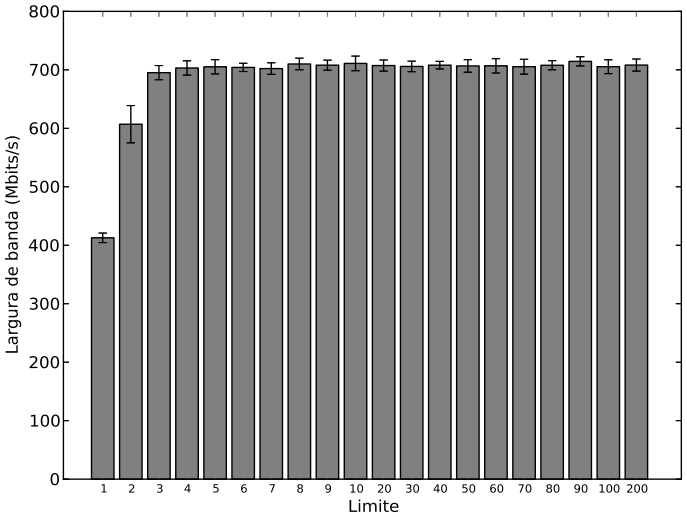

Figura 4.5: *Largura de banda na recepção de pacotes com protocolo UDP modificando o buffer de recepção*

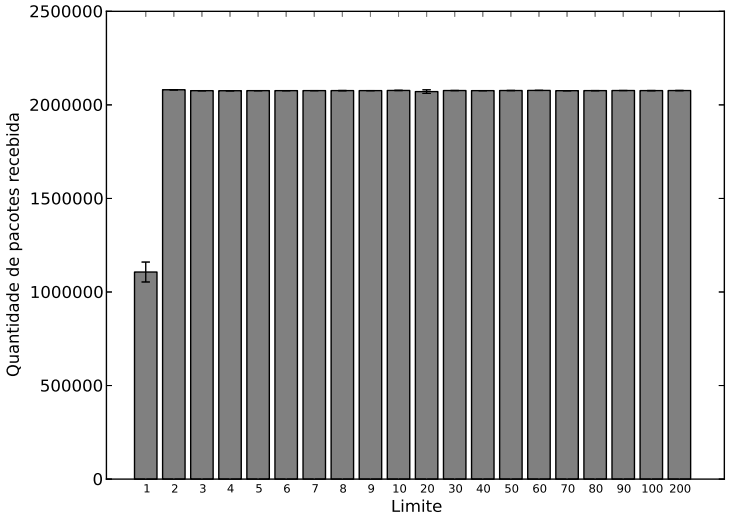

Figura 4.6: *Quantidade de pacotes recebida pelo driver com protocolo UDP modificando o buffer de recepção*

4.2 Interrupções

Foram realizados experimentos com o limite em relação as interrupções com o protocolo *UDP*. Como *hypervisor* foram usados o *VirtualBox*, o *VMware* e o *Xen* com virtualização completa.

No experimento com o *VirtualBox* e *VMware* foram usadas as mesmas configurações de *hardware* e sistema operacional que o experimento anterior. No *Xen*, a máquina física contém um processador i7 Ivy bridge de quatro núcleos e oito fluxos de execução, 16 Gigabytes de memória *RAM* e sistema operacional *Ubuntu* 12.10 enquanto que a máquina virtual usa dois fluxos de execução, 5 Gigabytes de memória *RAM* e sistema operacional *Ubuntu* 11.10 com núcleo *Linux* 3.0.12.

Foram variados a largura de banda de transmissão de 100 até 1000 Mbits/s de cem em cem e o limite em 1, 2, 60 e 200 usando protocolo *UDP*. A escolha do *UDP* foi devido a possibilidade de controlar a banda de envio e a pouca intervenção no fluxo de pacotes em relação ao *TCP*. O Tamanho do *buffer* de recepção do *socket* foi definido para 8 Mbytes. Foram medidos a largura de banda de recepção do `iperf`, a quantidade de pacotes recebida usando o `ifconfig`, o uso de *CPU* total e o uso de *CPU* pelas interrupções de software usando `sar`. No uso de *CPU*, é considerando 100% o uso de uma linha inteira de execução e 200% duas linhas e assim, sucessivamente. Cada gráfico mostra a média dos resultados do experimento. Cada experimento foi rodado 5 vezes.

4.2.1 VirtualBox

Recepção de Pacotes

Na Figura 4.7 é mostrada a largura de banda recebida pelo `iperf` em relação a largura de banda transmitida. Nota-se que todos os limites têm perdas de pacotes, mas conforme a banda aumenta, essa perda é reduzida. Percebe-se também que com limite igual a 1, quando a banda tem transmissão maior que 600 Mbits/s a banda passa a processar aproximadamente 400 Mbits/s. Nos próximos gráficos será analisada a causa desse comportamento.

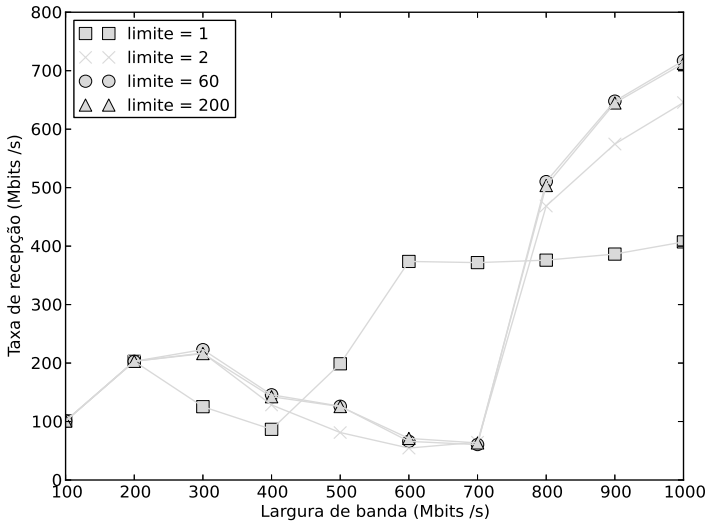

Figura 4.7: *Largura de banda de recepção no VirtualBox*

Na Figura 4.8 é mostrada a quantidade de pacotes recebida pelo *driver*, percebe-se que todas as curvas começam crescente e num determinado momento, elas ficam constantes limitada por algum fator. Esse é possivelmente o ponto de *MLFRR* (máxima velocidade de recepção livre de perdas) citado em [SEB05]. O gráfico com limite igual a 1 passa a ser constante quando a banda é maior que 500 Mbits/s. Provavelmente, o sistema começou a descartar pacotes pois percebeu que não seria

capaz de processá-los, uma característica da *NAPI*. Já os outros limites passam a ficar constantes somente em 800 Mbits/s que é próximo do limite da banda de transmissão, nesse caso, o *driver* conseguiu processar todos os pacotes e não houve perdas. Assim, novamente entre o *driver* e a recepção de pacotes do `iperf`, alguns pacotes deixaram de ser processados.

