MARCO ANTONIO JONACK

LIMITES DE CAPACIDADE E PROTEÇÃO DE SERVIDORES EM REDES GIGABIT

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dra. Cristina Duarte Murta

CURITIBA

2005

MARCO ANTONIO JONACK

LIMITES DE CAPACIDADE E PROTEÇÃO DE SERVIDORES EM REDES GIGABIT

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dra. Cristina Duarte Murta

CURITIBA

2005

Dedicatória

Dedico este trabalho ao meu sobrinho, Gabriel, cuja recente vinda a este mundo acabou por se tornar uma das grandes fontes de motivação para a conclusão deste e vários outros projetos da minha vida. Espero sinceramente estar de alguma forma contribuindo para um futuro melhor para ele e toda a sociedade.

Agradecimentos

A Deus, que mesmo nos momentos de descrença não me privou das suas bênçãos.

Aos meus pais, Celso e Maria, cujo apoio e carinho foram fundamentais para esta conquista.

Ao meu irmão, Carlos, pela paciência nos momentos de crise.

À minha orientadora, Cristina Duarte Murta, pelos inestimáveis conselhos, pela infinita paciência, pela grande ajuda, e principalmente por toda a confiança depositada na minha pessoa ao longo desses vários anos.

Ao professor Alexandre Ibrahim Direne, pela motivação nos momentos difíceis.

Aos meus colegas, Pedro Rodrigues Torres Júnior e Rodrigo Augusto Alves, pelo auxílio prestado.

A todos aqueles que de alguma forma participaram do desenvolvimento deste trabalho.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

Resumo

O crescimento significativo de usuários com acesso à Internet, em conjunto com a popularização de redes de alta velocidade, estão criando novas e interessantes possibilidades de serviços oferecidos pela grande rede mundial. É comum que os servidores que hospedam tais serviços enfrentem, devido a eventos externos e as próprias características do tráfego Web, situações temporárias onde a carga observada é muito maior do que a sua capacidade, isto é, situações de *sobrecarga transiente*. Este fato tem motivado o desenvolvimento de diversos mecanismos que visam proteger os servidores de eventos de sobrecarga. Entre os mecanismos mais populares está o *controle de admissão*. Contudo, mecanismos de controle de admissão podem falhar caso não considerem as conseqüências de um efeito característico de sistemas orientados à interrupções, o efeito *receive livelock*, cuja ocorrência é potencialmente maior em redes de alta velocidade. Algumas soluções para este efeito foram propostas e implementadas nos mais diversos sistemas operacionais.

Este trabalho apresenta a avaliação de dois mecanismos de controle de *livelock* atuando em máquinas de diferentes capacidades, interligadas por uma rede gigabit. O primeiro mecanismo é conhecido como moderação de interrupções e está presente na maioria das interfaces de rede gigabit comercializadas atualmente. O segundo mecanismo, denominado NAPI, é uma nova API para o desenvolvimento de drivers de rede do sistema Linux, que utiliza a técnica de espera ocupada para o tratamento de pacotes. Os resultados mostram que a NAPI é mais eficiente do que a moderação de interrupções em vários aspectos, incluindo estabilidade do sistema em situações de tráfego extremo e utilização de CPU.

Abstract

The significant growth of users with access to the Internet and the widespread deployment of high-performance networks have fuelled new and interesting services offered through the large world-wide net. Due to external events and to proper characteristics of the Web traffic, computers that host these services have faced temporary situations where the observed load is higher than their capacity. These events are called transient overloads.

This fact has motivated the development of diverse mechanisms that aim to protect the servers from the overload. Admission control is one of the most popular mechanisms. However, it can fail in the presence of other event called receive livelock, which is a common event in interrupt-driven systems, in particular in high-speed networks. Some solutions for this event have been designed and implemented in most operating systems.

This dissertation presents the evaluation of two mechanisms for livelock control, experienced in computers of different capacities in a gigabit network. The first mechanism is called interrupt moderation. It is implemented in most high-performance network interfaces today. The second mechanism, called NAPI, is a new API for Linux network drivers. NAPI uses busy waiting to handle packets. The results show that NAPI is more efficient than interrupt moderation in many aspects, including the system stability in situations of huge traffic and CPU utilization.

Capítulo 1

Introdução

O crescente número de usuários com acesso a Internet e a utilização cada vez maior das interfaces Web estão sendo os precursores do recente fenômeno que está mudando a forma de uso da grande rede mundial, que passa de uma infraestrutura de comunicação e navegação para um promissor meio para conduzir negócios, compra e venda de serviços.

Ao mesmo tempo em que os serviços oferecidos pela Internet se tornam populares, há um considerável aumento em termos de complexidade e escala destes serviços, muitos dos quais exigem garantias rígidas de desempenho e disponibilidade. Do ponto de vista de um servidor Web, satisfazer tais garantias se torna especialmente difícil devido a própria natureza do tráfego da Internet, conhecido por sua extrema variabilidade, rajadas e picos de carga [12].

Eventos externos, como promoções, datas festivas, acidentes, ataques terroristas, notícias de interesse geral, bem como ações mal-intencionadas, como ataques de negação de serviço, podem fazer com que um servidor experimente uma carga muitas vezes maior do que a sua capacidade, caracterizando uma situação de *sobrecarga*, onde pelo menos um dos seus recursos (CPU, memória, disco, banda de rede ou outro) se torna saturado. Se tal sobrecarga é observada ocasionalmente e tem períodos não muito longos de duração, então ela é chamada de *sobrecarga transiente*. No entanto, os efeitos colaterais de um período de sobrecarga incidem principalmente sobre as aplicações hospedadas pelo servidor: mensagens de erro nos navegadores dos usuários, tempos longos de resposta e, inevitavelmente,

perda de clientes e de lucro.

Este quadro tem motivado a realização de vários estudos e pesquisas relativos ao tratamento e à prevenção de sobrecarga, visto que os métodos tradicionais, tais como Web *caching* e balanceamento de carga, se mostram pouco eficazes nestes casos [7]. Alguns estudiosos argumentam que o tratamento de sobrecarga deve ser parte integrante do projeto de sistemas e serviços baseados na Internet [16, 42].

Entre os trabalhos desenvolvidos, o *controle de admissão* tem recebido especial atenção nos artigos publicados nos últimos anos, se mostrando um mecanismo eficiente e versátil no gerenciamento da sobrecarga em servidores Internet. De forma geral, um mecanismo de controle de admissão deve monitorar constantemente alguma medida de desempenho do sistema e, caso esta medida atinja um certo valor considerado crítico, o mecanismo deve restringir o aceite de novas requisições no sistema. Além disso, é interessante que tal mecanismo, ao rejeitar uma requisição, forneça uma mensagem explícita ao usuário, evitando assim uma sobrecarga adicional causada por novas tentativas por parte do usuário.

Entretanto, a popularização progressiva de redes de alta velocidade, principalmente da ordem de mega e gigabit, as quais deixam de fazer parte apenas da infraestrutura da Internet e redes locais para chegar até os usuários e provedores de serviço, muda o foco das atenções para um efeito comum em sistemas orientados à interrupções: o *receive livelock* [34]. Um sistema em *livelock* gasta todos os seus recursos processando interrupções, não dando chance para que outras tarefas (inclusive aquelas relativas a um possível mecanismo de controle de admissão) sejam executadas. Para um observador externo, o sistema parece estar em *deadlock*, pois nenhum avanço é visto no que se refere aos processos do sistema e, conseqüentemente, nenhum progresso em termos de vazão (*throughput*).

Além disso, há alguns anos a velocidade das redes vem aumentando muito mais rapidamente do que a velocidade dos processadores, das memórias e do barramento de E/S [21, 33], os quais se apresentam como elementos fundamentais para a determinação da capacidade de um sistema. Tal desigualdade contribui diretamente para a ocorrência de *livelocks*.

Felizmente alguns mecanismos de controle de *livelock* foram propostos e implementados nos mais diversos sistemas operacionais, incluindo BSD [26], Windows NT [17] e Linux [36]. Também é crescente o número de interfaces de rede, principalmente interfaces gigabit, que oferecem algum tipo de mecanismo para limitar o número de interrupções gerados pela chegada de pacotes, o que ajuda a diminuir a carga imposta à CPU do sistema.

Tendo como motivação o contexto descrito acima, este trabalho apresenta a avaliação de dois mecanismos de controle de *livelock* atuando em máquinas de diferentes capacidades, as quais estão interligadas por uma rede gigabit. O primeiro mecanismo é conhecido como *moderação de interrupções* e está presente na maioria das interfaces de rede gigabit comercializadas atualmente. O segundo mecanismo, denominado NAPI, é uma nova API para o desenvolvimento de *drivers* de rede do sistema Linux, que utiliza a técnica de espera ocupada para o tratamento de pacotes. Estes dois mecanismos foram avaliados sob diversas perspectivas, incluindo testes como uma aplicação real, o servidor Web Apache.

Os resultados mostram que a NAPI é mais eficiente do que a moderação de interrupções em vários aspectos, incluindo estabilidade do sistema em situações de tráfego extremo e utilização da CPU, revelando-se um mecanismo de proteção efetivo contra *livelocks*. Também foi constatado que a NAPI produz um grande número de perdas de pacotes, o que certamente não é desejado em sistemas que devem apresentar características de qualidade de serviço.

O restante do trabalho é organizado como descrito a seguir. O Capítulo 2 discorre sobre os principais mecanismos e técnicas empregadas no controle de sobrecarga em sistemas, além de apresentar os principais conceitos necessários ao entendimento deste trabalho. O Capítulo 3 descreve a configuração de experimentos definida para avaliar os mecanismos de controle de *livelock*, cujos resultados são mostrados no Capítulo 4. Finalmente, o Capítulo 5 conclui o trabalho e apresenta as possibilidades para trabalhos futuros.

Capítulo 2

Controle de Sobrecarga

A crescente demanda por serviços que apresentem características de qualidade e diferenciação de serviço tem motivado o desenvolvimento de sistemas cada vez mais robustos à sobrecarga. Praticamente qualquer sistema, em especial aqueles que operam na Internet, pode enfrentar situações em que a carga oferecida ultrapassa a sua capacidade. Nesses casos, é desejável que os sistemas ofereçam mecanismos que realizem eficientemente o descarte da carga excedente, permitindo que o sistema continue a operar de forma estável. De fato, alguns estudiosos defendem que o gerenciamento de sobrecarga deveria ser parte integrante do projeto de sistemas [16, 42].

A abordagem padrão utilizada por grande parte dos sistemas para controlar a sobrecarga é a *contenção de recursos* [42]. Nesta abordagem, o acesso aos recursos é limitado por parâmetros definidos pelos administradores do sistema em tempo de compilação ou através de arquivos de configuração. Estes parâmetros atuam sobre estruturas internas do sistema, como filas, tamanho dos *buffers*, número máximo de conexões ou usuários simultâneos, dentre outros. Sob estas estruturas, muitas vezes a política utilizada para descartar a carga excedente é a *rejeição de cauda* (*tail drop*), na qual as últimas tarefas a chegar são descartadas caso o limite da estrutura tenha sido atingido. Também é importante notar que a decisão de qual tarefa deve ser rejeitada pode ter um impacto significativo sobre o desempenho do sistema.

O grande problema com a contenção de recursos é o seu caráter estático. Os valores

dos parâmetros que limitam o acesso aos recursos muitas vezes não podem ser alterados em tempo de execução, implicando que o sistema não se adapta à intensidade da carga. Além disso, tais valores, em geral, não são definidos com base na capacidade real do sistema, mas sim definidos de forma arbitrária e rígida.

A principal consequência da definição de limites estáticos e arbitrários sobre os recursos de um sistema é que, na maioria das situações práticas, o fato de um dado limite sobre um recurso ter sido alcançado não caracteriza necessariamente um estado real de sobrecarga. Por exemplo, o servidor Web Apache [15] limita o número máximo de conexões simultâneas que podem ser atendidas. Quando este número é alcançado, independente do estado do sistema (sobrecarregado ou não), o servidor pára de aceitar novas conexões.

Em particular, a contenção de recursos pode se mostrar bastante ineficiente quando empregada em sistemas que estão submetidos à cargas extremamente variáveis, como os sistemas baseados na Internet. Nestes sistemas, nos quais picos e rajadas de carga de diversas intensidades são freqüentes [12], é muito difícil definir uma configuração estática ótima para os valores dos parâmetros que limitam o acesso aos recursos. Estas mesmas dificuldades são observadas até mesmo quando soluções mais tradicionais como Web *caching* e balanceamento de carga são empregadas com a finalidade de amenizar os efeitos de situações de sobrecarga [7].

Frente a este quadro, vários pesquisadores têm proposto abordagens mais eficientes para controlar a sobrecarga. Entre as propostas mais populares figura o *controle de admissão*. Diferentemente das outras abordagens, o controle de admissão se apresenta como um mecanismo robusto e eficiente que procura adaptar, de forma dinâmica, a carga à capacidade do sistema. No entanto, qualquer mecanismo de controle de admissão que não leve em consideração as consequências de um efeito comum em sistemas orientados à interrupções, o *receive livelock*, certamente falhará, visto que um sistema em tal estado gasta todos os seus recursos apenas processando interrupções.

Dentro deste contexto, a Seção 2.1 define os principais conceitos relacionados ao desempenho de sistemas computacionais. As Seções 2.2, 2.3 e 2.4 apresentam o estado da arte quanto às pesquisas relacionadas ao tema de gerenciamento de sobrecarga. Final-

mente, a Seção 2.5 compara as principais características dos mecanismos de controle de admissão e controle de *livelock*.

2.1 Questões de Desempenho em Sistemas Computacionais

Um *sistema computacional* ou apenas *sistema* pode ser definido como uma coleção de recursos de hardware e software que oferece algum serviço. Por exemplo, um sistema pode ser um único componente de hardware, como uma CPU, pode ser um sistema de software, com um sistema de banco de dados ou um servidor Web, ou até mesmo uma rede com vários computadores interligados.

A quantidade de tarefas imposta ao sistema por unidade de tempo caracteriza a *carga de trabalho* ou *carga* experimentada pelo sistema. A carga também pode ser definida em relação a capacidade de processamento do sistema. A *capacidade do sistema*, por sua vez, é dada em função de diversas variáveis, incluindo a arquitetura do sistema, a plataforma de hardware, o tipo do serviço oferecido, bem como pelas características da carga.

O comportamento do sistema em relação a carga determina o seu desempenho ou, em outras palavras, quão bem o sistema responde as solicitações de serviço impostas pela carga. A maneira usual utilizada para mensurar o desempenho de um sistema é através de *métricas*. As métricas mais comuns utilizadas na avaliação de desempenho de sistemas são o *tempo de resposta* e a *vazão* (*throughput*) [19].

O tempo de resposta pode ter várias definições dependendo do sistema em questão. Uma definição possível seria o intervalo entre a emissão de um pedido e o primeiro sinal de resposta por parte do sistema. Outra definição seria o intervalo entre a emissão de um pedido e o final da resposta correspondente. Em especial, no caso de aplicações distribuídas, o tempo de resposta pode incluir os tempos de transmissão do pedido e da resposta. Por fim, em alguns casos, o tempo de resposta também é referenciado como *latência*.