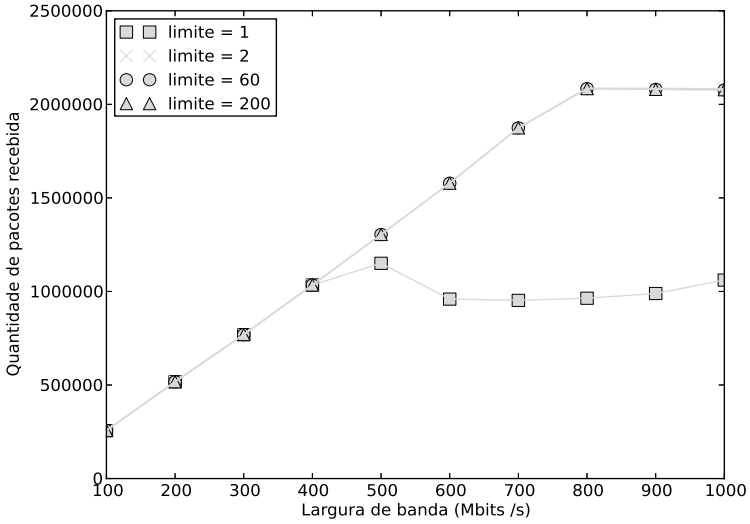

Figura 4.8: *Quantidade de pacotes recebida pelo driver no VirtualBox*

Interrupções de Software e Uso de CPU

Na Figura 4.9 é mostrado o uso de *CPU* pelas interrupções de software. Notamos que com qualquer limite, houve inicialmente um aumento no uso de *CPU* até chegar no intervalo entre 90%-100% onde o uso permanece entre esses valor e com exceção da configuração com limite igual a 1, todos tiveram uma redução com transmissão acima de 700 Mbits/s. Esse aumento provavelmente foi limitado pela *CPU* que será analisada no próximo gráfico. Essa redução será analisada melhor no próximo experimento.

Na Figura 4.10 é mostrado o uso de *CPU* dentro da máquina virtual. Percebemos que com limite igual a 1, a máquina virtual excedeu consideravelmente seu uso em relação aos demais. Monitorando os processos em execução, notamos que com limite igual a 1 e largura de banda de transmissão maior que 500 Mbits/s o `ksoftirqd` estava usando a *CPU* próximo de 100%. Possivelmente o `ksoftirqd` estava processando as interrupções de *software*. Isso ocupou uma *CPU* inteira enquanto que o `iperf` ocupou a outra *CPU*.

Já com limites maiores que 1, o uso de *CPU* se limitou a 100%. Isso foi devido tanto ao sistema quanto ao `iperf` dividirem a mesma *CPU* e não conseguirem processar os pacotes paralelamente. Em 300 Mbits/s, o sistema atingiu 100% e houve perda de pacotes, mas diferentemente do que aconteceu com limite igual a 1, as interrupções não ocuparam 100% da *CPU* como mostrado na Figura 4.9. Nesse caso, o sistema perdeu os pacotes no `iperf`, que divide a *CPU* com as interrupções, e não conseguiu *CPU* o suficiente para processar os pacotes. Somente quando a largura de banda de transmissão se aproximou de 800 Mbits/s, o sistema teve uma considerável redução de uso de *CPU* pelas interrupções de *software* permitindo o `iperf` usar mais a *CPU*.

Na Figura 4.11 mostramos o uso da *CPU* da máquina virtual usando o monitor `top` na máquina física. Comparando com a medição feita dentro da máquina virtual mostrada na Figura 4.10, percebemos que existe uma carga adicional de *CPU* pela virtualização da máquina. Nesse caso,

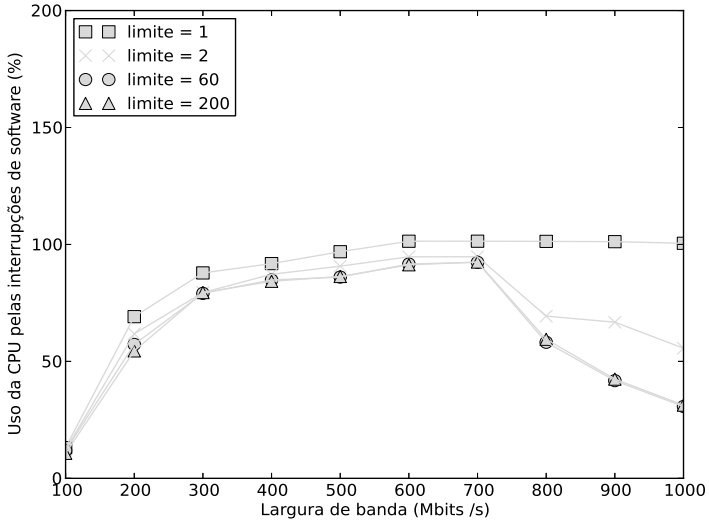

Figura 4.9: *Uso da CPU pelas interrupções de software no VirtualBox*

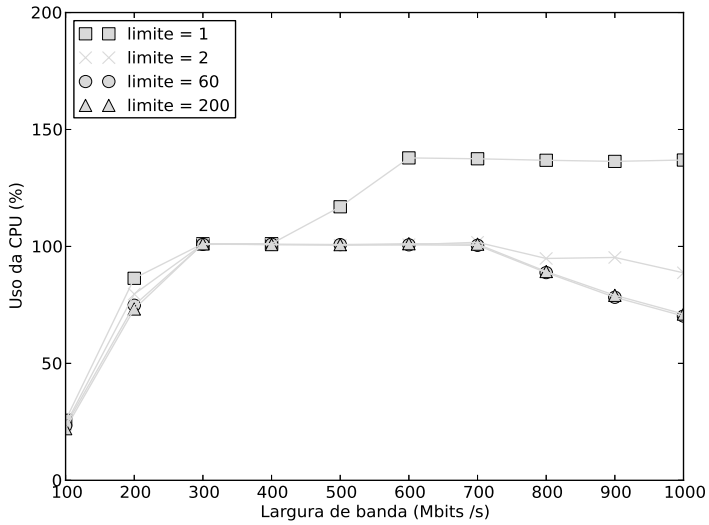

Figura 4.10: *Uso da CPU no VirtualBox*

consideramos o resultado da máquina física, pois a máquina virtual não considera o uso da *CPU* da emulação do sistema e dos dispositivos de rede.

Conclusão

Nesse experimento percebemos que uma única *CPU* é dividida entre as interrupções de *software* e o `iperf` o que causa perda de pacotes no `iperf`. Vimos que com exceção da configuração com limite igual a 1, todos tiveram uma redução no uso de *CPU* com transmissão acima de 700 Mbits/s. Isso será melhor analisado no próximo experimento. Também vimos que o processamento das interrupções de *software* quando ocupa uma *CPU* inteira, faz o sistema alocar processos, nesse caso o `iperf`, para outra *CPU*. Percebemos uma diferença no resultado da medição dentro e fora da máquina virtual, foi considerado o resultado da medição fora da máquina virtual. No próximo experimento separaremos em diferentes *CPUs* as interrupções de *software* do `iperf` para que o

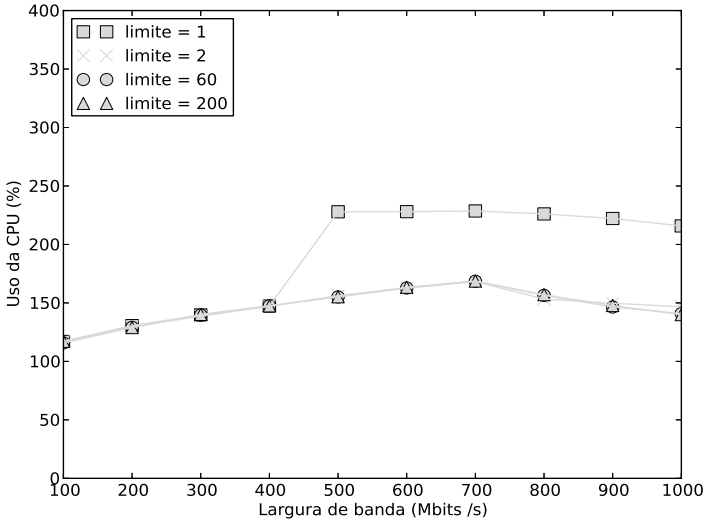

Figura 4.11: *Uso da CPU pela máquina física*

sistema possa ter um melhor proveito da *CPU*.

4.2.2 VirtualBox com Afinidade de CPU

No experimento anterior, vimos que as interrupções de *software* e o `iperf` usam a mesma *CPU* e como resultado, o `iperf` não conseguiu processar todos os pacotes. Assim, temos duas soluções: definir uma *CPU* para o `iperf` ser executado ou definir uma *CPU* para a interrupção. A configuração de associar um processo a uma determinada *CPU* é chamada de afinidade de *CPU*, já a de associar uma interrupção a uma *CPU* é chamada de balanceamento de interrupções. Decidimos associar uma *CPU* para o `iperf` por ser uma solução simples feita através da ferramenta `taskset`. Nesse experimento, foram usados o mesmo *hardware* e sistema operacional do experimento anterior.

Largura de Banda e Recepção pelo *Driver*

As Figuras 4.12 e 4.13 mostram respectivamente a largura de banda recebida pelo `iperf` e a quantidade de pacotes recebida pelo *driver* no *VirtualBox*. Percebemos que são muito semelhantes e não existe uma queda na banda como no experimento anterior (Figura 4.7). Notamos também que com limite igual a 1 continua tendo perdas de pacotes no *driver*.

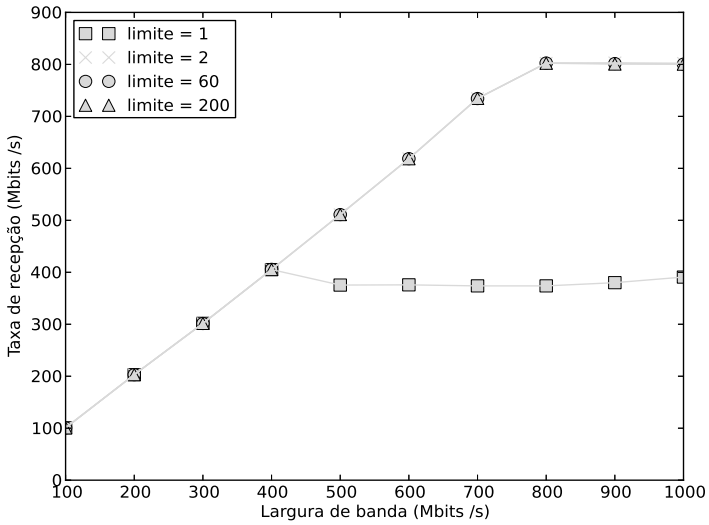

Figura 4.12: *Largura de banda de recepção no VirtualBox com afinidade de CPU*

Uso de CPU

Na Figura 4.14 é mostrado o uso da *CPU* pelo *VirtualBox* na máquina física. Vemos que o uso de *CPU* muito foi maior que em relação ao experimento passado (Figura 4.11) chegando a ultrapassar 200%. Esse aumento foi devido a configuração de afinidade de *CPU*.

Notamos que como no experimento anterior, em transmissões acima de 700 Mbits/s houve uma redução no uso de *CPU*. Usando o `dmesg`, programa que imprime as mensagens do núcleo do sistema, e modificando o código fonte do *driver* para imprimir a quantidade de pacotes por ciclo de varredura, veremos a causa dessa redução. Iremos dividir essa explicação em dois cenários: entre 700 Mbits/s e 800 Mbits/s e acima de 800 Mbits/s.

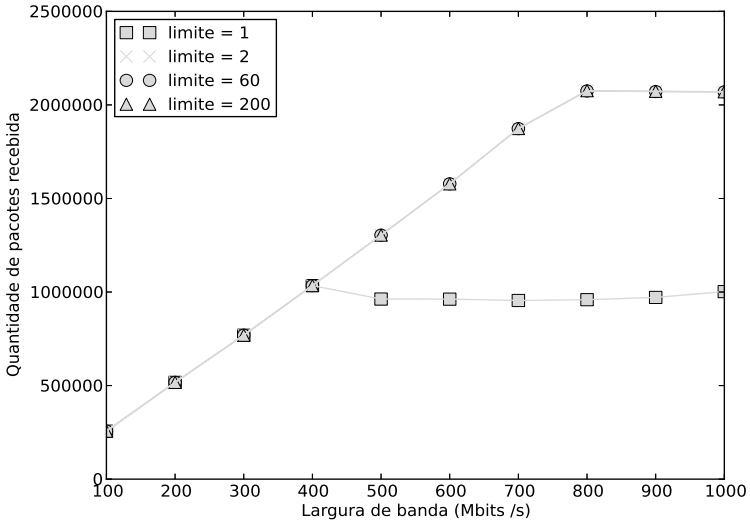

Figura 4.13: *Quantidade de pacotes recebida pelo driver no VirtualBox com afinidade de CPU*

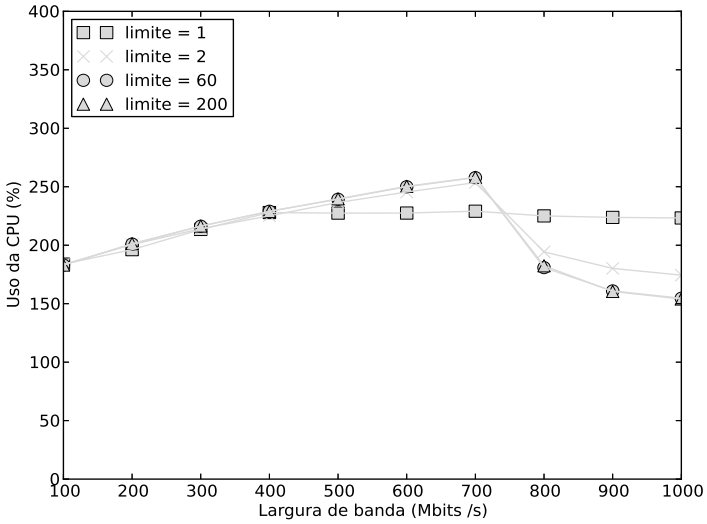

Figura 4.14: *uso da CPU pela máquina física com afinidade de CPU*

Entre 700 Mbits/s e 800 Mbits/s, as Figuras 4.15 e 4.16 mostram a frequência de pacotes processada por ciclo de varredura com limite igual a 60 e respectivamente largura de banda de transmissão igual a 700 e 800 Mbits/s. Com transmissão de 700 Mbits/s, nota-se que o sistema processa num intervalo entre 5 e 12 pacotes, já com 800 Mbits/s o sistema passa a processar quantidades maiores de pacotes, ainda na maioria das vezes é processado entre 5 e 12 pacotes, mas em frequência menor. Com o aumento de pacotes processados por ciclo de varredura, a quantidade de ciclos para processar todos os pacotes é reduzida e, consequentemente, o uso de *CPU* pelas interrupções de *software*.