A vazão é definida como a taxa na qual tarefas são processadas pelo sistema, ou seja, a quantidade de serviço que o sistema produz por unidade de tempo. Em vista disso, esta

métrica muitas vezes é chamada de *taxa de serviço*[1]. Por exemplo, a vazão de servidores Web pode ser medida em requisições HTTP completadas por segundo (req/s). Já em redes, a vazão é medida em pacotes por segundo (p/s) ou em bits por segundo (b/s).

Outra métrica de interesse para a avaliação de sistemas é o *jitter*. Esta métrica, principalmente utilizada em estudos de sistema interligados em rede, captura o comportamento de pacotes transmitidos entre dois pontos, medindo a variação da latência entre a chegada de dois pacotes consecutivos.

A Figura 2.1, reproduzida de [19], mostra o comportamento típico da vazão e do tempo de resposta em função da carga. A vazão máxima alcançada em condições ideais de carga é chamada de *capacidade nominal*. Da mesma forma, em um sistema ideal, a vazão sempre condiz com a carga imposta ao sistema. Entretanto, como os sistemas têm uma capacidade finita, a vazão geralmente acompanha a carga apenas até um certo ponto. Este ponto é considerado muitas vezes o *ponto ótimo de operação* do sistema, visto que o tempo de resposta é extremamente baixo se comparado à quantidade de serviço produzida. Para cargas além deste ponto, a vazão já não apresenta ganhos significativos, enquanto o tempo de resposta cresce consideravelmente. À medida que a carga continua a aumentar, os recursos do sistema vão ficando saturados e a *capacidade útil* do sistema, que é capacidade máxima obtida em termos práticos, finalmente é atingida.

Quando a carga imposta ao sistema ultrapassa a sua capacidade útil, o sistema entra em um estado de *sobrecarga*. Em outras palavras, as tarefas chegam ao sistema a uma taxa maior do que o sistema pode completá-las. Em casos em que os períodos de sobrecarga enfrentados pelo sistema não são muito longos e ocorrem ocasionalmente, a sobrecarga é definida como *transiente*. Sobrecargas transientes são muito comuns em sistema baseados na Internet, onde o tráfego é conhecido por sua extrema variabilidade, rajadas e picos de carga [12]. Porém, quando um sistema está em sobrecarga, é comum que um dos seus recursos seja mais requisitado e atinja primeiro a saturação. Tal recurso caracteriza o *ponto de contenção* do sistema.

Embora a vazão de um sistema não acompanhe a carga além da sua capacidade, é

[1]Os termos "vazão" e "taxa de serviço" serão usados neste trabalho como sinônimos.

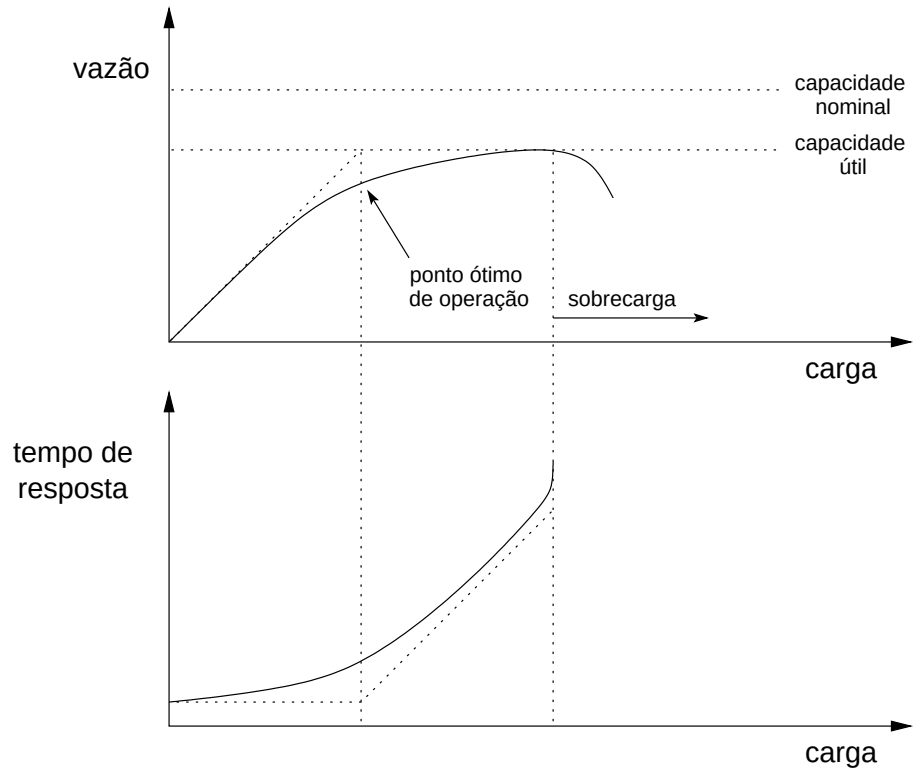


Figura 2.1: Comportamento da vazão e do tempo de resposta em função da carga.

esperado que pelo menos a vazão se mantenha ao máximo enquanto a sobrecarga persistir. Entretanto, em sistemas que não possuem mecanismos eficientes para realizar o descarte da carga excedente, a vazão tende a diminuir consideravelmente quando a carga ultrapassa a sua capacidade útil (Figura 2.1). Esta e outras questões relacionadas ao gerenciamento de sobrecarga serão discutidas nas próximas seções.

2.2 Controle de Admissão

Esta seção apresenta diversos trabalhos no âmbito do controle de admissão em servidores. Os mecanismos de controle de admissão podem ser divididos em dois grupos, os que fazem o controle com base em pacotes ou requisições, discutidos na Seção 2.2.1, e os que fazem o controle baseado em sessões, descritos na Seção 2.2.2. Embora os termos "pacote" e "requisição" sejam genéricos, a partir deste ponto, para fins de simplicidade, eles estarão sendo utilizados para representar, respectivamente, pacotes IP/TCP/UDP e requisições HTTP. Será deixada explícita no texto qualquer outra definição diferente desta.

2.2.1 Controle de Admissão Baseado em Pacotes ou Requisições

O objetivo do controle de admissão baseado em pacotes é, de forma geral, limitar a taxa na qual os pacotes enviados por clientes são aceitos e, conseqüentemente, processados pelo servidor. A decisão de aceitar ou não o pacote em questão é tomada com base em alguma medida de desempenho (ou uma combinação delas) do sistema. De maneira semelhante, esta definição se aplica aos mecanismos de controle de admissão baseados em requisições.

Além de um bom comportamento quando submetido a uma sobrecarga, é desejável que um servidor Internet implemente funcionalidades referentes à diferenciação e qualidade de serviço. Seguindo essa tendência, Jamjoom *et al.* apresentam em [20] o protótipo do QGuard, definido como uma combinação de mecanismos de proteção de sobrecarga em servidores baseados na Internet. Tais mecanismos operam no nível do kernel e nos níveis intermediários (*middleware*), fazendo o controle da taxa de pacotes que é submetida às aplicações executadas no servidor. A proposta do QGuard, além da proteção contra sobrecarga e da garantia de níveis de qualidade de serviço, inclui proteção contra alguns tipos de ataques de negação de serviço.

O controle de admissão definido pelo QGuard [20] é baseado em um mecanismo chamado de *modelador de tráfico*. Neste mecanismo, a fim de promover a diferenciação de serviço, são definidas classes de tráfego que representam as aplicações do servidor. Cada classe pode ter um conjunto de regras associadas. Tais regras controlam quais serviços podem ser acessados por uma classe, bem como a taxa máxima de requisições que uma classe pode tratar.

Além deste, outro importante mecanismo do QGuard é o modulo de monitoramento de carga do sistema. Implementado no nível do kernel, esse mecanismo pode ser configurado para monitorar diversos recursos do sistema (como CPU, memória, disco, rede, etc.), permitindo uma indicação mais precisa de um possível estado de sobrecarga. Por fim, os resultados da avaliação do QGuard atingiram as expectativas dos autores, pois os níveis de qualidade e diferenciação de serviço foram alcançados com perdas pequenas (2–3%) na vazão do servidor; o sistema QGuard se mostrou eficiente contra ataques de negação de serviço.

Outro esquema de controle de admissão que também oferece garantias de qualidade de serviço é proposto por X. Chen *et al.* em [8]. Os autores apresentam um algoritmo de controle de admissão chamado PACERS (*Periodic Admission Control based on Estimation of Request rate and Service time*), cujo objetivo é impor limites no tempo de resposta para requisições que chegam a um sistema que esteja exposto a uma grande variabilidade de carga. Neste esquema, a admissão de uma requisição é feita comparando-se a capacidade disponível do sistema com o tempo de serviço estimado da requisição. A estimativa do tempo de serviço de uma requisição é baseada no seu tipo (estática, dinâmica, vídeo, áudio, etc.). Com o objetivo de tratar a imprecisão do modelo, tais estimativas são dinamicamente ajustadas por uma arquitetura de fila dupla. Além disso, a fim de prover diferenciação de serviço, a alocação de recursos do sistema pelo PACERS é feita levando em conta a classe de prioridade de uma requisição, tendo como objetivo limitar o atraso das requisições de mais alta prioridade em detrimento das requisições de baixa prioridade.

Para avaliar o algoritmo PACERS apresentado em [8], os autores utilizaram um modelo de simulação orientado a eventos juntamente com uma carga de trabalho baseada em registros de acesso reais. Os resultados, obtidos para apenas duas classes de prioridade, mostram que o algoritmo não deteriora de forma significativa a vazão do sistema para tarefas de alta prioridade quando submetido a cargas elevadas. Neste mesmo cenário, foi mostrado que o algoritmo proposto mantém a vazão do sistema para tarefas de alta prioridade em detrimento do tempo médio de atraso de resposta, mas ainda assim deixando este atraso dentro de determinados limites.

Ainda sobre o foco da diferenciação de serviço combinada com controle de sobrecarga, Welsh e Culler [43] apresentam um conjunto de técnicas para controle de admissão adaptativo baseado na arquitetura SEDA [44]. SEDA é uma de arquitetura para serviços Internet de larga escala que demandam suporte para concorrência massiva e que exigem um bom comportamento quando submetidos a grandes cargas de trabalho. Segundo o modelo da arquitetura SEDA, o sistema é decomposto em um conjunto de estágios interligados através de filas de eventos, nos quais cada estágio processa um determinado aspecto de uma requisição. A proposta dos autores é aplicar o controle de admissão em cada estágio,

gerenciando a sobrecarga através de medições constantes do desempenho do estágio. Da mesma forma, a diferenciação de serviço (baseada em classes de usuários) também é feita por estágio. Como complemento, os autores utilizam degradação de serviço como uma das técnicas de controle de sobrecarga.

Diferentemente de outros trabalhos, os autores que utilizam a arquitetura SEDA consideram o $90^{\underline{o}}$ percentil do tempo de resposta como medida de desempenho do servidor e o utilizam como parâmetro para ajustar a taxa de admissão de requisições em cada estágio. O trabalho mostra a avaliação do sistema sobre duas aplicações (um servidor de Web *mail* e um servidor Web), mostrando que o gerenciamento de sobrecarga proposto é efetivo quando aplicado a populações crescentes de usuários, bem como a picos de carga.

Uma inovativa abordagem para o controle de sobrecarga é apresentada por Nahum *et al.* em [29]. Partindo do fato de que muitos sistemas de comércio eletrônico são compostos por uma arquitetura de três camadas (um servidor Web, um servidor de aplicação e um servidor de banco de dados), os autores propõem um mecanismo de controle de admissão e escalonamento de requisições que opera de forma transparente em tal arquitetura. Diferindo de muitas abordagens que exigem alterações no sistema operacional ou no servidor Web utilizados, o método proposto foi implementado externamente como um *proxy* (chamado de *Gatekeeper*) instalado entre o servidor de aplicação e o servidor de banco de dados, dispensando qualquer tipo de alteração na aplicação em questão. Outra diferença com relação às demais abordagens refere-se ao tipo de conteúdo tratado pelo Gatekeeper, que também considera páginas Web com conteúdo dinâmico (Java *servelets*).

O controle de admissão feito no Gatekeeper toma como base a carga estimada que um dado tipo de requisição (*servelet*) impõe sobre o sistema. Quando uma requisição solicita uma conexão com o banco de dados, a carga estimada da requisição é examinada. Se tal carga não ultrapassar a capacidade máxima do sistema (medida previamente de forma *offline*), a requisição é admitida, caso contrário ela é colocada em uma fila de admissão governada pela política de SJF (*Shortest Job First*). Os resultados da avaliação do esquema proposto mostraram que o Gatekeeper aumentou a vazão de pico em 10% e reduziu o tempo médio de resposta por um fator de 14 pontos percentuais quando

comparado ao sistema original sem controle de admissão e escalonamento.

Os trabalhos descritos acima demonstram a variedade de esquemas que empregam o controle de admissão com um meio de descartar a carga. A maioria absoluta dos trabalhos pesquisados considerou o controle de admissão baseado em requisições.

Embora a abordagem que considera requisições seja a mais popular, ela pode não ser tão eficiente quanto uma abordagem que considere pacotes. Em primeiro lugar, quando requisições são consideradas, o tempo para se obter a requisição propriamente dita depende de todo o processamento dos protocolos inferiores que a transportam. Segundo, os mecanismos baseados em requisições costumam atuar no nível de aplicação, onde podem não ter toda a prioridade de execução que deveriam, ficando mais suscetíveis à deterioração dos recursos do sistema. Por outro lado, um controle de admissão baseado em pacotes que seja implementado no kernel do sistema – o único lugar onde é possível atuar diretamente sobre os pacotes – pode oferecer uma solução mais robusta e eficiente. Os pacotes são descartados mais cedo e menos recursos são consumidos. No entanto, é muitas vezes preferível alterar a aplicação do que o sistema operacional em que ela opera.

Também é claro que as abordagens que consideram pacotes são mais flexíveis, pois estas sugerem pouca ou nenhuma dependência em relação ao protocolo que a aplicação de alto nível utiliza, podendo ser aplicadas em servidores em geral (servidores Web, de e-mail, etc.). Por outro lado, o uso de um mecanismo de controle de admissão baseado em pacotes muitas vezes dificulta (podendo até mesmo impedir) que as decisões de aceite sejam baseadas em parâmetros específicos da aplicação, os quais são necessários para implementar um controle pontual de qualidade e diferenciação de serviço.

2.2.2 Controle de Admissão Baseado em Sessões

O protocolo HTTP é basicamente um protocolo sem estado [22]. Uma aplicação que utilize este protocolo (por exemplo, um servidor Web) não precisa manter qualquer informação de estado sobre requisições anteriores ou futuras, o que implica que o protocolo não contém o conceito de *sessão*. No entanto, muitos serviços oferecidos na Web necessitam deste conceito. Por exemplo, considere um *site* de compras *on-line*. Uma funcionalidade

comum neste tipo de *site* é o "carrinho de compras", no qual o usuário pode reunir seletivamente os itens que deseja comprar, como livros ou CDs . Tal funcionalidade só é possível caso exista alguma forma de "lembrar" dos itens escolhidos pelo usuário enquanto este visita as páginas do *site*.