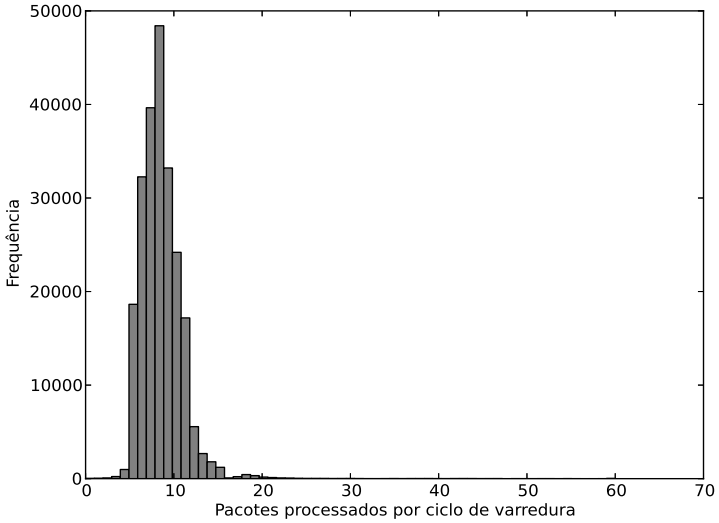

Figura 4.15: *Quantidade de pacotes processada por ciclo de varredura com largura de banda de transmissão de 700 Mbits/s*

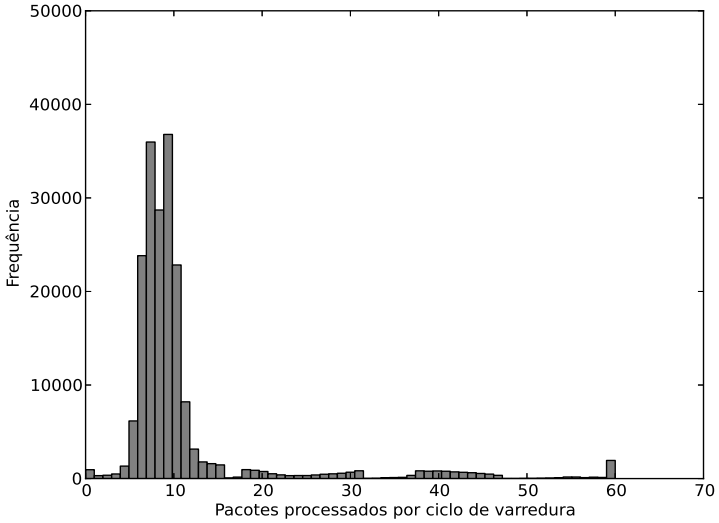

Figura 4.16: *Quantidade de pacotes processada por ciclo de varredura com largura de banda de transmissão de 800 Mbits/s*

Com limite igual a 2, o sistema é capaz de processar no máximo 2 pacotes por ciclo. Nesse caso, não teria como reduzir a quantidade de ciclos de varredura, então o sistema processa continuamente os pacotes em vários ciclos contínuos de varredura. Como exemplo, se 10 pacotes foram processados em um ciclo com limite igual a 60, com limite igual a 2, seriam necessários 5 ciclos, em que 2 paco-

tes serão processados a cada ciclo. Entre ciclos, normalmente o sistema reabilita as interrupções de *hardware* e espera a chegada de pacotes, mas como o limite de 2 foi atingido e há pacotes esperando serem processados, o sistema não reabilita as interrupções e executa outro ciclo de varredura.

A Figura 4.17 mostra a taxa de interrupções de *hardware* por segundo em relação a largura de banda. Com limite igual a 1 e largura de banda igual a 500 Mbits/s a taxa de interrupções de *hardware* se aproximou de 0 e, nesse caso, os pacotes foram processados continuamente em vários ciclos de varredura. Consequentemente, a quantidade de tarefas de varredura sobrecarregou o sistema e este passou a descartar pacotes. Com limite igual a 2 a taxa de interrupções não se aproximou de 0, mas comparando com limite igual a 60 e 200, teve maiores agregações de interrupções de *hardware*, pois muitas vezes o sistema teve que processar em ciclos contínuos de varredura. Com limite igual a 60 e 200 houve uma redução na taxa de interrupções quando a banda foi maior que 800 Mbits/s. Isto pode ser explicado pelo gráfico de frequência de pacotes por ciclo de varredura mostradas nas Figuras 4.15 e 4.16. Com o aumento na frequência de pacotes, menos ciclos de varredura são necessários e menor a taxa de interrupções de *hardware*. Pelos resultados de largura de banda e uso de *CPU* não houve muita diferença entre reduzir a quantidade de ciclos de varredura e agregar as interrupções através de ciclos contínuos de varredura variando o limite.

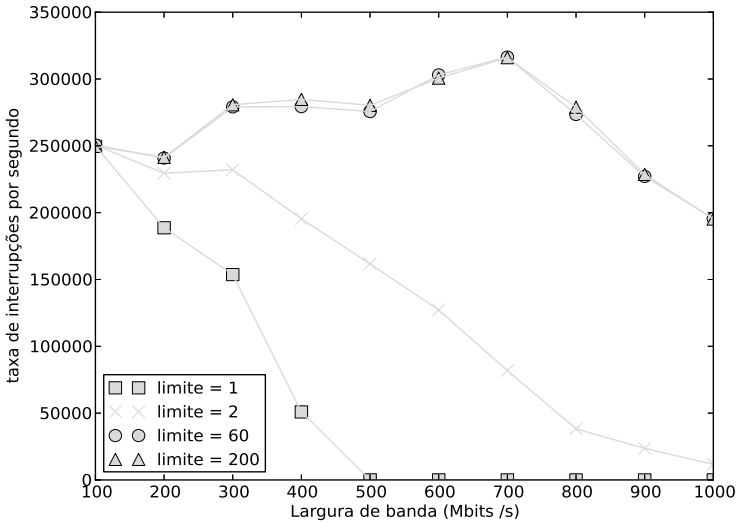

Figura 4.17: *Quantidade de interrupções de hardware gerada pela placa de rede virtual no VirtualBox*

Acima de 800 Mbits/s de transmissão, percebe-se que mesmo com o limite de largura de banda de transmissão sendo de 810 Mbits/s, existe uma diferença entre o uso da *CPU*. Na Figura 4.18 é mostrada a frequência de pacotes processada por ciclo de varredura com banda de transmissão igual a 1000 Mbits/s e limite igual a 60. Observa-se que em relação ao gráfico de 800 Mbits/s, com 1000 Mbits/s, o sistema processou mais pacotes por ciclo de varredura. Um resultado curioso, pois como a transmissão máxima é de 810 Mbits/s, ambos deveriam mostrar resultados semelhantes. Este resultado ocorreu pela forma como o `iperf` transmite os pacotes. Olhando o código-fonte do `iperf`, quando se é usado o protocolo *UDP*, existe um atraso entre envios de pacotes que é calculado pela formula:

$$atraso\_entre\_envios = \frac{tamanho\_do\_pacote}{largura\_de\_banda\_de\_transmisso} + reajuste$$

Onde reajuste é o valor reajuste de acordo com o tempo gasto no laço para transmitir o pacote.

Assim, quanto maior a largura de banda de transmissão, menor o atraso entre envios. Quando a taxa de transmissão do `iperf` é maior que o máximo que o dispositivo de rede é capaz de transmitir, os pacotes são enviados inicialmente em atrasos menores e conforme o dispositivo é sobrecarregado de pacotes para transmitir, o `iperf` reajusta o valor de atraso. Com esse ajuste dinâmico de envios, é muito provável que ocorra mais rajadas de grande quantidade de pacotes, do que se tentasse transmitir com o atraso entre envios constante podendo alterar o desempenho na recepção de pacotes.