Este e outros tipos de necessidades motivaram o desenvolvimento de mecanismos para administrar o estado no protocolo HTTP. Um destes mecanismos se vale de pequenas quantidades de estado enviadas pelo servidor ao navegador do usuário, chamadas de *cookies* [23], a fim de manter informações de contexto entre várias requisições HTTP. Além dos *cookies*, outra maneira de manter estado no protocolo HTTP é encapsular as informações pertinentes ao serviço em questão na própria URL na forma de parâmetros, gerando páginas dinamicamente a partir destes parâmetros [28].

Independente da forma escolhida para manter o estado no protocolo HTTP, um mecanismo de controle de admissão baseado em sessões deve ter a capacidade de distinguir a importância de uma requisição dentro de um conjunto de requisições relacionadas, pois a decisão de aceitação ou não desta requisição, e a própria eficiência do mecanismo, depende disto. Voltando ao exemplo do carrinho de compras, as requisições relacionadas com a seleção de itens podem ter menor prioridade do que as requisições referentes ao pagamento dos itens. Da mesma forma, é sensato privilegiar as requisições dos usuários que estão com itens no carrinho, ou seja, dar maior prioridade aos usuários que tem maior probabilidade de efetivar a compra do que aqueles que estão apenas navegando pelo site.

O primeiro trabalho a considerar sessões em um esquema de controle de admissão foi proposto por Cherkasova e Phaal [9]. Os autores defendem que, segundo a perspectiva do usuário, a medida real de desempenho de um servidor Web é a sua habilidade de processar uma seqüência de requisições necessárias para completar uma transação. Como base nesse conceito, os autores definiram uma nova métrica de desempenho chamada *useful server utilization*, que representa o tempo que o servidor gasta processando apenas as requisições de sessões que serão completadas com sucesso.

Em conjunto com esta nova métrica, os autores também definem o conceito de *qualidade de serviço Web* como sendo os níveis de serviço necessários para completar sessões

Web. Segundo esta definição, as características de um servidor Web que oferece tais níveis de serviço são:

- O servidor provê uma chance justa para qualquer sessão aceita ser completada, independentemente do tamanho da sessão;
- O servidor maximiza a métrica *useful server utilization*;
- O servidor minimiza o número de sessões abortadas, sendo este número idealmente zero.

Além disso, o mecanismo de controle de admissão proposto em [9] considera a métrica utilização da CPU, medida em intervalos pré-definidos de tempo, como base para a aplicação da política de admissão. Quando esta medida ultrapassa um dado limiar, todas as requisições para novas sessões são rejeitadas. Para cada requisição rejeitada, uma mensagem explícita deve ser enviada para o usuário. Neste ponto cabe observar que, conforme exposto em [11], o principal ponto de contenção de um servidor Web é a fila IP do subsistema de rede. Logo, a utilização da CPU como único indicativo de sobrecarga é questionável. Os autores também discutem a importância do envio de mensagens explícitas de rejeição ao cliente durante uma situação de sobrecarga, afirmando que tal procedimento pode poupar o servidor de uma sobrecarga adicional causada por novas tentativas dos clientes.

Entre os principais resultados do modelo simulado em [9], estão:

- Um servidor sobrecarregado sem o controle de admissão proposto apresenta perdas significativas de vazão (definida em termos de sessões completadas). Esta perda é muito maior se comparada à vazão definida em termos de requisições completadas. Além disso, tal servidor discrimina sessões longas em relação às sessões menores.
- O custo adicional imposto pela rejeição explícita de requisições corresponde a menos de 10% da carga do servidor. Tal resultado se aplica somente a casos nos quais a capacidade do servidor não está completamente comprometida, ou seja, a carga não é tão alta a ponto de impedir que o servidor envie as mensagens de rejeição.

- Um servidor que utilize o controle de admissão proposto atende aos requisitos de qualidade de serviço Web.

Uma possível expansão do modelo de controle de admissão baseado em sessões é dividir uma sessão em estágios (tela inicial, listagem, compra, etc.) e empregar um método de controle adicional sobre esses estágios como, por exemplo, escalonamento. Esta abordagem é utilizada em [7] e [6].

Em [7], H. Chen e Mohapatra sugerem que a utilização de um mecanismo de controle de admissão simples assistido por um algoritmo de escalonamento de requisições eficiente é uma alternativa melhor em uma situação de sobrecarga do que apenas o controle de admissão isolado. No modelo proposto, quando uma requisição pertencente a uma sessão é admitida no sistema, esta é classificada e colocada em uma das filas que representam os estágios da sessão. Sobre estas filas, um esquema de escalonamento dinâmico, chamado de DWFS (*Dynamic Weighted Fair Scheduling*), controla o acesso aos recursos do sistema. O objetivo deste algoritmo é maximizar o que os autores chamaram de *produtividade do servidor*, definida como o número de requisições completadas menos o número de requisições abortadas em um intervalo de tempo definido, um conceito semelhante ao que foi chamado de *useful server utilization* em [9].

A fim de comprovar a viabilidade do esquema sugerido, o trabalho em [7] apresenta uma implementação do algoritmo DWFS no servidor Web Apache [15]. Os resultados da avaliação desta implementação com a utilização de uma carga artificial mostram que o algoritmo proposto melhora em mais de 50% a produtividade do servidor quando submetido a cargas de trabalho pesadas.

Carlström e Rom [6] utilizam a técnica de escalonamento GPS (*Generalized Processor Sharing*) em conjunto com o controle de admissão baseado em sessões. No esquema proposto pelos autores, sessões são divididas em estágios nos quais as requisições têm acesso a recursos específicos do sistema. A transição entre estágios é definida por uma matriz de probabilidades. Além disso, uma função dependente da aplicação premia transições entre estágios. Cada estágio possui uma fila tipo FIFO onde as requisições aguardam para acessar recursos do sistema. Quando um recurso se torna disponível, um algoritmo de

escalonamento aloca o recurso para a requisição que está na primeira posição da sua respectiva fila de estágio. A escolha da requisição leva em conta o valor dado pela função de premiação, sendo assumido que o algoritmo de escalonamento emula a política GPS. Em suma, o objetivo principal desse esquema é manter o tempo de resposta baixo para sessões que geram os maiores valores de premiação. Isto implica que em uma situação de sobrecarga sessões em estágios que não geram grandes valores de premiação experimentarão maiores atrasos.

Para avaliar o modelo proposto em [6], os autores utilizaram análise numérica que considerou um *site* de comércio eletrônico fictício com quatro estágios. Nesta avaliação, a política GPS foi comparada à política FCFS, e a GPS se mostrou mais eficiente, gerando menores degradações de desempenho. O maior problema deste esquema está na necessidade de definir configurações e ajustar parâmetros externos a fim de obter melhores resultados, o que pode não se mostrar atrativo se comparado a outras técnicas menos parametrizadas. Da mesma forma, o modelo se mostra extremamente complexo e com várias premissas. Logo resultados provindos de análises numéricas e simulações podem não ser tão significativos quanto uma análise feita em um sistema real.

Diferente das abordagens vistas anteriormente, Voigt e Gunningberg apresentam em [40] um controle de admissão baseado em sessões no nível de kernel. O trabalho em questão é uma expansão da arquitetura proposta por Voigt *et al.* em [41]. O objetivo da expansão é implementar um esquema de controle de admissão que considere requisições HTTP com conexões persistentes, utilizando *cookies* para classificar a importância das conexões. Como definido em [27], conexões persistentes permitem que clientes HTTP enviem várias requisições em uma mesma conexão TCP. Além de tratar conexões persistentes, os autores consideram também mecanismos para prover diferenciação de serviço na forma de um conjunto de filtros e regras, os quais podem utilizar informações provindas do processamento de pacotes TCP (diferenciação no nível de rede) e informações coletadas dos cabeçalhos HTTP (*cookies*, URL e tipo da requisição). Este esquema difere da proposta inicial descrita em [9] (ver acima) por não recusar conexões de forma indiscriminada, pois em uma situação de sobrecarga definida com base na utilização da CPU, apenas conexões

que não sejam consideradas importantes são abortadas.

Os autores em [40] apontam os seguintes pontos como limitações da implementação do mecanismo proposto no sistema operacional Linux versão 2.4:

- Devido ao processamento de cabeçalhos HTTP no nível de kernel, apenas a última requisição da sessão pode ser uma requisição dinâmica (*script* CGI).
- O controle de admissão considera *cookies* para classificar as conexões. Assim, caso um usuário desabilite *cookies* em seu navegador, a conexão de tal usuário será invariavelmente abortada durante um período de sobrecarga.
- As simulações da arquitetura consideraram o tempo de pensar do usuário como sendo zero.

Por fim, uma análise geral dos mecanismos de controle de admissão apresentados nesta seção indica que estes mecanismos estão voltados para serviços que necessitam manter estado entre as transações HTTP, como, por exemplo, *sites* de comércio eletrônico, bancos e leilões *on-line*. É comum que em tais serviços sejam utilizados *cookies* para implementar a suas funcionalidades. Contudo, o uso de *cookies* como um meio de gerenciar o estado em HTTP é muita vezes desencorajado, pois pode trazer implicações de segurança e privacidade no que se refere aos dados dos usuários [28]. Também é certo que a aplicação prática das abordagens que consideram sessões é bem mais restrita do que aquelas que consideram pacotes ou requisições, pois nem todo serviço disponível na Web necessita de manutenção de estado para funcionar como, por exemplo, *sites* de notícias, de empresas, de instituições educacionais, dentre outros.

2.3 Controle de *Livelock*

Este seção apresenta um efeito comum em sistemas orientados à interrupção, conhecido como *receive livelock*, bem como os mecanismos propostos na literatura que visam tratar este problema.

2.3.1 O Efeito *Receive Livelock*

A grande maioria dos sistemas operacionais modernos baseia-se em uma arquitetura orientada a interrupções para tratar eventos de E/S [38]. A natureza orientada a interrupções destes sistemas, embora eficiente, pode trazer alguns problemas de escalonamento em situações de sobrecarga, visto que o tratamento de interrupções tem prioridade sobre outras tarefas em execução. Isto é especialmente verdade quando se trata do subsistema de rede.

A Figura 2.2 mostra um esquema clássico de processamento de pacotes de rede adotado por diversos sistemas operacionais[2]. Quando um pacote chega à interface de rede, este evento é sinalizado ao sistema através de uma interrupção de hardware. O processamento desta interrupção tem prioridade sobre qualquer outra tarefa em execução no sistema e, caso não exista outra interrupção de mesma prioridade sendo tratada, a CPU é alocada imediatamente para a interrupção em questão. No contexto desta interrupção, uma rotina pertencente ao *driver* de rede faz um processamento mínimo sobre o pacote recebido e o coloca em uma fila de entrada de pacotes IP. Todos os pacotes IP processados pelo *driver* de rede são colocados nesta fila. Então o *driver* de rede agenda uma interrupção de software para que o pacote seja posteriormente processado.

No contexto do tratamento da interrupção de software, uma rotina específica de processamento de protocolo retira o pacote da fila de entrada IP e, dependendo do conteúdo do pacote, efetua uma das seguintes ações:

- Encaminha o pacote para a fila de redirecionamento (IP *forwarding*) de uma interface específica;

[2]A fim de simplificar esta discussão, apenas o processamento dos protocolos IP/TCP/UDP será considerado.

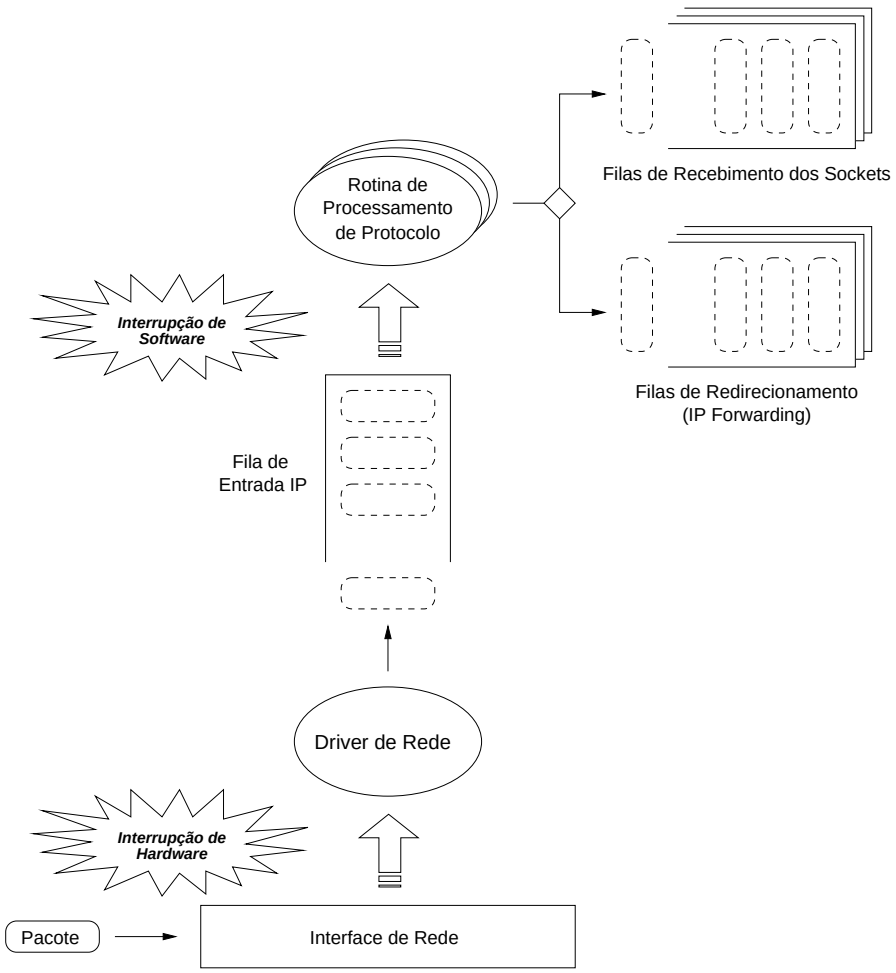


Figura 2.2: Esquema clássico de processamento de pacotes IP.

- Após uma potencial remontagem de pacotes IP fragmentados, chama a rotina apropriada para o processamento de protocolo de mais alto nível (TCP ou UDP), a qual finalmente passa o pacote para a fila de recebimento de um *socket* específico de onde uma chamada de sistema (geralmente do tipo *receive()*), quando invocada, pode copiar os dados para uma região de memória local à aplicação receptora.

Embora o tratamento de interrupções de software tenha prioridade sobre as demais tarefas do sistema, elas ainda têm uma prioridade inferior às interrupções de hardware. Isto significa que as rotinas de processamento de protocolo e qualquer outro processo de usuário são interrompidos sempre que um novo pacote chega ao sistema. Tal situação implica em uma anomalia no escalonamento de tarefas: caso pacotes de rede cheguem a uma taxa muito alta (maior que a capacidade do sistema), o sistema gastará todo o seu

tempo processando interrupções; nenhum recurso do sistema será destinado ao processamento de pacotes por parte das rotinas específicas de protocolo nem ao processamento das tarefas no nível de aplicação (por exemplo, a execução de um *script* CGI em um servidor Web). Conseqüentemente, as filas de entrada (IP e *sockets*) e a fila redirecionamento ficarão lotadas e pacotes serão processados apenas para serem posteriormente descartados, reduzindo a vazão do sistema a zero.

A condição descrita acima é definida como *receive livelock* (ou simplesmente *livelock*) [34]: o sistema não está em *deadlock*, porém não apresenta qualquer progresso útil, visto que todos os seus recursos estão sendo consumidos pelo processamento de interrupções. Quando a taxa de chegada diminui suficientemente, o sistema volta à apresentar resposta em termos de vazão. Logo, um estado de *livelock* caracteriza um sistema em sobrecarga no qual o ponto de contenção é a CPU.

Em sistemas bem projetados espera-se que a taxa com que os pacotes são processados e liberados para as aplicações destino, isto é, a vazão do subsistema de rede, acompanhe o tráfego de entrada até um ponto chamado de Taxa de Chegada Máxima Livre de Perdas (TCMLP) e se mantenha relativamente constante a partir deste ponto [26]. Contudo, em sistemas propensos a *livelocks*, cargas acima da TCMLP fazem com que o sistema entre em colapso, elevando rapidamente o tempo de resposta e reduzindo a vazão à zero. A Figura 2.3, adaptada de [16], ilustra de forma conceitual esta situação.

O principal problema com as arquiteturas puramente orientadas à interrupções é a importância dada ao tráfego que chega ao sistema, em uma hierarquia de processamento, que pode ser resumida em:

1. grande prioridade para a rotina que captura e transfere os pacotes para a memória principal;

2. prioridade secundária para a rotina de processamento de protocolo;

3. baixa prioridade para as aplicações que efetivamente são as destinatárias das mensagens.

Além disso, o descarte de pacotes usado como um meio de aliviar a sobrecarga do

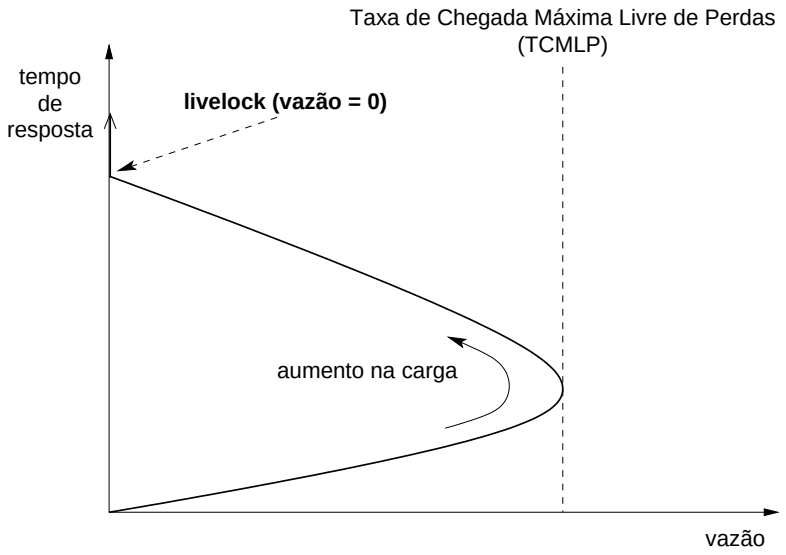


Figura 2.3: Curva de *livelock*.

sistema ocorre apenas depois de uma significativa quantidade de recursos do sistema ter sido investida no pacote rejeitado.

Também é importante ressaltar que *livelocks* não ocorrem na transmissão de pacotes. Primeiro, porque a transmissão de pacotes tem menor prioridade que a recepção, e segundo, porque a maior parte do processo de enviar os dados escritos em um *socket* é executado no contexto da aplicação que realizou a chamada de sistema (*send()*) sobre o *socket*, estando assim sujeito ao escalonador do sistema.

Como pôde ser notado, a formação de *livelocks* está estreitamente relacionada com a capacidade de processamento do computador que recebe o tráfego, com a intensidade do tráfego e principalmente com a forma com que os pacotes de entrada são tratados pelo sistema.

Além disso, como já observado na área de medição de redes [21, 33], há alguns anos a velocidade das redes vem aumentando muito mais rapidamente do que a velocidade dos processadores, das memórias e do barramento de E/S, os quais se apresentam como elementos fundamentais para a determinação da capacidade de um sistema. Tal desigualdade contribui diretamente para a ocorrência de *livelocks*.

Comparando especificamente a velocidade das redes com a velocidade dos processadores, há uma concepção geral de 1MHz de CPU é o ideal para comportar 1Mb/s de vazão

de rede [18, 24]. Seguindo esse raciocínio, um sistema capaz de comportar uma interface de rede de 10Gb/s (já disponível no mercado[3]) deveria ter uma CPU de 10GHz, sendo que CPUs com este poder de processamento provavelmente não serão comercializadas tão cedo. Mas mesmo que existam casos em que é viável trabalhar com CPUs adequadas à velocidade da rede, questões econômicas impedem que isto seja feito de forma massiva.

A questão é que, com redes cada vez mais velozes frente a computadores não tão potentes, aliado ao crescimento substancial dos usuários com acesso a grande rede mundial, provocar um estado de *livelock* um sistema se torna uma tarefa ainda mais simples. Tudo isso motiva a proposta de novas arquiteturas para o processamento de pacotes, tema que é tratado na próxima seção.

2.3.2 Soluções para o Efeito *Receive Livelock*

O primeiro trabalho a descrever o efeito *receive livelock* foi publicado no início da década de noventa por Ramakrishnan [34]. Neste trabalho, o autor já apontava como principais causas do efeito *receive livelock* questões como redes de alta velocidade conectadas a computadores não tão potentes, sobrecargas transientes e subsistemas de rede mal projetados. Ramakrishnan descreve as seguintes alternativas para os problemas causados pela arquitetura orientada a interrupções, quando utilizada no processamento de pacotes de rede, as quais serão informalmente definidas como *técnicas de controle de livelock* [4]:

- limitar a taxa de interrupções gerada pela interface de rede;
- utilizar um esquema de espera ocupada (*polling*) no lugar de interrupções, no qual o *driver* de rede ficaria "escutando" a espera de pacotes para serem transmitidos ou recebidos;
- empregar *threads* nas tarefas de processamento de pacotes de rede, bem como no resto do sistema, limitando o número de ciclos de CPU utilizado em cada *thread*.

[3] http://www.intel.com/support/network/adapter/pro10gbe/pro10gbelr/index.htm.

[4] Da mesma forma, para fins de simplicidade, qualquer implementação de técnicas que objetivam evitar situações de *livelock* será referenciada como um *mecanismo de controle de livelock* ou simplesmente *controle de livelock*.

Contudo, tais técnicas também têm certas limitações e problemas associados. Impor restrições sobre a taxa de interrupções pode limitar a possibilidade de *livelocks* no recebimento de pacotes, mas ainda assim é uma proposta extremamente dependente de hardware, pois baseia-se na capacidade do sistema em prover mecanismos para diminuir e, em certos casos, desabilitar interrupções. A técnica de espera ocupada teoricamente adiciona uma latência maior no processamento de pacotes, se comparada ao esquema tradicional de interrupções, o que é compensado por um maior controle da execução das rotinas de processamento de protocolo. A técnica que emprega *threads* em vários níveis do sistema, apesar de também trazer a vantagem relacionada ao maior controle das rotinas associadas ao processamento de pacotes, exige mecanismos mais sofisticados em termos de escalonamento, hardware (processadores com registradores de alta resolução para contar os ciclos de CPU), bem como a disponibilidade de *kernel threads*, que é uma funcionalidade não implementada em alguns sistemas operacionais.

O primeiro mecanismo de controle de *livelock* foi apresentado por Mogul e Ramakrishnan [26]. Os autores implementaram em um sistema 4.2BSD uma abordagem híbrida que utiliza o esquema tradicional para o tratamento de interrupções aliado à técnica de espera ocupada. Nesta abordagem, interrupções causadas pela interface de rede servem apenas para ativar um *kernel thread* que irá manter o sistema em espera ocupada, verificando todas as interfaces de rede em busca de pacotes pendentes. Enquanto este *thread* estiver ativo, as interrupções ficam desabilitadas. Quando o *thread* identifica que uma interface tem pacotes a serem recebidos, o controle é passado para uma rotina de tratamento de pacotes (*callback procedure*), juntamente com um limite (*quota*) para o número de pacotes que podem ser processados, impedindo assim que uma única interface monopolize a CPU.

A rotina de tratamento de pacotes, por sua vez, passa o pacote recebido diretamente para a rotina de processamento do protocolo IP, eliminando a necessidade da fila de entrada IP. Durante o período em que as interrupções estiverem desabilitadas, qualquer pacote que chegue ao sistema e que não possa ser armazenado nas estruturas de memória pertencentes à interface de rede é descartado (*descarte antecipado*). As interrupções são habilitadas novamente quando não houver mais pacotes a serem tratados ou quando o

limite de pacotes processados for alcançado. Um esquema similar é utilizado para a transmissão de pacotes.

A arquitetura para o subsistema de rede descrita acima também conta com um mecanismo que monitora alguns parâmetros do sistema, como o tamanho das filas dos *sockets* e os ciclos de CPU gastos no recebimento de pacotes, com o objetivo de garantir justiça na alocação de recursos entre o processamento de rede e aplicações de usuários. Neste esquema, caso um dos parâmetros monitorados atinja o seu respectivo limiar, o processamento de pacotes é interrompido para dar chance aos processos no nível de aplicação completarem as suas tarefas. Por fim, os resultados dos experimentos realizados mostram que a solução proposta é eficaz contra situações de sobrecarga, mantendo a vazão relativamente constante mesmo quando o tráfego de entrada supera a capacidade do sistema, evitando assim situações de *livelock*.

Baseando-se na solução descrita acima, Hansen e Jul apresentam em [17] um subsistema de rede voltado para sistemas multiprocessados baseados na plataforma Windows NT. Tal solução opera em dois níveis: interrupções e espera ocupada. Em condições normais de carga, o sistema opera no modo tradicional, isto é, com interrupções. Quando a quantidade de tempo gasta no processamento de interrupções, a qual é constantemente monitorada, ultrapassa um dado limiar, o sistema alterna para o modo de espera ocupada, retornando para o modo de interrupções apenas quando não houver mais pacotes a serem processados. Os resultados mostram que a sobrecarga adicionada pelo esquema de dois níveis de operação é negligenciável se comparada ao esquema tradicional que utiliza apenas interrupções.

Também inspirados pelo trabalho de Mogul e Ramakrishnan descrito anteriormente, Salim *et al.* apresentam em [36] uma nova *Application Programming Interface* (API) para o desenvolvimento de *drivers* de rede em Linux chamada NAPI[5]. Entre os objetivos desta nova API estão (1) a redução das interrupções em situações de sobrecarga, (2) o descarte antecipado de pacotes e (3) a independência de hardware específico. O primeiro objetivo é atingido através da utilização do esquema de espera ocupada, o qual é implementado de

[5]A NAPI está disponível em caráter experimental nas distribuições Linux desde o kernel 2.4

forma muito similar a proposta original mostrada em [26]. O segundo objetivo é alcançado com a ajuda do mecanismo de *Direct Memory Access* [6] (DMA) da interface de rede. Todo pacote que chega à interface de rede é posto via DMA em uma lista circular (DMA *ring*) localizada em memória principal para só então ser gerada a interrupção referente a este pacote. Quando as interrupções estiverem desabilitadas por causa da espera ocupada, todo pacote que não possa ser colocado na lista circular do mecanismo de DMA será descartado "silenciosamente" (descarte antecipado). O terceiro objetivo também é alcançado, pois a antiga API para processamento de pacotes ainda está disponível.

Outro objetivo da NAPI, embora fora do escopo deste trabalho, é impedir a reordenação de pacotes, um efeito comum em sistemas multiprocessados. Este objetivo é cumprido através do próprio funcionamento dos componentes da nova API. O primeiro é a lista circular do mecanismo de DMA. Em vez de múltiplas filas IP por processador, há apenas esta lista circular que é acessada somente por um processador: aquele que for sinalizado primeiro pela interrupção que inicia a espera ocupada. Este esquema também traz a vantagem de, em sistemas com múltiplas interfaces de rede, cada processador acessa de forma mutuamente exclusiva as interfaces com pacotes pendentes. Tudo isso reforça a serialização dos pacotes, conforme comprovado pelos resultados mostrados no trabalho. Contudo, tais técnicas não tiram vantagem do paralelismo dos diversos processadores. Para resolver este problema, Lemoine *et al.* apresentam em [24] uma proposta muito semelhante a NAPI específica para sistemas multiprocessados, visando implementar um subsistema de rede paralelo e imune a *livelocks*. Os resultados desta nova arquitetura, quando comparada a NAPI, mostram significativos ganhos de desempenho.

Diferentemente das abordagens baseadas em espera ocupada, uma interessante arquitetura para o subsistema de rede é proposta por Druschel e Banga em [13]. Os autores chamaram esta nova arquitetura de *Lazy Receiver Processing* (LRP), cujo principal objetivo é prover maior justiça na alocação de recursos despendidos no processamento de pacotes, bem como maior estabilidade em situações de sobrecarga. Na arquitetura LRP, a fila de entrada IP, conforme apresentada na Seção 2.3.1, é substituída por diversas filas

[6] A disponibilidade do recurso de DMA na interface de rede é um dos pré-requisitos para se utilizar a NAPI.

(*channels*), uma para cada *socket* criado no sistema. Tal estrutura reduz a probabilidade de um pacote destinado a uma aplicação seja atrasado ou descartado porque o tráfego de entrada de outra aplicação é mais intenso e sobrecarregou os recursos do sistema.

Na proposta original da arquitetura LRP, implementada em um sistema BSD, os pacotes são demultiplexados para as filas dos *sockets* pela própria interface de rede[7]. Caso não haja espaço na fila, o pacote é descartado sem perturbar o sistema (descarte antecipado). Uma vez na fila de um *socket* específico, o pacote é processado de forma "preguiçosa", ou seja, somente quando o pacote for solicitado pela aplicação, sempre que a semântica do protocolo carregado por pacote permitir. Além disso, o processamento de protocolo é executado com a mesma prioridade da aplicação que está recebendo os pacotes, atribuindo o tempo de CPU à aplicação receptora. Isto implica em mais justiça entre as diversas aplicações, visto que o uso de CPU influencia a prioridade que o escalonador do sistema atribui a um processo.

Os experimentos conduzidos com a arquitetura LRP mostram que a TCMLP, sob a configuração de hardware adotada, é 44% maior do que a TCMLP de um sistema BSD tradicional. Em especial, foram realizados experimentos utilizando-se o servidor Web NCSA com o objetivo de testar a eficiência da arquitetura LRP sobre uma aplicação real. Para este teste, um conjunto de oito clientes HTTP realizavam constantemente requisições a uma estação que executava o servidor Web. Um segundo programa cliente tratava de saturar esta estação tentando continuamente estabelecer conexões falsas com um segundo servidor Web, o qual também estava sendo executado na estação servidora. Os resultados mostram que, com a arquitetura LRP, a vazão do sistema diminui vagarosamente, operando a 50% da vazão máxima, mesmo sob a maior intensidade de tráfego considerada. Por outro lado, o sistema sem a LRP pára de responder qualquer requisição HTTP quando a intensidade do tráfego chega a metade do valor máximo utilizado. Contudo, há uma limitação associada a estes resultados, pois eles foram realizados para um número restrito de clientes. Conforme apontado pelos autores, a proposta original da LRP pode não esca-

[7]Neste caso supõe-se que a placa de rede possui uma CPU embutida que pode ser programada para executar esta tarefa. Para interfaces de rede que não possuem este recurso, a demultiplexação deve ser feita pelo próprio sistema.

lar bem para um número muito grande de conexões ativas, visto que cada conexão teria a sua própria fila de entrada (*channel*) na qual a interface de rede estaria despendendo recursos.