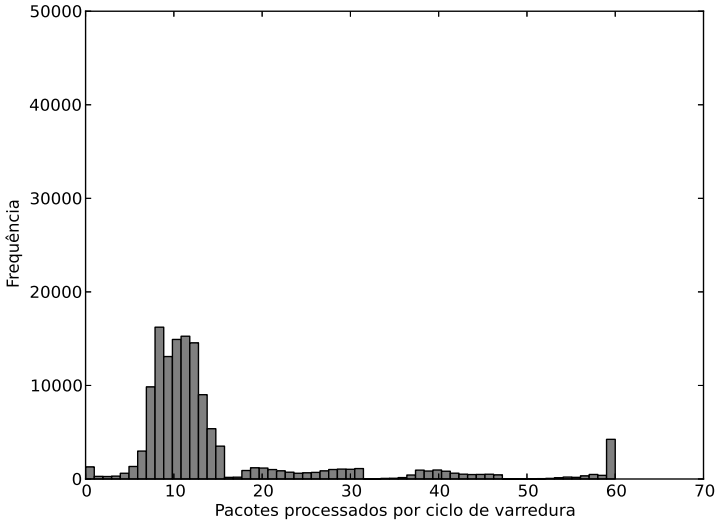

Figura 4.18: *Quantidade de pacotes processada por ciclo de varredura com largura de banda de transmissão de 1000 Mbits/s*

Interrupções de Software

Nas Figuras 4.19 e 4.20 é mostrado o uso da *CPU* respectivamente pelas interrupções de software e total. Comparando com os resultados sem afinidade de *CPU*, percebemos que o uso de *CPU* pelas interrupções de software e total foram muito menores. Mas como já foi mostrado na Figura 4.14, o uso da *CPU* pelo *VirtualBox* foi muito maior em relação ao experimento anterior. Não foi possível descobrir o motivo do `sar` mostrar uma redução do uso de *CPU* dentro da máquina. É possível que seja um problema do próprio *VirtualBox*.

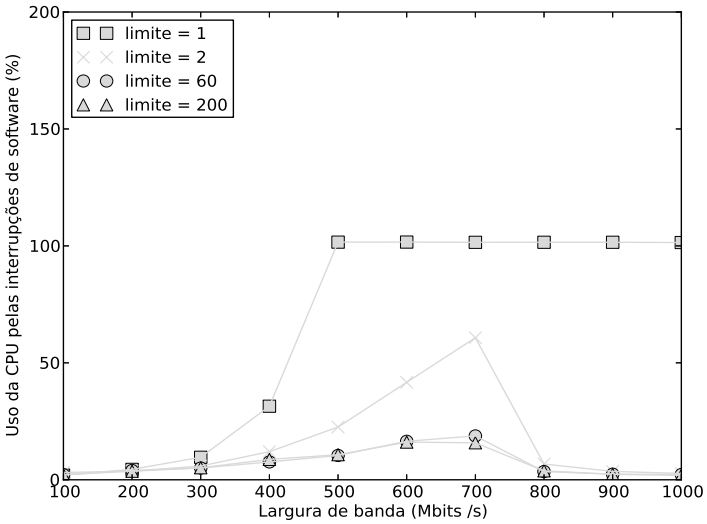

Figura 4.19: *Uso da CPU pelas interrupções de software no VirtualBox com afinidade de CPU*

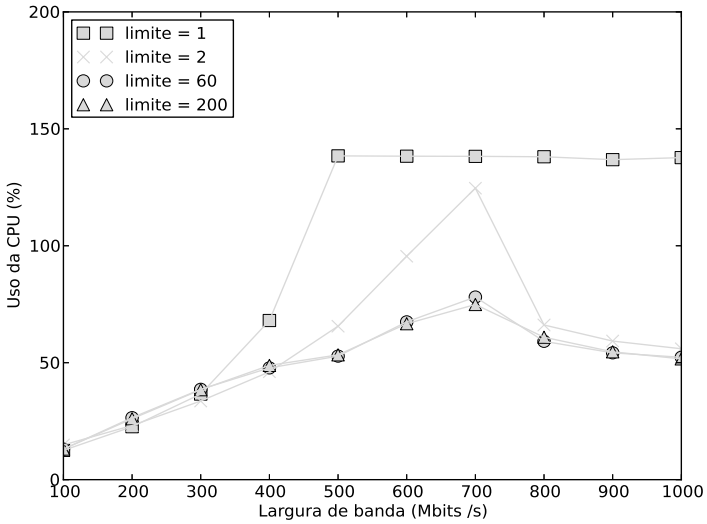

Figura 4.20: *Uso da CPU no VirtualBox com afinidade de CPU*

Conclusão

Concluímos que separando o `iperf` das interrupções, é possível aproveitar melhor a *CPU* para o processamento de pacotes. Porém, isso gera uma carga extra no processamento, quando não necessita usar duas *CPUs*. Assim, é interessante saber se os processos usarão mais de uma *CPU* ou se uma será o suficiente.

Novamente, com exceção do experimento com limite igual a 1, vimos o uso de *CPU* reduzir entre 700 Mbits/s e 800 Mbits/s. Há duas explicações, a agregação de interrupções através do processamento contínuo de pacotes e a redução de ciclos de varredura.

Percebemos que o `iperf` configurado para enviar pacotes em taxas maiores que a capacidade do dispositivo passa a ajustar a transmissão dinamicamente, alterando a frequência de envio e aumentando a quantidade de rajadas longas de pacotes. Isso provavelmente fez reduzir o uso da *CPU* com 900 Mbits/s e 1000 Mbits/s em relação ao resultado com 800 Mbits/s.

Também concluímos que o parâmetro de limite não afeta a largura de banda ou o uso real da *CPU*, a não ser que o valor seja muito baixo. As medições de dentro da máquina virtual não mostraram com precisão o uso da *CPU* pelas interrupções de *software* quando usamos duas *CPUs*, não foi possível descobrir a causa.

4.2.3 Xen

Nos experimentos com o *Xen*, ocorreram problemas tanto no *buffer* do *socket* quanto no uso da mesma *CPU* pelo `iperf` e as interrupções de *software* como no experimento com o *VirtualBox*. Para resolvê-los, novamente aumentamos o tamanho do *buffer* e associamos uma *CPU* para o `iperf`.

Nas Figuras 4.21 e 4.22 que mostram respectivamente a banda de recepção, a quantidade de pacotes recebida pelo *driver*, ocorreu uma situação semelhante a vista no *VirtualBox*, ambos os gráficos ficaram iguais e com limite igual a 1, houve perda de pacotes, porém, ao invés da perda ocorrer quando a transmissão é maior que 400 Mbits/s como no caso do *VirtualBox*, esta ocorre com transmissões maiores que 600 Mbits/s. O limite da transmissão novamente é próximo de 810 Mbits/s.

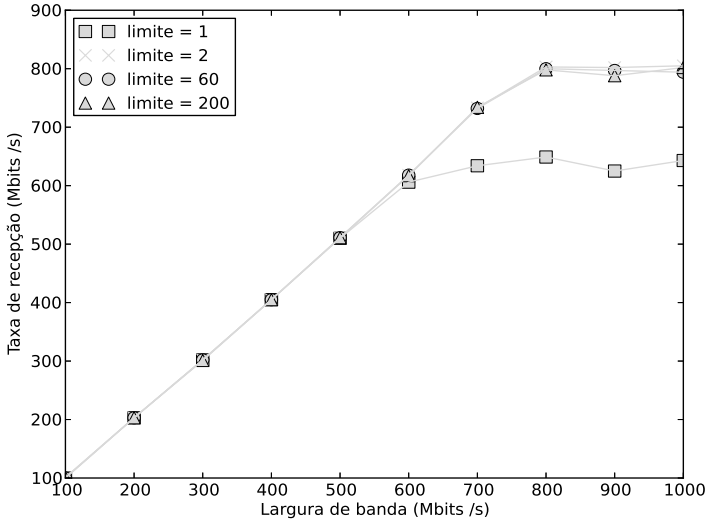

Figura 4.21: *Largura de banda de recepção no Xen*

Na Figura 4.23, vemos o uso de *CPU* da máquina virtual. Esses dados foram obtidos através do `xentop`, aplicação de monitoramento que o *Xen* fornece. Percebe-se que o uso de *CPU* é maior quanto maior o limite, mas entre experimentos com limite igual a 60 ou 200 quase não existe diferença. Nota-se também que o uso de *CPU* não excedeu 100%, mas com limite igual a 1 e largura de banda maior que 600 Mbits/s o sistema parece estar no máximo não conseguindo exceder o uso da *CPU*. Acima de 800 Mbits/s o sistema parece reduzir pouco o uso da *CPU*.