Outra proposta de arquitetura para o subsistema de rede é apresentada por Brustoloni *et al.* em [5]. Esta outra arquitetura é chamada de *Signaled Receiver Processing* (SRP) e possui algumas semelhanças com a LRP como, por exemplo, o esquema de filas de entrada de pacotes por aplicação que substitui a fila de entrada IP. Na verdade, os autores apresentam a arquitetura SRP como uma alternativa à LRP, apontando diversas dificuldades relacionadas à implementação desta última. Enquanto a arquitetura LRP é totalmente transparente para as aplicações, a SRP provê controle total do processamento de pacotes por parte da aplicação através de um esquema baseado em sinais. Implementada em um sistema Eclipse/BSD (uma variante do sistema FreeBSD), a arquitetura SRP não exige qualquer requisito especial em termos de hardware ou alterações no *driver* da interface de rede. Apenas os componentes que atuam ao processamento de protocolo foram alterados.

Na arquitetura SRP, todo processo (aplicação de usuário ou processo do sistema) que receba pacotes possui uma fila de pacotes não processados. Os pacotes, após serem tratados pelo *driver* de rede, são demultiplexados por uma rotina especial e colocados nestas filas. Cada fila tem um limite de pacotes determinado por parâmetros específicos do seu respectivo processo receptor. Toda vez que um pacote é posto em uma destas filas, um sinal é emitido para o processo ao qual a fila pertence. A ação padrão deste sinal é retirar o pacote da fila e realizar o processamento de protocolo (processamento assíncrono). Entretanto, o processo receptor pode escolher sincronizar este processamento por capturar, bloquear ou ignorar o sinal. Neste último caso, o processamento de protocolo ocorrerá apenas quando o processo solicitar explicitamente o pacote através de uma chamada de sistema (*receive()*).

Embora os criadores da arquitetura SRP a definam como um mecanismo de controle de *livelock*, e os resultados experimentais mostrem que ela apresenta um comportamento estável em situações de sobrecarga, a arquitetura não leva em conta uma das principais técnicas de controle de *livelock*: a limitação da taxa de interrupções gerada pela interface

de rede (interrupções de hardware). Na verdade, a arquitetura SRP limita apenas a quantidade de recursos utilizados no processamento de pacotes por descartá-los caso não haja espaço na fila de um processo receptor, mas não a taxa de interrupções de hardware. Por exemplo, em casos em que o tempo entre chegadas dos pacotes for menor do que o tempo que a rotina utilizada pela SRP leva para demultiplexar os pacotes, o sistema certamente entrará em *livelock*.

Dada a observação acima, é fundamental que fique claro que quando se trata da importância dos componentes responsáveis pela ocorrência de *livelocks* em sistemas, o papel principal é destinado às interrupções de hardware e a prioridade dada a elas; o processamento de protocolo, realizando no contexto de uma interrupção de software nas arquiteturas tradicionais, é apenas um coadjuvante. Logo, mudar apenas o funcionamento deste último componente não soluciona completamente o problema de *livelocks*, apenas posterga a sua ocorrência.

Além das sofisticadas técnicas descritas ao longo desta seção, também é crescente o número de interfaces de rede de alta velocidade que implementam mecanismos próprios para moderar a taxa de interrupções, conhecidos como mecanismos de *moderação de interrupções*[8]. Tais mecanismos podem evitar ou adiar *livelocks* por limitar a taxa na qual as interrupções são impostas ao sistema. Em geral, estes mecanismos contam com dois elementos principais: um contador de pacotes recebidos/transmitidos e o seu respectivo temporizador. A combinação destes elementos permite que a interface de rede agrupe vários pacotes em uma única interrupção e associe a esta interrupção um limite de tempo que ela pode ficar pendente. Os experimentos descritos em [21] mostram que a utilização do mecanismo de moderação de interrupções reduz a utilização da CPU e aumenta a vazão do sistema, ambos de forma significativa. Logicamente, a forma de acesso e configuração dos parâmetros relacionados a estes mecanismos depende muito da implementação do *driver* da interface.

Finalmente, Salah e El-Badawi apresentam em [35] modelos analíticos baseados na teoria de filas para representar o efeito *receive livelock*. São apresentadas fórmulas que

[8]Os mecanismos de moderação de interrupções também são referenciados na literatura como mecanismos de *coalescência de interrupções*.

modelam analiticamente dois tipos de sistemas: aqueles equipados e aqueles não equipados com mecanismos de DMA nas interfaces de rede. Os resultados numéricos, os quais foram validados através de simulações e comparados com os resultados experimentais apresentados em [26], mostram que os sistemas que não possuem mecanismos de DMA são bem mais suscetíveis a *livelocks*. A despeito da obviedade deste resultado, o trabalho mostra o ponto preciso em que sistemas sem DMA entram em *livelock*. Este ponto é exatamente quando a taxa de chegada assume um valor maior ou igual à taxa na qual o sistema é capaz de copiar pacotes da interface de rede para a memória principal. Além disso, os resultados mostram que sistemas com DMA precisam de valores para a taxa de chegada relativamente altos (tendendo ao infinito) para apresentar *livelocks*.

Contudo, algumas observações são necessárias sobre este último resultado, especialmente sobre o nível da taxa de chegada necessário para provocar *livelocks* em sistemas com DMA. Primeiro, vale lembrar que a definição formal de um sistema em *livelock*, conforme apresentada em [26], é vazão igual a zero. Porém, sistemas próximos de um estado de *livelock* também se comportam de forma extremamente ruim. Segundo, nos modelos apresentados pelos autores, a taxa de chegada dos sistemas foi representada por uma distribuição de Poisson, uma escolha que pode não refletir a natureza real do tráfego observado em sistemas que operam na WWW, cujo comportamento é extremamente variável [12].

Entre os mecanismos de controle de *livelock* apresentados nesta seção, nota-se que todos eles utilizam, de alguma forma, a técnica de *descarte antecipado* de pacotes. A finalidade desta técnica é descartar a carga excedente sem "perturbar" o sistema, rejeitando os pacotes com o mínimo de utilização dos recursos. Também é observado que a maioria das abordagens implementa uma variante da proposta inicial feita por Ramakrishnan: a técnica de *espera ocupada* (*polling*). Embora fosse esperado que este tipo de técnica aumentasse a latência do subsistema de rede, prejudicando o desempenho do sistema, os resultados dos trabalhos mostram justamente o contrário. Na média, os sistemas que utilizam espera ocupada se mostram tão eficientes quando os sistemas tradicionais orientados à interrupções. Isto é explicado pela maneira com que a espera ocupada é utilizada. Em situação de tráfego moderado, o esquema com espera ocupada converge para um esquema

orientado a interrupções. Já em situações de tráfego intenso, o bom comportamento do sistema compensa uma possível latência adicional.

2.4 Abordagens Alternativas para o Controle de Sobrecarga

A utilização de uma política de escalonamento sofisticada pode trazer grandes benefícios para o desempenho de um sistema. Crovella *et al.* demonstraram isso no contexto de servidores Web, utilizando a política SRPT (*Shortest Remaining Processing Time First*) [11]. Seguindo este mesmo raciocínio, Schroeder e Harchol-Balter utilizam uma aproximação da política SRPT como uma alternativa para o tratamento de sobrecarga [37]. Os resultados dos experimentos (realizados com versões modificadas do servidor Web Apache e do sistema operacional Linux) mostram que utilização política SRPT melhora o tempo médio de resposta por um fator de 1,5 a 8 em situações de sobrecarga transiente quando comparada à política nativa do sistema. Além disso, diferentemente da intuição geral, foi mostrado que a política SRPT não discrimina de forma significativa as requisições a arquivos grandes.

Em vez de descartar requisições de clientes como resposta à uma situação de sobrecarga, é possível utilizar uma técnica alternativa que apenas diminua a qualidade do serviço oferecido. Tal técnica é conhecida como *degradação de serviço* e é utilizada por Bhatti e Abdelzaher em [3]. Neste trabalho, os autores apresentam a proposta de uma arquitetura para adaptação de conteúdo Web cujo objetivo é melhorar o desempenho de um servidor que experimenta uma situação de sobrecarga. A idéia-chave desta técnica consiste em oferecer diversas versões de conteúdo Web que diferem entre si em termos de qualidade e requisitos de processamento (imagens com menor resolução, páginas com menos *links* e menos objetos encapsulados, versões mais simples de *scripts* CGI, etc.). Então, quando o servidor estiver sobrecarregado, as versões mais "leves" do conteúdo são enviadas para os clientes, aliviando a utilização de recursos do sistema.

Outra abordagem que pode ser utilizada no tratamento e prevenção de situações de sobrecarga é a *teoria de controle*. Abdelzaher et al [1] apresentam o protótipo de uma arquitetura que utiliza os conceitos da teoria de controle a fim de promover proteção contra

sobrecargas, garantias de desempenho e diferenciação de serviço a um servidor Web. Os autores também demonstram que, a fim de aplicar os conceitos definidos pela teoria de controle, o comportamento de um servidor Web pode ser aproximado por um modelo linear. Contudo, isto caracteriza um dos problemas da aplicação desta técnica, pois conforme exposto em [43], a própria imprevisibilidade do tráfego Internet e da demanda de recursos (que este vai exigir) torna tais sistemas difíceis de serem representados por modelos lineares.

De forma geral, as abordagens apresentadas nesta seção podem não se mostrar tão eficientes se utilizadas de forma isolada, sendo comum a sua combinação com outras abordagens, como é o caso de muitos trabalhos realizados no campo do controle de admissão. Por exemplo, um sofisticado de mecanismo de *feedback* (comum na teoria de controle) é utilizado em [20] para regular a carga imposta ao sistema. Algoritmos de escalonamento são utilizados em [6], [7] e [29]. Por fim, a técnica de degradação de serviço é considerada em [44].

2.5 Controle de Admissão *versus* Controle de *Livelock*

É certo que os mecanismos de controle de *livelock* compartilham algumas semelhanças com os mecanismos de controle de admissão. Por exemplo, ambas as abordagens visam descartar a carga excedente, evitando assim que uma situação de sobrecarga prejudique o desempenho do sistema. Porém, é importante ressaltar que existem também diferenças e problemas associados em cada abordagem. Da mesma forma, também é necessário comparar características específicas de cada abordagem. Dentro desse tema, segue abaixo uma lista de questões que tem como objetivo orientar esta discussão.

- Qual é principal diferença entre controle de *livelock* e controle de admissão?
- O controle de admissão é eficiente contra o efeito *receive livelock*?
- De que forma o controle de *livelock* pode prejudicar a aplicação de mecanismos de qualidade e diferenciação de serviço em sistemas?

A grande maioria dos mecanismos de controle de admissão apresentados neste trabalho são projetados para serem utilizados em conjunto com servidores Web, onde o controle de admissão propriamente dito é feito no nível de requisições ou no nível de sessões. Embora exista uma diferença conceitual entre estas duas abordagens, ambas consideram o elemento "requisição" quando decisões de admissão são tomadas. Mais especificamente, o termo "requisição" refere-se a um pedido feito pelo cliente ao servidor através do protocolo HTTP, que por sua vez é um protocolo da camada de aplicação.

Independente da forma de operação de um mecanismo de controle de admissão, o seu objetivo principal é evitar que um número excedente de requisições (ou pacotes) sobrecarregue o sistema. De maneira semelhante, um mecanismo de controle de *livelock* implementa um conjunto de técnicas cujo objetivo é evitar que o sistema entre em colapso em razão do tráfego excessivo de entrada. Contudo, este último sempre atua no nível do kernel, muitas vezes alterando a forma com que o subsistema de rede opera, incluindo modificações no *driver* da interface de rede e no processamento de protocolo.

De fato, tanto o controle de *livelock* como o controle de admissão são abordagens utilizadas para descartar a carga excedente. A principal diferença entre estas duas abordagens está em quão específicas se mostram as soluções e em que nível elas atuam. Enquanto o controle de admissão geralmente reside no nível de aplicação, o controle de *livelock* atua no nível do sistema operacional. Na realidade, muitos dos mecanismos de controle de admissão são apenas soluções *ad hoc* para tratar a sobrecarga. Outra diferença refere-se à maneira e à rigidez com que o descarte de carga é feito. Mecanismos de controle de admissão tendem a ser dinâmicos e adaptativos, mas descartam a carga excedente apenas depois que muitos recursos do sistema tenham sido empregados para tratá-la. Por outro lado, as soluções empregadas no controle de *livelock* em geral adotam estruturas estáticas, por exemplo, o número de pacotes processados durante a espera ocupada, porém descartam a carga excedente o mais cedo possível.

Quando a atenção recai sobre os mecanismos de controle de admissão aplicados à prevenção de situações de *livelock*, o próprio funcionamento de tais mecanismos sugere a sua fragilidade. Mesmo que alguns destes mecanismos sejam implementados no nível

de kernel e controlem a taxa com que os pacotes são despachados para as aplicações ([20] e [40] são exemplos), eles não atuam na verdadeira causa de *livelocks* em sistemas: a forma com que são tratadas as interrupções geradas pela chegada de pacotes. Isto implica que quando o sistema for submetido a um tráfego excessivo de entrada, mesmo os mecanismos referentes ao controle de admissão poderão entrar em inanição devido ao tratamento privilegiado das interrupções. Na verdade, este raciocínio pode ser expandido a qualquer outra abordagem de descarte de carga que não controle a taxa de interrupções gerada pela interface de rede ou que atue em um sistema que não provê tal controle.

Também é observado que muitos mecanismos de controle de admissão implementam funcionalidades referentes à qualidade e diferenciação de serviço. Estas funcionalidades são utilizadas pelo controle de admissão quando, na iminência de uma situação de sobrecarga, deve-se decidir qual requisição deve ser descartada. A decisão de qual requisição descartar pode ter efeitos significativos tanto para os clientes do serviço em questão como para o servidor. O mesmo se aplica a mecanismos de controle de admissão que consideram pacotes ao invés de requisições. Entretanto, para tomar tal decisão, o mecanismo de controle de admissão deve ter acesso aos dados da requisição (ou ao pacote) antes de descartá-la.

É exatamente nesse ponto que o controle de *livelock* pode prejudicar a incorporação de características de qualidade e diferenciação de serviço ao sistema, pois uma técnica comum utilizada neste tipo de controle é justamente o descarte antecipado de pacotes. Com esta técnica, muitas vezes o sistema não tem "conhecimento" dos pacotes descartados (os pacotes são descartados sem que o sistema faça *qualquer* processamento sobre eles). Realmente, um sistema que implemente uma das soluções para o efeito *receive livelock* não tem qualquer controle sobre *qual* pacote será descartado. É importante notar que o sistema continua seguindo a política tradicional de rejeição de cauda, que determina apenas *quando* os pacotes serão descartados. Contudo, outros esquemas de descarte podem ser empregados para produzir melhores resultados, como a política *Random Early Detection* (RED) [14].

Capítulo 3

Configuração dos Experimentos

Este capítulo descreve a configuração dos experimentos realizados, bem como os objetivos e motivações de cada teste.

3.1 Ambiente de Testes e Geração de Carga

O ambiente de testes utilizado neste trabalho é composto de cinco máquinas interligadas entre si através de um *switch* gigabit, cuja disposição é mostrada na Figura 3.1. Neste ambiente, três categorias de máquinas foram definidas:

Máquinas LG – geradoras de carga (três máquinas fixas);

Máquina SS_i – interpreta o papel de um servidor a ser saturado por tráfego de entrada. Esta máquina será variada ao longos dos testes;

Máquina AL – Em testes específicos, recebe e analisa o tráfego enviado pela máquina SS_i.

A configuração das máquinas LG é mostrada na Tabela 3.1. Interfaces de rede gigabit da linha Intel PRO/1000 (modelo MT), comumente chamadas de "e1000", são utilizadas por todas as máquinas LG. A configuração das máquinas que assumem o posto de SS_i (três ao todo) é mostrada na Tabela 3.2. Em especial, as máquinas da categoria SS_i possuem duas interfaces de rede: uma usada exclusivamente para a comunicação com

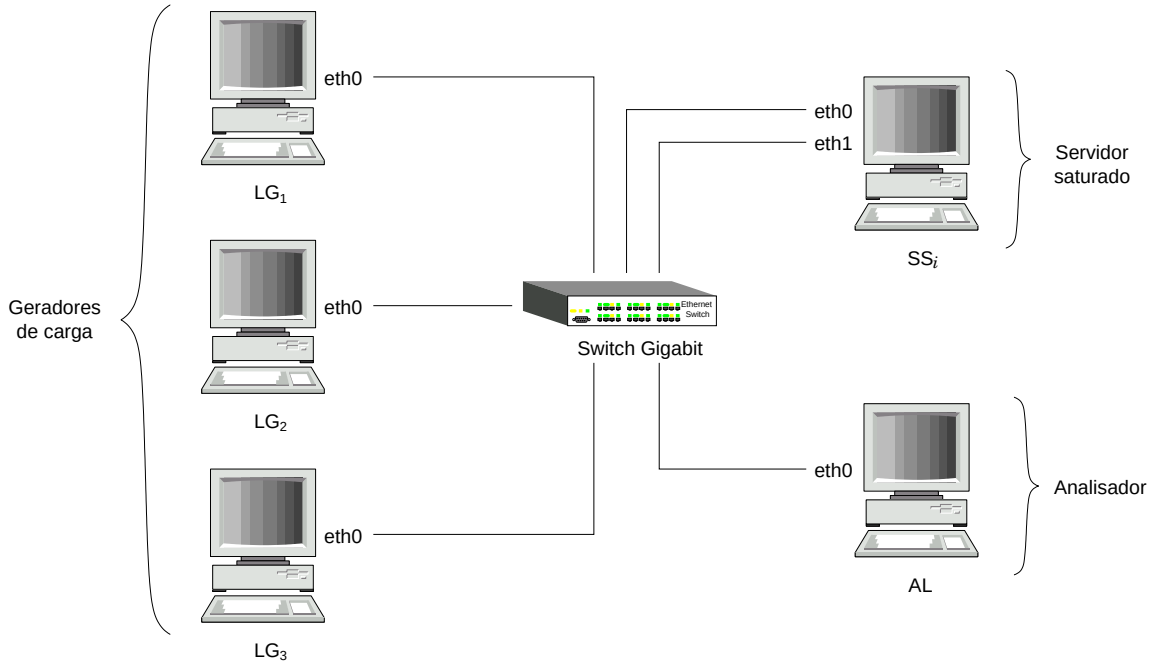


Figura 3.1: Ambiente de testes.

as máquinas LG (eth0) e outra usada para a comunicação com a máquina AL (eth1). A posição da máquina AL será preenchida com uma das máquinas SS_i que não estiver sendo usada nos testes. Todas as máquinas executam o sistema operacional Linux Debian (kernel 2.6.11.10) e são sincronizadas através do *Network Time Protocol* (NTP) [25].

	LG_1	LG_2	LG_3
CPU	Pentium IV 2,8GHz	Pentium IV 2,8GHz	Athlon 1,8GHz
Memória	512MB	512MB	512MB
eth0	e1000	e1000	e1000

Tabela 3.1: Configuração dos computadores da categoria LG.

Dois tipos de avaliações serão feitas neste ambiente de testes. A primeira, descrita na Seção 3.2, compara dois mecanismos de controle de *livelock*, moderação de interrupções e NAPI, os quais estarão atuando na máquina SS_i. A segunda, detalhada na Seção 3.3, avalia o desempenho de um servidor Web instalado na máquina SS_i, a qual novamente terá os dois mecanismos de controle de *livelock* considerados. Em ambos os casos, diversos níveis de carga, gerados pelas máquinas LG, serão submetidos à máquina SS_i com o objetivo de saturá-la. Em um caso extremo, espera-se que SS_i entre em *livelock*.

	SS_1	SS_2	SS_3
CPU	Duron 750MHz	Athlon 1,2GHz	Pentium IV 3GHz
Memória	256MB	512MB	512MB
eth0	e1000	e1000	e1000
eth1	Tulip 100Mb/s	Via 100Mb/s	Sis 100Mb/s

Tabela 3.2: Configuração dos computadores da categoria SS.

A escolha de um gerador de carga eficiente que pudesse gerar os níveis de carga necessários para os testes representou um dos desafios deste trabalho. Entre os geradores considerados estiveram o Iperf [30], o TG [39], o KUTE [45] e o `pktgen` [31]. O dois primeiros são geradores que atuam no nível do usuário e foram logo descartados por não produzirem taxas de envio de pacotes suficientemente altas. O KUTE é uma ferramenta que atua no nível do kernel na forma de um módulo, fato que deveria ajudá-la a produzir a intensidade de tráfego requerida pelos testes. Contudo, o KUTE apresentou problemas de compilação e também teve quer ser desconsiderado. Por fim, a ferramenta `pktgen`, a qual é distribuída juntamente com o código-fonte do sistema Linux, igualmente funciona como um módulo do kernel e foi capaz de gerar as taxas de envio de pacotes necessárias para os testes. Entretanto, devido a alguns problemas de configuração, o `pktgen` pode gerar apenas duas taxas por máquina: uma taxa máxima (a máquina envia pacotes o mais rápido que ela pode) e uma taxa intermediária.

A combinação das taxas produzidas pela ferramenta `pktgen` em cada máquina LG permitiu a definição de oito níveis de carga (N_c), conforme mostrado na Tabela 3.3. Cada nível de carga representa um fluxo de pacotes UDP de 64 bytes transmitidos através das interfaces eth0 das máquinas LG e recebidos pela interface eth0 de SS_i. Tal fluxo é denotado $(LG, eth0) \rightarrow (SS_i, eth0)$.

Embora seja mais fácil atingir taxas de transferência altas com o uso de pacotes grandes, no caso do fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$ pacotes pequenos são os mais indicados, pois quanto maior a taxa de envio, maior será o número de interrupções causadas pelo recebimento de pacotes em SS_i se comparado a pacotes grandes.

Nível de Carga (N_c)	Vazão	Máquina(s) Geradora(s)
N_1	127Kp/s ($\sim$ 65Mb/s)	LG_3
N_2	189Kp/s ($\sim$ 97Mb/s)	LG_1
N_3	236Kp/s ($\sim$ 121Mb/s)	LG_3
N_4	324Kp/s ($\sim$ 166Mb/s)	LG_1
N_5	371Kp/s ($\sim$ 190Mb/s)	LG_1, LG_2
N_6	498Kp/s ($\sim$ 255Mb/s)	LG_1, LG_2, LG_3
N_7	639Kp/s ($\sim$ 327Mb/s)	LG_1, LG_2
N_8	875Kp/s ($\sim$ 488Mb/s)	LG_1, LG_2, LG_3

Tabela 3.3: Níveis de carga gerados pelas máquinas LG (Kp/s = Kilo pacotes por segundo).

3.2 Comparação de Mecanismos de Controle de *Livelock*

O objetivo principal dos experimentos descritos nesta seção é avaliar o desempenho de dois mecanismos de controle de *livelock* atuando em máquinas de diferentes capacidades em uma rede gigabit. O primeiro deles é conhecido como *moderação de interrupções* e está presente na maioria das interfaces de rede gigabit comercializadas atualmente. O segundo, denominado NAPI, é uma nova API para o desenvolvimento de *drivers* de rede do sistema Linux que utiliza a técnica de espera ocupada para o tratamento de pacotes. Ambos os mecanismos são implementados pelo *driver* da interface rede e1000, sendo utilizados em todos os testes com as suas respectivas configurações padrão.

São dois os parâmetros que controlam a configuração da NAPI. O primeiro, `dev_weight`, cujo valor padrão é 64, define o número máximo de pacotes processados em um laço de espera ocupada. O segundo, `netdev_max_backlog`, ajustado com o valor 300 como padrão, limita o número máximo de pacotes processados pelo subsistema de rede antes deste ceder a CPU para outras tarefas do sistema. Ambos os parâmetros podem ser acessados através de diretório especial `/proc/sys/net/core/`.

O *driver* da e1000 oferece diversos parâmetros para o controle do tráfego de rede, incluindo temporizadores para o envio e recepção de pacotes, quantidade de descritores de pacotes (tamanho do DMA *ring*), controle de fluxo (processamento de quadros *Ethernet*

tipo *PAUSE*), limitação da taxa máxima de interrupções, entre outros. A configuração padrão do *driver* da e1000 deixa muitos destes parâmetros desabilitados. De interesse deste trabalho, estão os parâmetros correspondentes à taxa máxima de interrupções e ao número de descritores de pacotes de entrada, cujos valores padrão são, respectivamente, 8000 interrupções por segundo e 128 descritores. Experimentos indicam que bons valores da taxa máxima de interrupções para sistemas Linux estão entre 1000 e 8000 interrupções por segundo [10], o que motiva a utilização do valor padrão do *driver*. Note ainda que o valor para o número de descritores de pacotes de entrada se aplica tanto à moderação de interrupções como à NAPI.

Três tipos de testes foram definidos para avaliar os mecanismos de controle de livelock. São eles:

Teste de Vazão – avalia a capacidade da máquina SS_i de manter um fluxo constante de 10Mb/s para AL enquanto é submetida a uma carga de entrada. A taxa de 10Mb/s pode ser facilmente alcançada por um servidor Web [4]. Neste teste também é feita uma análise de pacotes que quantifica a parcela da carga de entrada que é recebida com sucesso por SS_i, bem como a parcela que é rejeita por SS_i.

Teste de utilização de CPU – mede o impacto da carga sobre a utilização de CPU de SS_i.

Teste de tempo de resposta – mostra o efeito da carga sobre o tempo de resposta de SS_i.

Nos testes de vazão, o fluxo constante de 10Mb/s entre SS_i e AL, denotado $(SS_i, eth1) \rightarrow (AL, eth0)$, é composto de pacotes UDP de 1470 bytes os quais são gerados por um processo Iperf cliente executado em SS_i. Um processo Iperf servidor que é executado em AL recebe os pacotes e reporta medições de vazão e *jitter* a cada 0.5 segundo.

O esquema geral do teste de vazão é mostrado na Figura 3.2. O fluxo $(SS_i, eth1) \rightarrow (AL, eth0)$ dura 180 segundos. As máquinas LG iniciam o fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$ no segundo 60, o qual é mantido até o segundo 120. O intervalo de tempo entre 65 e 115 segundos representa o *intervalo de confiança* do teste, onde é certo que todas a

máquinas LG estão gerando carga para SS_i. Tal intervalo é justificado pelo fato de que, mesmo as máquinas do ambiente sendo sincronizadas antes de cada teste via NTP, existe a possibilidade de haver um pequeno atraso entre início e o fim da geração de tráfego nas máquinas LG. É considerada como vazão de SS_i a média dos valores coletados em AL durante o intervalo de confiança.

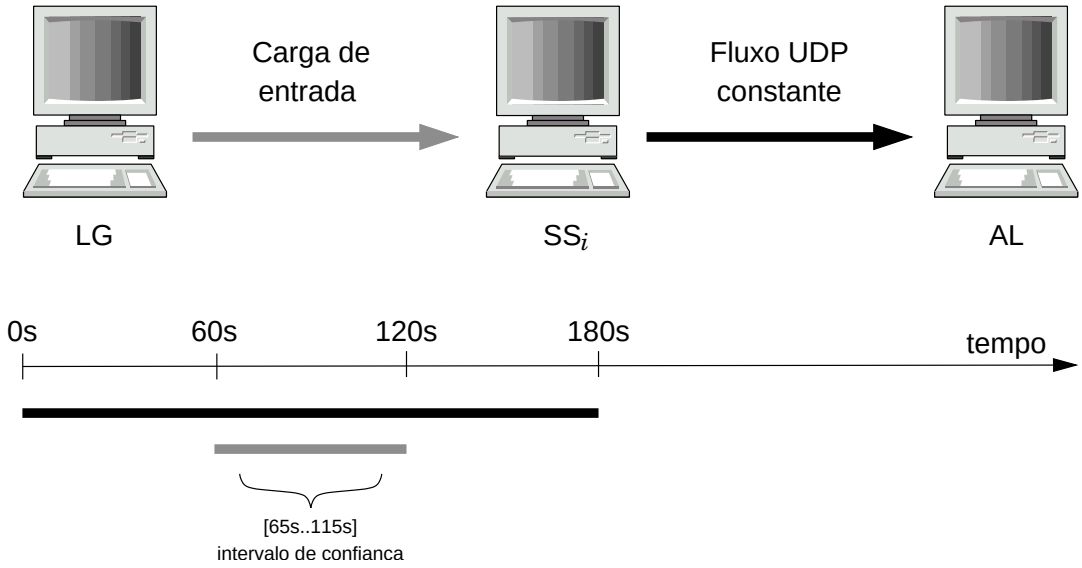


Figura 3.2: Esquema geral do teste de vazão.

A análise de pacotes nos testes de vazão é feita com o auxílio do comando `netstat`, o qual é executado antes e após a transmissão do fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$. A diferença entre os valores coletados fornece os dados necessários para a análise de pacotes. Entre as diversas informações reportadas pelo comando `netstat`, serão consideradas apenas o número de pacotes recebidos com sucesso (campo RX-OK), o número de pacotes descartados (campo RX-DRP) e o número de pacotes contados como *overrun* (campo RX-OVR).

Nos testes de utilização de CPU, apenas o fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$ figura no ambiente de testes, o qual dura 60 segundos, conforme mostrado na Figura 3.3. Para fazer as medições da utilização de CPU, o comando `vmstat` é configurado em SS_i para imprimir informações sobre o sistema a cada segundo. A média dos valores do campo "cpu/sy" (CPU utilizada pelo sistema) coletados no intervalo de confiança é definida como a utilização de CPU em SS_i.