Nas medições de uso de *CPU* dentro da máquina virtual ocorreu algo semelhante ao experimento do *VirtualBox*. Houve uma diferença entre a medição interna e externa. Então, desconsideramos a medição interna.

Conclusão

No *Xen*, percebemos que o uso de *CPU* pelo tráfego de rede foi menor em relação ao *VirtualBox*. Também notamos que com limite igual a 1, o sistema é capaz de processar mais pacotes. Entre 700 Mbits/s e 800 Mbits/s novamente com exceção do experimento com limite igual a 1, houve uma redução no uso da *CPU* mas muito menor em relação ao *VirtualBox*. Com limites igual a 1 vimos que novamente houve perda de pacotes já com limite igual a 2 houve um uso maior da *CPU* em relação aos limites iguais a 60 e 200.

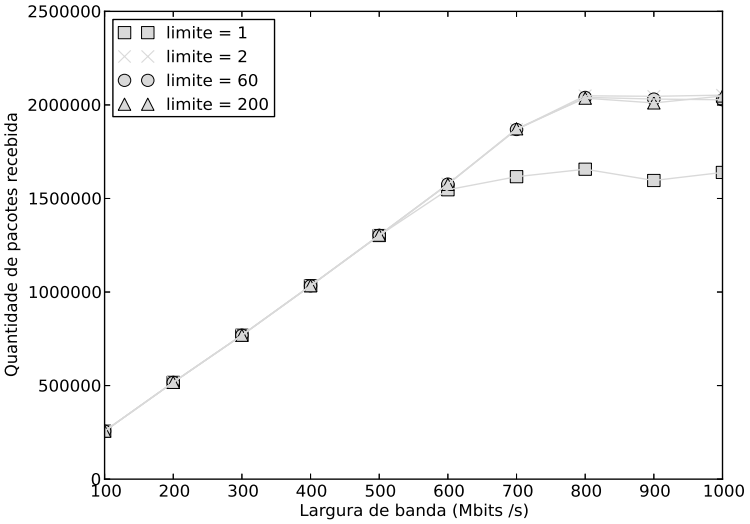

Figura 4.22: *Quantidade de pacotes recebida pelo driver no Xen*

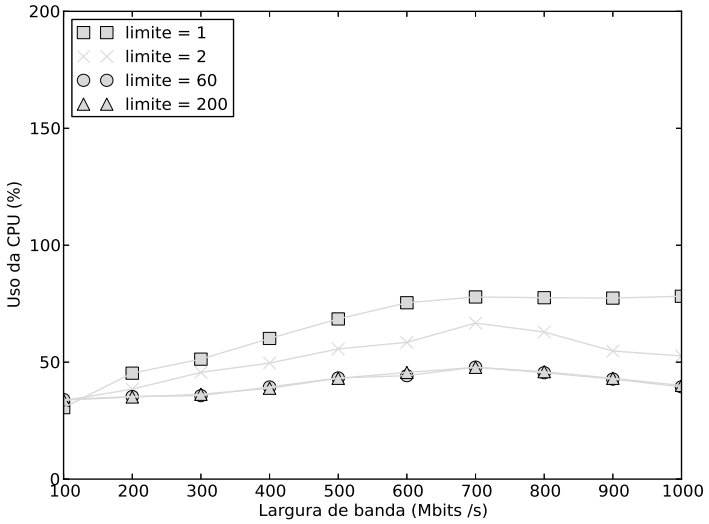

Figura 4.23: *uso da CPU no Xen*

4.2.4 VMware

No *VMware*. Ocorreram os mesmos problemas do *Xen* e do *VirtualBox* então reconfiguramos o *buffer* do *socket* e a afinidade de *CPU* do `iperf`.

Nas Figuras 4.24 e 4.25, que mostram respectivamente a banda de recepção e a quantidade de pacotes recebida pelo *driver*, ambos os gráficos ficaram iguais. Com limite igual a 1 e 2, houve perda de pacotes quando a transferência é maior que 400 Mbits/s. Notamos que houve perda de pacotes com banda de transmissão maior que 800 Mbits/s. O limite da transmissão, novamente, é próximo de 810 Mbits/s.

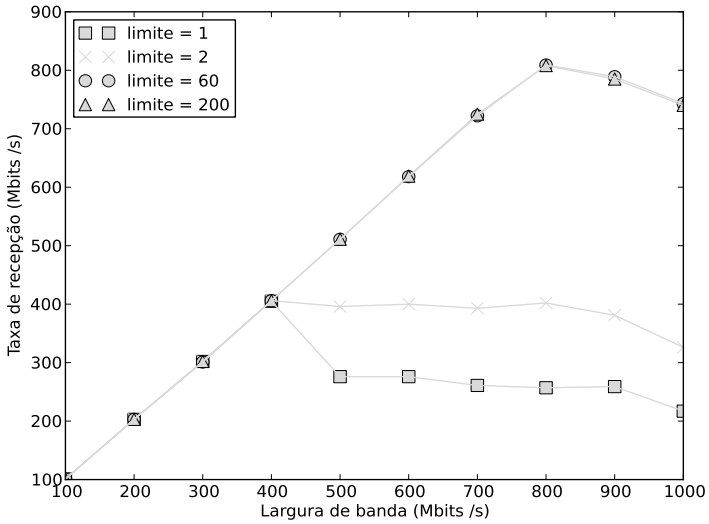

Figura 4.24: *Largura de banda de recepção no VMware*

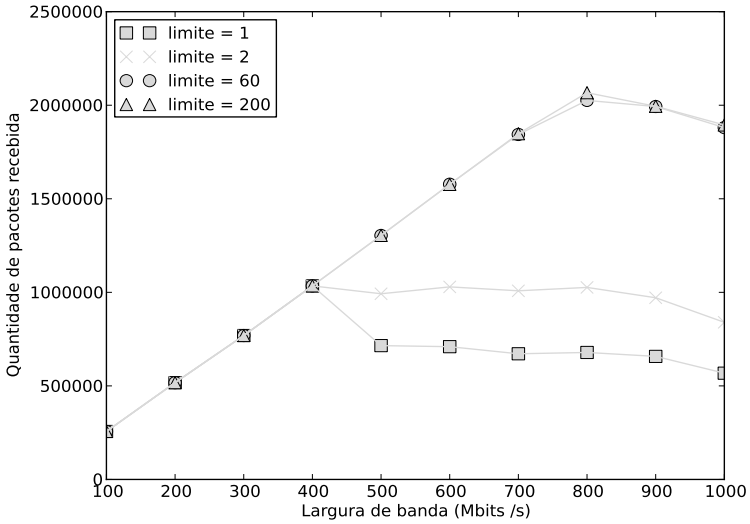

Figura 4.25: *Quantidade de pacotes recebida pelo driver no VMware*

Na Figura 4.26, observamos que o uso de *CPU* chega ao máximo quando a largura de banda

é maior que 200 Mbits/s, tendo uma queda entre 700 Mbits/s e 800 Mbits/s provavelmente pela transmissão rápida de pacotes. Acima de 800 Mbits/s nota-se que o uso de *CPU* não variou.

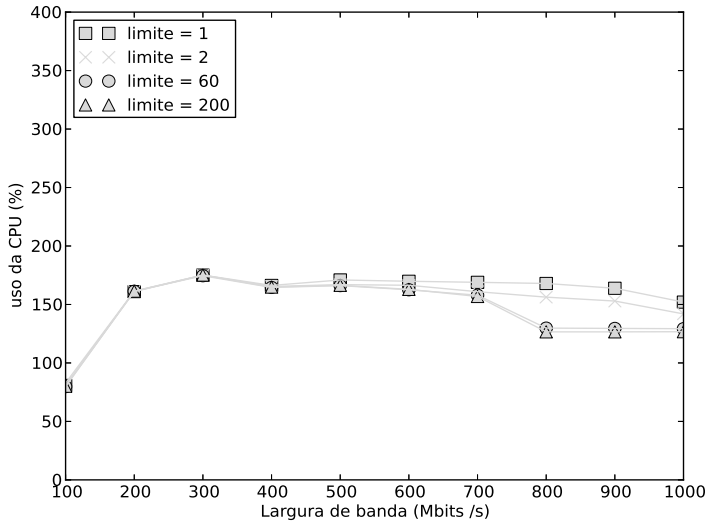

Figura 4.26: *Uso da CPU no VMware*

Conclusão

Vimos que, com largura de banda maior que 800 Mbits/s, há perda de pacotes, enquanto que nos outros *hypervisors*, a banda permanece constante e o uso de *CPU* é reduzida. Provavelmente, o sistema não conseguiu processar os pacotes em grandes rajadas. Com limite igual a 1 e 2 houve perda de pacotes com transferência maior que 400 Mbits/s.

4.3 Análise dos Resultados

Na Tabela 4.27, é comparado os resultados com diferentes hypervisors. Em todos experimentos, tivemos que configurar o tamanho do *buffer* do *socket* e selecionar a *CPU* na qual o `iperf` seria executado pois estavam comprometendo a medição.

Em nossos experimentos, percebemos que o parâmetro limite com valores baixos (1 e 2) causou perda de pacotes ou aumento de uso da *CPU* comparado com os limites altos (60 e 200). Houve um bom desempenho tanto em uso de *CPU* como largura de banda com limites altos na maioria dos casos e não houve diferença no resultado entre limites altos.

Entre 700 e 800 Mbits/s em todos experimentos, houve uma redução de uso de *CPU* devido a agregação de interrupções através do processamento contínuo de pacotes e a redução de ciclos de varredura. Valores de banda de transmissão acima de 800 Mbits/s causaram um aumento de rajadas longas de pacotes. No *VMware*, isso gerou perdas de pacotes, no *VirtualBox* e no *Xen* houve uma redução de uso de *CPU*.

Houve uma grande diferença no uso de *CPU* pelas máquinas virtuais. O *VirtualBox* foi o que mais gastou *CPU* com até 270% de uso. Já o *Xen* foi o que menos gastou com até 80% de uso. Por fim, o *Vmware* gastou até 175% da *CPU*, porém, ele chegou ao limite de uso de *CPU* mais rápido, enquanto que os outros *hypervisors*, atingiram o máximo com 700 Mbits/s, o *Vmware* atingiu o máximo em 300 Mbits/s.

		Hypervisors		
		VirtualBox	Xen	Vmware
Detalhe de Implementação	Foi necessário ter afinidade de CPU?	Sim	Sim	Sim
	Foi necessário ter aumento no buffer do socket	Sim	Sim	Sim
Resultado da Largura de Banda	houve perda de pacotes com limite igual a 1?	Sim, acima de 500 Mbits/s e o tráfego se manteve em 400 Mbits /s	Sim, acima de 700 Mbits/s e o tráfego se manteve em 620 Mbits/s	Sim, acima de 500 Mbits/s e o tráfego se manteve em 360 Mbits/s
	houve perda de pacotes com limite igual a 2?	Não	Não	Sim, acima de 500 Mbits/s e o tráfego se manteve em 400 Mbits/s
	houve perda de pacotes com limite igual a 60 e 200?	Não	Não	Não
	com rajada maiores de pacotes nas transmissões acima de 800 Mbits/s, houve perda de pacotes?	Não	Não	Sim
Resultado do Uso de CPU pela Máquina Virtual	com limite igual a 1, o uso de CPU foi maior em relação aos demais?	apenas com transmissão acima de 800 Mbits/s	Sim	apenas acima de 500 Mbits/s
	com limite igual a 2, o uso de CPU foi maior em relação aos limites iguais a 60 e 200?	apenas acima de 800 Mbits/s	Sim	apenas acima de 700 Mbits/s
	o uso de CPU foi identico nos experimentos com limites iguais a 60 e 200.	Sim	Sim	Sim
	com limite igual a 60 ou 200, o uso de CPU foi maior em relação aos limites iguais a 1 e 2?	apenas entre 500 e 700 Mbits/s	Não	Não
	qual foi o uso máximo de CPU? E qual foi taxa de transmissão mínima necessária?	270%, 700 Mbits/s	80%, 700 Mbits/s	175%, 300 Mbits/s
	Com transmissão de 800 Mbits/s houve redução no uso de CPU em relação a transmissão de 700 Mbits/s?	Sim	Sim	Sim
	com rajada maiores de pacotes nas transmissões acima de 800 Mbits/s, houve redução no uso de CPU?	Sim	Sim	Não

Figura 4.27: *Planilha comparando os resultados com diferentes hypervisors*

Capítulo 5

Conclusões

Em nossos experimentos percebemos que o parâmetro limite igual a 1 causou uma perda de pacotes *CPU*. Isso ocorreu devido ao processamento contínuo de pacotes que, quando reduz a taxa de interrupções próximo a 0, sobrecarrega o sistema com interrupções de *software* e este passa a descartar pacotes.

Com limite igual a 2, no *VirtualBox* não houve diferença em relação aos limites 60 e 200, no Xen, o sistema usou mais *CPU* em relação aos limites 60 e 200, por fim, no *VMware* houve perda de pacotes semelhante ao que ocorreu com limite igual a 1. Com limites altos (60 e 200) houve um bom desempenho tanto em uso de *CPU* como largura de banda e não houve diferença no resultado entre valores altos.

Assim, sabe-se que diferementemente dos dispositivos físicos que têm bons resultados com limites baixos, nos dispositivos virtuais, limites altos tiveram um bom desempenho em todos os casos comparado com limites baixos. No caso particular, com o e1000, *driver* usado nos experimentos, o valor de limite por padrão é 64, não sendo necessário alterá-lo se decidirmos escolher o melhor limite.

Vimos que o uso de *CPU* é sensível a mudanças na frequência de envio de pacotes mesmo quando a largura de banda não varia. É provável que controlar a frequência de envio de pacotes, aumente a frequência de pacotes processada e, consequentemente, reduza o uso de *CPU* como ocorreu no *VirtualBox* e *Xen* com transmissão acima de 800 Mbits/s.

Quando monitoramos o uso de *CPU* da máquina virtual com o *Xen*, *VirtualBox* e *Vmware*, os resultados mostraram que o *Xen* foi o que usou menos *CPU* e teve o melhor desempenho da largura de banda.

5.1 Sugestões para Pesquisas Futuras

Durante o experimento configuraramos o tamanho do *buffer* do *socket*. A escolha do valor para esse parâmetro poderia ser automatizado para um valor que não use muita memória e não descarte pacotes.