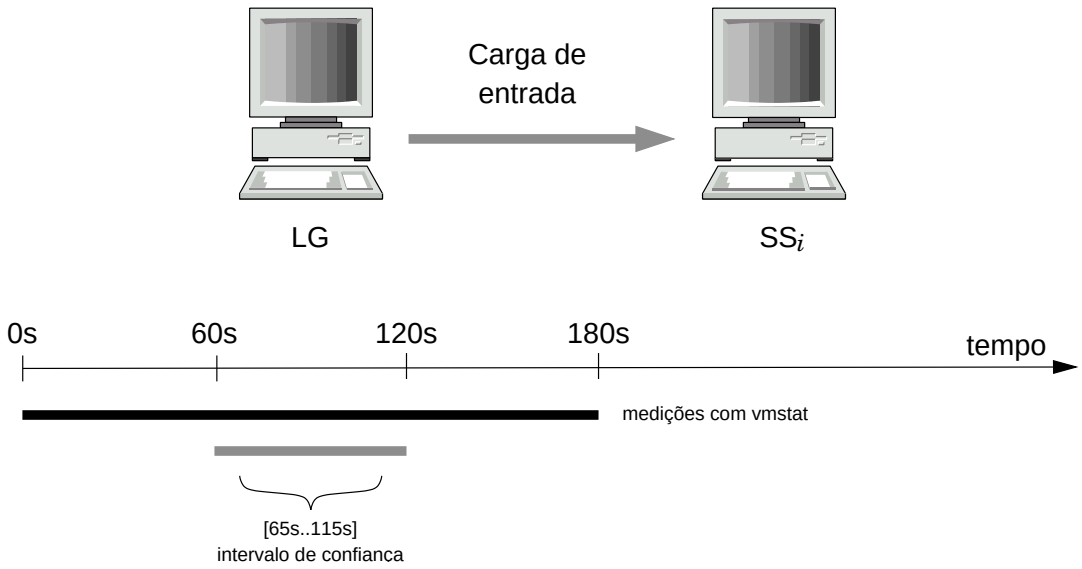


Figura 3.3: Esquema geral do teste de CPU.

O teste de tempo de resposta, cujo esquema geral é mostrado na Figura 3.4, é executado com o auxílio do comando `ping`. O comando `ping` é iniciado na máquina AL e configurado de forma a enviar pacotes durante 60 segundos para a máquina SS_i. Ao mesmo tempo, o fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$ é iniciado, o qual também dura 60 segundos (exceto para o primeiro teste, onde o tempo de resposta é medido sem a carga das máquinas LG). A média dos valores de tempo reportados pelo comando `ping`, desconsiderando os três primeiros e três últimos valores reportados, é definido como o tempo de resposta de SS_i. Os valores descartados para o cálculo do tempo de resposta objetivam, da mesma forma que o intervalo de confiança definido para os testes descritos anteriormente, evitar possíveis erros de sincronia entre as máquinas geradoras de carga e AL.

Todos os testes são repetidos para cada nível de carga gerado pelas máquinas LG. Além disso, durante os testes, diversas informações adicionais são coletadas para conferência dos resultados.

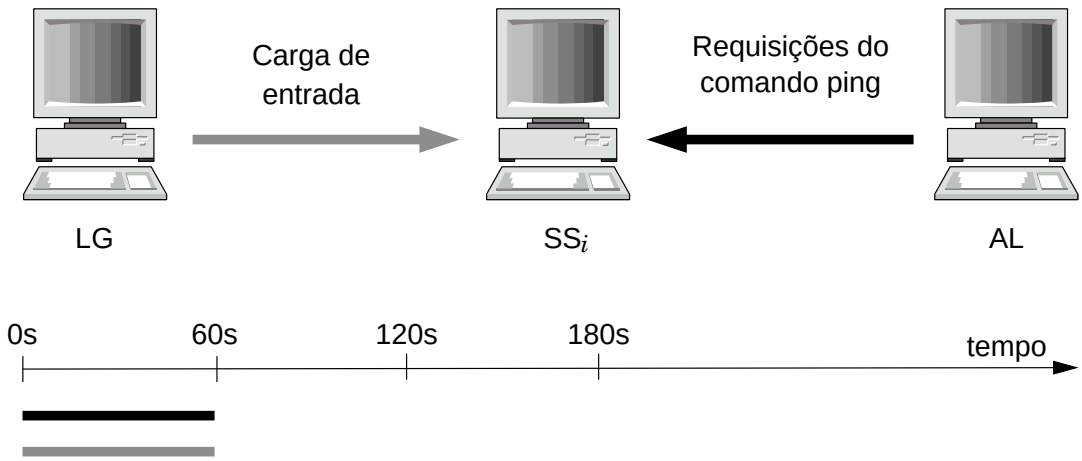


Figura 3.4: Esquema geral do teste de tempo de resposta.

3.3 Desempenho de um Servidor Web em um Sistema com Controle de *Livelock*

O experimento descrito nesta seção tem como objetivo avaliar os efeitos de vários níveis de carga submetidos a uma máquina de alta capacidade (SS_3) que executa um servidor Web e possui um mecanismo de controle de *livelock* em operação. A máquina em questão executa a versão 2.0.54 do servidor Web Apache [15] em sua configuração padrão. Os mecanismos de controle de *livelock* utilizados, como nos testes descritos na seção anterior, são a moderação de interrupções e a NAPI.

A execução de um teste compreende, além do tráfego produzido pelas máquinas LG, a geração de um fluxo constante de 150 requisições HTTP por segundo entre as máquinas AL e SS_3. Este fluxo, denotado $(AL, eth0) \rightarrow (SS_i, eth1)$, é gerado pela ferramenta `httperf` instalada em AL e simula o acesso à uma página Web estática contendo 15 arquivos, cujos tamanhos variam de 1KB a 5KB. O tamanho médio dos arquivos é 3KB. Esses valores são coerentes com os padrões de acesso observados na Internet [32].

O esquema geral do teste com o servidor Web é mostrado na Figura 3.5. O fluxo constante de requisições HTTP, $(AL, eth0) \rightarrow (SS_i, eth1)$, dura aproximadamente 180 segundos, enquanto o fluxo que visa saturar a máquina SS_3, $(LG, eth0) \rightarrow (SS_i, eth0)$, é aplicado entre os segundos 60 e 120.

As métricas *vazão* e *tempo de resposta* serão consideradas para este teste, ambas me-

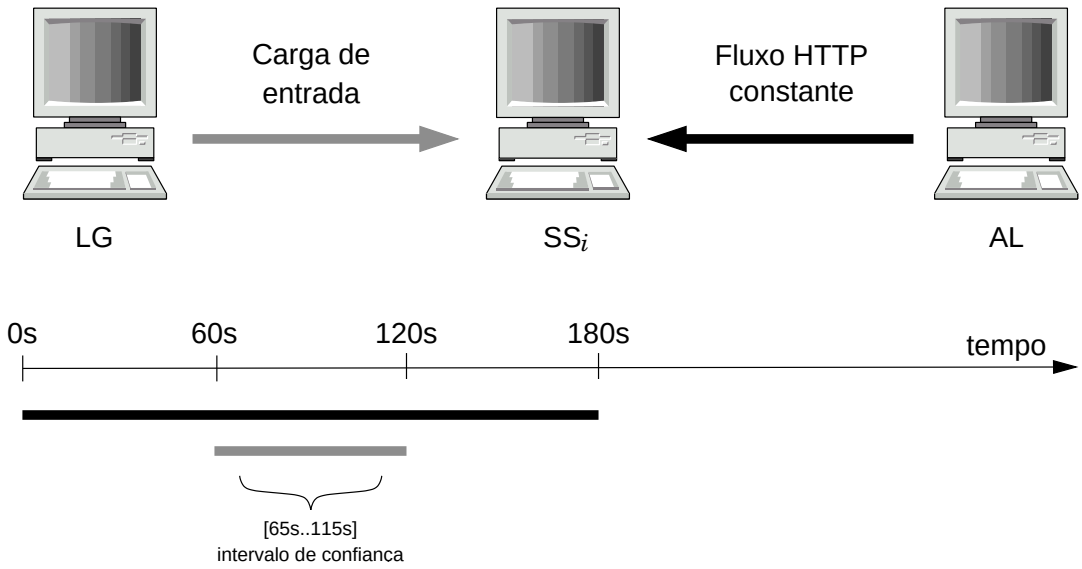


Figura 3.5: Esquema geral do teste com o servidor Web.

didas na máquina AL pela ferramenta `httperf`, a qual teve seu código-fonte modificado para imprimir, além da vazão, o tempo de resposta a cada 5 segundos. A vazão representa o número de requisições HTTP respondidas a cada segundo (respostas/s). O tempo de resposta corresponde ao tempo entre o envio do primeiro byte da requisição e o recebimento do primeiro byte da resposta do servidor. Serão definidos como valores destas métricas a média dos dados coletados em AL no intervalo de confiança.

Também cabe observar que o objetivo do fluxo $(AL, eth0) \rightarrow (SS_i, eth1)$ não é o de saturar o servidor Web, mas apenas avaliar para quais níveis de carga o servidor mantém estável as métricas consideradas. O problema de saturar servidores Web com cargas artificiais é discutido em [2].

3.4 Considerações sobre os Experimentos

Esta seção traz os principais pontos que foram considerados durante a definição do ambiente de testes e da configuração dos experimentos. São eles:

Verificação de estados de *livelock* – Em um estado de *livelock*, em que a CPU do sistema é utilizada apenas para o tratamento de pacotes de entrada, nem sempre é possível realizar medições de forma satisfatória diretamente na máquina saturada.

Logo, a determinação do estado de *livelock*, bem como a avaliação da eficiência do mecanismo testado, pode ser feita, em grande parte, de forma indireta. Este tipo de verificação é utilizado nos testes de vazão. A cada novo nível de carga submetido à SS_i, espera-se observar oscilações no fluxo $(SS_i, eth1) \rightarrow (AL, eth0)$, pois menos tempo de CPU é deixado para o processo Iperf que está gerando os pacotes para AL. Em um caso extremo, espera-se que o fluxo $(SS_i, eth1) \rightarrow (AL, eth0)$ cesse completamente, indicando que SS_i está em *livelock*, somente voltando ao normal após o término do fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$. A mesma idéia foi utilizada nos testes de tempo de resposta e nos testes com o servidor Web.

Isolamento de possíveis pontos de contenção – A definição de um sistema em *livelock* indica que o recurso mais requisitado neste sistema é a CPU. Contudo, em uma situação de tráfego extremo, é possível que algum outro recurso (enlace, interface de rede, barramento, etc.) atinja primeiro a saturação, fato que poderia prejudicar a observação de um estado real de *livelock*. Uma das medidas tomadas para evitar situações como esta no ambiente de testes foi separar os canais de recepção e transmissão de SS_i em todos testes. Enquanto a interface eth0 apenas recebe a carga gerada pelas máquinas LG, a interface eth1 fica encarregada de transmitir o fluxo constante para AL. Isto ajuda a isolar possíveis pontos de contenção, exceto a CPU da máquina SS_i.

Capítulo 4

Resultados

Neste capítulo são apresentados os resultados dos experimentos que têm como objetivo avaliar o desempenho de máquinas de diferentes capacidades submetidas a situações de tráfego intenso.

4.1 Moderação de Interrupções *versus* NAPI

Este seção apresenta os resultados dos experimentos definidos na Seção 3.2. Nestes experimentos, são comparados e avaliados os mecanismos de moderação de interrupções e NAPI atuando em máquinas de diferentes capacidades, as quais são submetidas a diversos níveis de cargas.

A Figura 4.1 apresenta o primeiro conjunto de resultados referentes à vazão e utilização de CPU da máquina SS_i para diferentes níveis de carga. Os gráficos mostram, para as três configurações da máquina SS_i, valores médios para as métricas consideradas, em função do nível de carga em milhares de pacotes por segundo (Kp/s). Observa-se que as máquinas SS_i utilizando NAPI são capazes de manter a vazão (gráficos à esquerda) do fluxo $(SS_i, eth1) \rightarrow (AL, eth0)$ constante, independente do nível de carga, enquanto as mesmas máquinas com moderação de interrupções têm a vazão bruscamente diminuída a partir de um certo nível de carga, chegando a zero (*livelock*) no nível seguinte. Os níveis de carga em que as máquinas SS_1, SS_2 e SS_3, utilizando o mecanismo de moderação de interrupções, entram em *livelock* são, respectivamente, N_4 (324Kp/s), N_6 (498Kp/s) e N_8

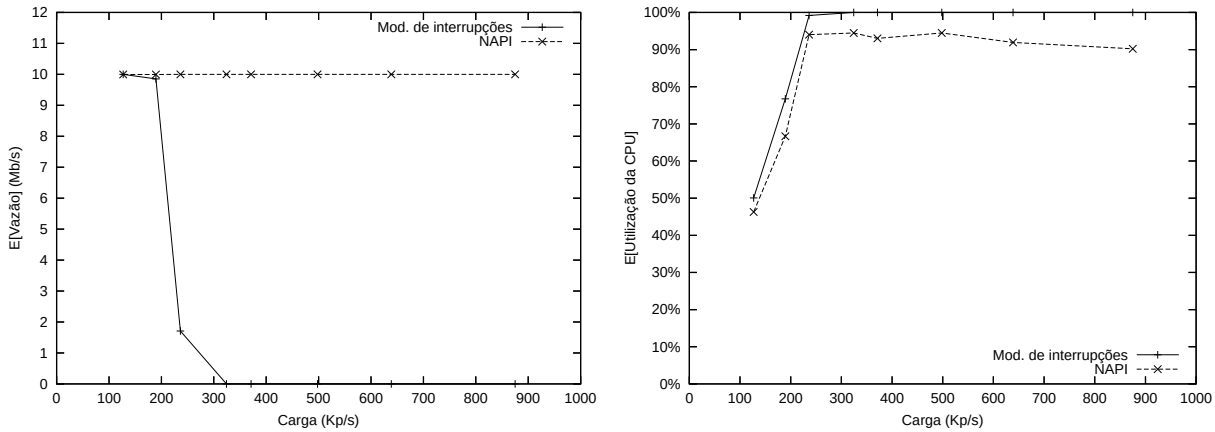
(875Kp/s). Nestes níveis, inclusive o console das máquinas não responde a comandos e nem mesmo a máquina mais potente (3GHz) é capaz de comportar a taxa de pacotes gerada pelas máquinas LG. Note ainda que, em todos os casos, as máquinas entram *livelock* para cargas que não chegam a utilizar a metade da capacidade nominal da rede.

Também é interessante observar a relação entre a capacidade das máquinas SS_i e o nível de carga no qual elas entram em *livelock* utilizando o mecanismo de moderação de interrupções. Por exemplo, enquanto a máquina de menor capacidade, SS_1 (750MHz), é completamente saturada por um carga de 324Kp/s, é necessário um pouco menos do que o triplo desta carga para saturar a máquina de maior capacidade, SS_3 (3GHz).