Também selecionamos a *CPU* na qual o `iperf` seria executado pois estavam comprometendo a medição. Entre usar uma *CPU* e usar duas, em tarefas que exigem apenas uma *CPU*, houve uma carga extra no processamento quando usamos duas *CPU*. Já com tarefas que exigem duas *CPUs*, experimentos com apenas uma CPU resultou em perda de pacotes. É interessante analisar quando os processos poderão usar uma *CPU* ou mais.

Referências Bibliográficas

[AFG+09] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica e M. Zaharia. Above the clouds: A berkeley view of cloud computing. Relatório Técnico UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009. 1, 6

[AMD08] AMD. AMD virtualization (AMD-V) technology. http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx, 2008. Acessado em 8/8/2013. 8

[AMN06] P. Apparao, S. Makineni e D. Newell. Characterization of network processing overheads in Xen. Em *Proceedings of the 2nd international Workshop on Virtualization Technology in Distributed Computing*, página 2. IEEE Computer Society, 2006. 17

[BDF+03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt e A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, Outubro 2003. 1, 7, 8, 15

[BM99] S. Bradner e J. Mcquaid. RFC 2544–benchmarking methodology for network interconnect devices. http://www.ietf.org/rfc/rfc2544.txt, 1999. Acessado em: 8/8/2013. 11

[Cor05] J. Corbet. NAPI performance - a weighty matter. http://lwn.net/Articles/139884/, 2005. Acessado em: 8/8/2013. 2, 13, 21

[Cor09] J. Corbet. Generic receive offload. http://lwn.net/Articles/358910/, 2009. Acessado em: 8/8/2013. 14

[Cor12] J. Corbet. Software interrupts and realtime. https://lwn.net/Articles/520076/, 2012. Acessado em: 8/8/2013. 5

[CRKH05] J. Corbet, A. Rubini e G. Kroah-Hartman. *Linux Device Drivers*, páginas 497–595. O'Reilly Media, 3 edição, 2005. 1, 13, 14

[CSYL10] J. Che, C. Shi, Y. Yu e W. Lin. A synthetical performance evaluation of OpenVZ, Xen and KVM. Em *Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference*, APSCC '10, páginas 587–594, Washington, DC, USA, 2010. IEEE Computer Society. 7

[Dun12] G. Dunlap. The paravirtualization spectrum. http://blog.xen.org/index.php/2012/10/23/the-paravirtualization-spectrum-part-1-the-ends-of-the-spectrum/, 2012. Acessado em 8/8/2013. 8

[DXZL11] Y. Dong, D. Xu, Y. Zhang e G. Liao. Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling. Em *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, páginas 26–34, Washington, DC, USA, 2011. IEEE Computer Society. 1, 12

[Eas07] T. Eastep. Xen network environment. http://www1.shorewall.net/XenMyWay.html, 2007. Acessado em: 8/8/2013. vii, 11

[EF10] J. Ekanayake e G. Fox. High performance parallel computing with clouds and cloud technologies. Em *Cloud Computing*, páginas 20–38. Springer Berlin Heidelberg, 2010. 1, 9, 15

[FA12] T. Fortuna e B. Adamczyk. Improving packet reception and forwarding within virtualized Xen environments. Em *Computer Networks*, páginas 153–160. Springer, 2012. 17

[GED+11] G.E. Gonçalves, P.T. Endo, T. Damasceno, A.V.A.P. Cordeiro, D. Sadok, J. Kelner, B. Melander e J.E. Mångs. Resource allocation in clouds: Concepts, tools and research challenges. *Simpósio Brasileiro de Rede de Computadores*, 2011. 1, 9

[Int07] Intel. Interrupt moderation, Intel gbe controllers. http://www.intel.com/content/www/us/en/ethernet-controllers/gbe-controllers-interrupt-moderation-appl-note.html, 2007. Acessado em: 8/8/2013. 19, 20

[Int11] Intel. Linux ixgbe* base driver overview and installation. http://www.intel.com/support/network/adapter/pro100/sb/CS-032530.htm, 2011. Acessado em: 8/8/2013. 19

[Jam04] T.Y. James. Performance evaluation of Linux bridge. http://facweb.cti.depaul.edu/jyu/Publications/Yu-Linux-TSM2004.pdf, 2004. Acessado em: 8/8/2013. 11

[JSJK11] J.W. Jang, E. Seo, H. Jo e J.S. Kim. A low-overhead networking mechanism for virtualized high-performance computing systems. *The Journal of Supercomputing*, páginas 1–26, 2011. 17

[Liu10] J. Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe NICs with SR-IOV support. Em *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, páginas 1–12. IEEE, 2010. 1, 9, 16

[MCZ06] A. Menon, A.L. Cox e W. Zwaenepoel. Optimizing network virtualization in Xen. Em *Proceedings of the annual conference on USENIX'06 Annual Technical Conference*, páginas 2–2. USENIX Association, 2006. 14

[NSL+06] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers e Rich Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. http://noggin.intel.com/content/intel-virtualization-technology-hardware-support-for-efficient-processor-virtualization, 2006. Acessado em 8/8/2013. 8

[ON09] H. Oi e F. Nakajima. Performance analysis of large receive offload in a Xen virtualized system. Em *International Conference on Computer Engineering and Technology, 2009 (ICCET'09)*, volume 1, páginas 475–480. IEEE, 2009. 17

[PZW+07] P. Padala, X. Zhu, Z. Wang, S. Singhal, K.G. Shin et al. Performance evaluation of virtualization technologies for server consolidation. *HP Laboratories Technical Report*, 2007. 7

[Rix08] S. Rixner. Network virtualization: Breaking the performance barrier. *Queue*, 6(1):37:36–37:ff, Janeiro 2008. 1, 8, 9, 10, 15

[Ros05] Rami Rosen. Introduction to the Xen virtual machine. http://www.linuxjournal.com/article/8540, 2005. Acessado em 8/8/2013. 8

[Sal07] K. Salah. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*, 61(4):215–225, 2007. 1, 12, 18, 20

[SBdSC] A.H. Schmidt, M.P. Boufleur, R.C.M. dos Santos e A.S. Charao. Análise de desempenho da virtualização de rede nos sistemas Xen e OpenVZ. http://www-usr.inf.ufsm.br/~canofre/site/arquivos/2007/errc_07_ap.pdf. Acessado em: 8/8/2013. 7

[SEB05] K. Salah e K. El-Badawi. Analysis and simulation of interrupt overhead impact on OS throughput in high-speed networks. *International Journal of Communication Systems*, 18(5):501–526, 2005. vii, 2, 5, 17, 18, 25

[Spe10] S. Spector. New to Xen guide. http://www.xen.org/files/Marketing/NewtoXenGuide.pdf, 2010. Acessado em: 8/8/2013. 10

[SQ09] K. Salah e A. Qahtan. Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme. *Computer Communications*, 32(1):179–188, 2009. 18, 20, 21

[Sta10] W. Stallings. *Computer organization and architecture*, páginas 195–232. Pearson Education India, 8 edição, 2010. vii, 3, 4, 5, 10

[STJP08] J. Santos, Y. Turner, G. Janakiraman e I. Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. Em *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, páginas 29–42, Berkeley, CA, USA, 2008. USENIX Association. vii, 9, 10, 11, 12, 16

[WCC+08] J.P. Walters, V. Chaudhary, M. Cha, S.G. Jr. e S. Gallo. A comparison of virtualization technologies for HPC. Em *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications*, AINA '08, páginas 861–868, Washington, DC, USA, 2008. IEEE Computer Society. 1, 6, 7, 9, 15

[WR12] C. Waldspurger e M. Rosenblum. I/O virtualization. *Communications of the ACM*, 55(1):66–73, 2012. 1, 9, 16