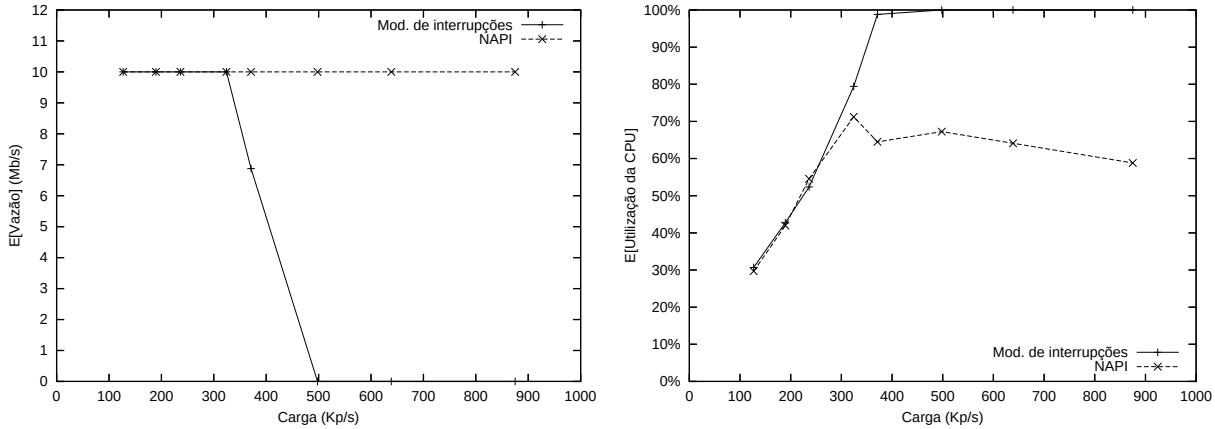
Os resultados para a utilização de CPU (Figura 4.1) ajudam a explicar o comportamento da vazão de SS_i. Seja N_l o nível de carga no qual a máquina SS_i, utilizando o mecanismo de moderação de interrupções, entra em *livelock*. A Tabela 4.1 compara os valores da vazão e utilização de CPU no níveis N_{l-1}, N_l, N_{l+1} para cada máquina SS_i. Nos casos em que a máquina SS_i está utilizando o mecanismo de moderação de interrupções, é observado que um nível de carga antes da máquina SS_i entrar em *livelock*, a vazão já está muito deteriorada, visto que a utilização da CPU se aproxima de 100%. Isto significa que, durante os testes de vazão, houve muito pouco tempo de CPU para que o programa Iperf, executado em SS_i, enviasse pacotes para a máquina AL. Porém, o mesmo não acontece quando as máquinas SS_2 e SS_3 utilizam a NAPI: mesmo para o nível N_{l+1}, a utilização média de CPU não apresenta aumentos capazes de deixar em inanição o processo Iperf; pelo contrário, em alguns casos são até mesmo observados decréscimos na utilização de CPU. Ainda assim, a porcetagem de CPU utilizada por SS_1 operando com NAPI é muito alta se comparada às outras máquinas.

Um exame específico das Figuras 4.1(a), 4.1(b) e 4.1(c) (gráficos à direita) revela que a métrica utilização de CPU no caso da NAPI apresenta o mesmo comportamento qualitativo à medida que a carga aumenta: a utilização cresce até certo ponto (que depende da capacidade de SS_i) e depois se estabiliza, tendo uma pequena diminuição.

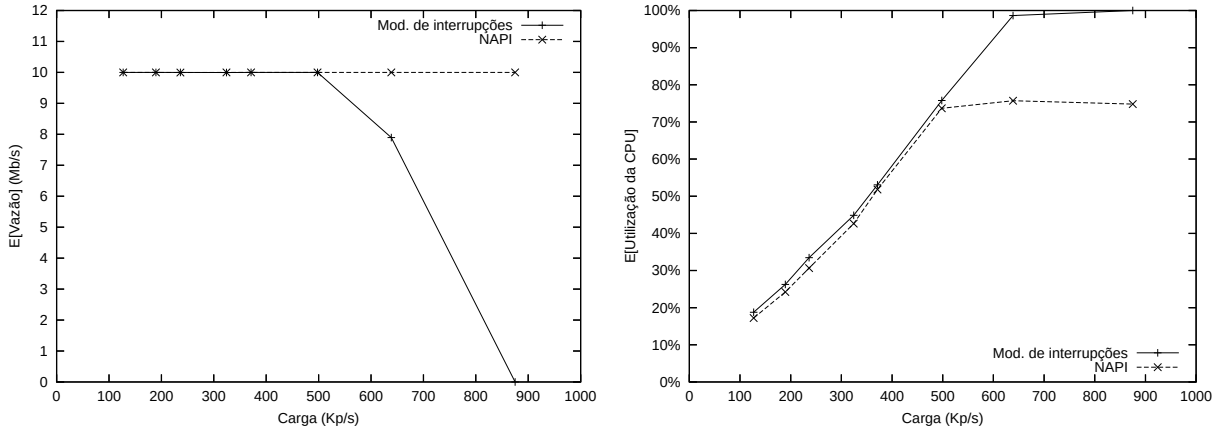
Tal diminuição pode ser explicada pelo funcionamento da NAPI. Na implementação da NAPI, o kernel processa os pacotes de rede via espera ocupada. Este processamento é



(a) SS_1 (Duron 750MHz)



(b) SS_2 (Athlon 2,8GHz)



(c) SS_3 (Pentium IV 3GHz)

Figura 4.1: Vazão média (gráficos à esquerda) e utilização média de CPU (gráficos à direita) em função da carga (milhares de pacotes por segundo) para três configurações de máquinas SS_i.

Máquinas		E[Vazão] (Mb/s)			E[Util. de CPU] (%)		
		N_{l-1}	N_l	N_{l+1}	N_{l-1}	N_l	N_{l+1}
SS_1	Mod. de interrupções	1,7	0	0	99	100	100
	NAPI	10	10	10	94	95	93
SS_2	Mod. de interrupções	6,9	0	0	99	100	100
	NAPI	10	10	10	71	64	67
SS_3	Mod. de interrupções	7,9	0	–	99	100	–
	NAPI	10	10	–	76	74	–

Tabela 4.1: Vazão média e utilização média de CPU para os níveis de carga N_{l-1}, N_l, N_{l+1} (as entradas de SS_3 marcadas com "–" referem-se a uma nível de carga não gerado, visto que $l = 8$ para SS_3).

realizado no contexto de interrupções de software (softirqs). Contudo, em casos de tráfego intenso, a fim de evitar que essas interrupções monopolizem a CPU, elas são executadas como kernel *threads* especiais de baixa prioridade (ksoftirqs), permitindo que processos de usuários também tenham chance de serem executados. Note ainda que, caso a máquina esteja atuando como um roteador (com poucos processos de usuário), os *threads*, mesmo com baixa prioridade, têm na maior parte do tempo a CPU à sua disposição, o que não implica em perdas significativas de desempenho. Logo, o modelo de operação da NAPI, em condições de carga intensa, retira gradativamente a prioridade do subsistema de rede, deixando que outras tarefas sejam executadas, diminuindo assim a utilização da CPU por parte do sistema.

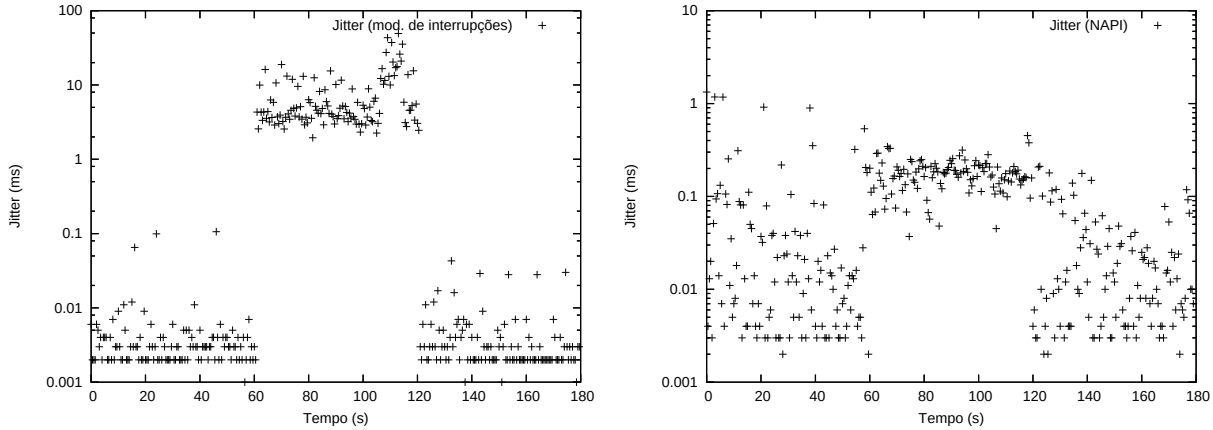
A Figura 4.2 mostra o conjunto de resultados para a métrica *jitter*. Os gráficos à esquerda (moderação de interrupções) e à direita (NAPI) apresentam o padrão do *jitter* (os valores individuais medidos na máquina AL) ao longo do tempo. O nível de carga submetido à máquina SS_i no intervalo [60s..120s] é igual a N_{l-1}. Observa-se que o *jitter* medido no intervalo de carga para a máquina SS_1 (menor capacidade) utilizando o mecanismo de moderação de interrupções (Figura 4.2(a), gráfico à esquerda) é cerca de dez vezes maior do que o *jitter* das outras máquinas no mesmo intervalo. Além disso, nos períodos sem carga, é nítida a variação do *jitter* quando compara-se a máquina SS_3

(máquina de maior capacidade) com a máquina SS_2. Também é claro que, para os testes com o mecanismo de moderação de interrupções, em todos os casos o *jitter* aumenta consideravelmente durante o intervalo de carga. Analisando os resultados do *jitter* para a NAPI (Figura 4.2, gráficos à direita), é observado que este mecanismo tem o efeito de diminuir o impacto da carga sobre o sistema, embora ainda se perceba o efeito da carga sobre o *jitter*.

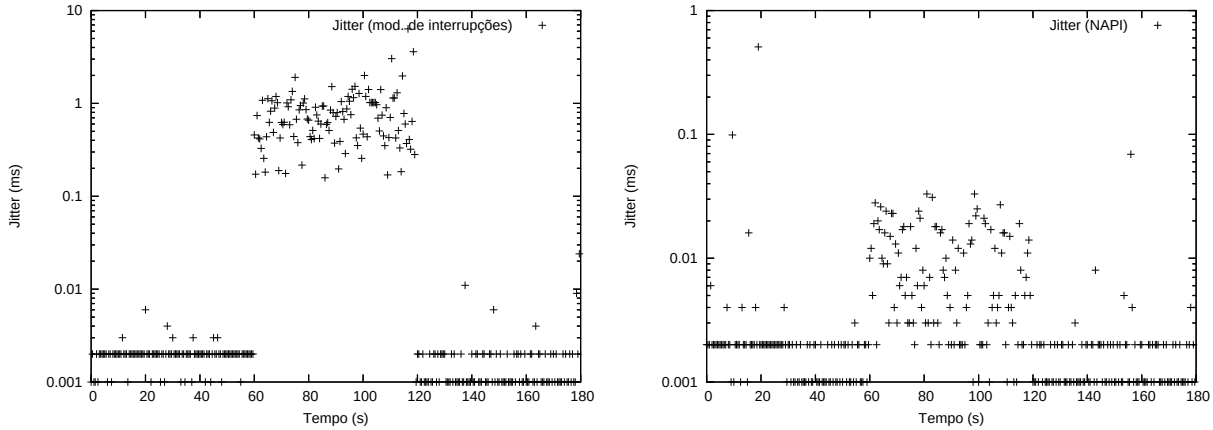
O próximo conjunto de resultados apresenta a quantidade de pacotes recebidos, descartados e *overrun* pela máquina SS_i para cada nível de carga. Neste ponto, algumas considerações sobre o número de pacotes descartados e *overrun* se fazem necessárias. O manual do comando `netstat` descreve os pacotes descartados como o número de pacotes que não puderam ser tratados pelo sistema, possivelmente devido a falta de memória, enquanto os pacotes contados como *overrun* são descritos como o número de pacotes que não puderam ser processados pelo sistema desde a última interrupção, visto que chegaram a uma taxa muito alta. Contudo, tal definição mostra-se insuficiente para uma análise detalhada dos resultados obtidos. Dessa forma, para fins de simplicidade, ambas as quantidades serão referenciadas apenas como *perdas* a partir deste ponto. De fato, uma definição precisa do número de pacotes descartados e *overrun* é muito dependente do hardware subjacente.

A Figura 4.3 mostra a quantidade de pacotes recebidos e perdas do fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$. Da mesma forma que os resultados para as outras métricas, estes também apresentam características qualitativas muito semelhantes: as máquinas recebem pacotes até certo nível de carga, onde então começam as perdas e *overruns*; após este nível, o número de pacotes recebidos tende a se estabilizar, enquanto as perdas aumentam continuamente. Em especial, o comportamento das curvas dos gráficos à direita mostra que a NAPI estabelece, a partir de um certo nível de carga, um limite superior para o número de pacotes recebidos.

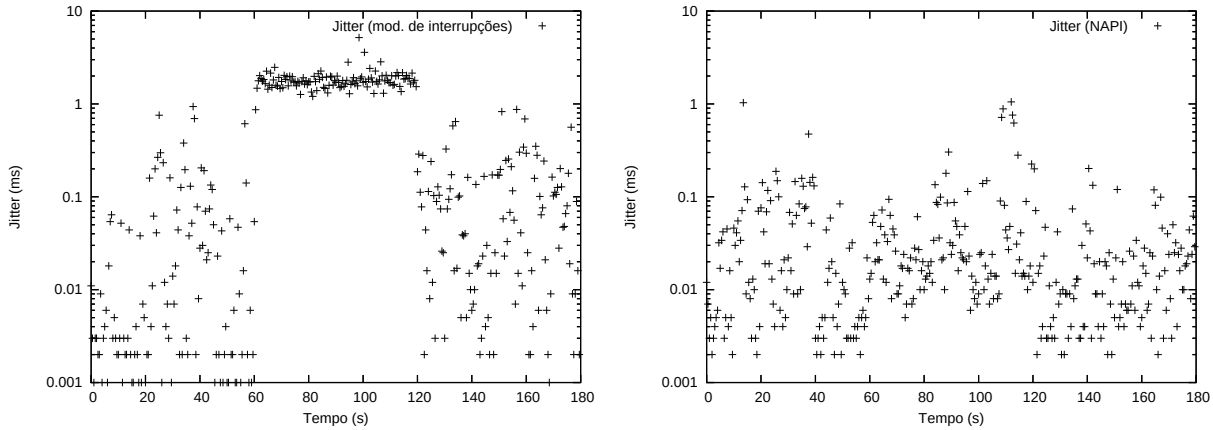
Definindo N_p como o nível de carga em que a máquina SS_i começa a apresentar perdas, os valores de N_{p-1}, N_p e $\frac{1}{N_{p-1}}$ para a máquinas SS_i operando com moderação de interrupções e NAPI são mostrados na Tabela 4.2. Nesta tabela, N_{p-1} representa o maior



(a) SS_1 (Duron 750MHz): nível de carga no intervalo [60s..120s] igual à N_3 (236Kp/s).

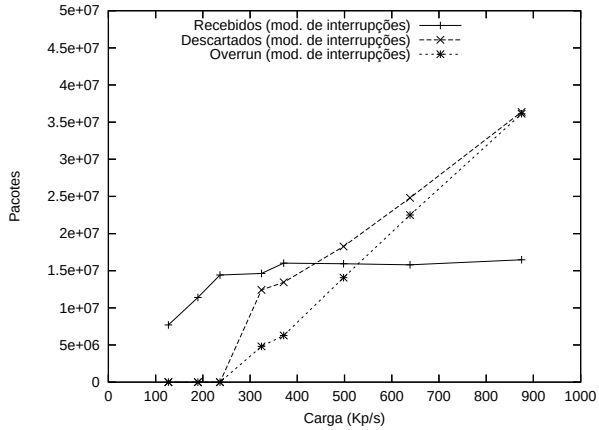


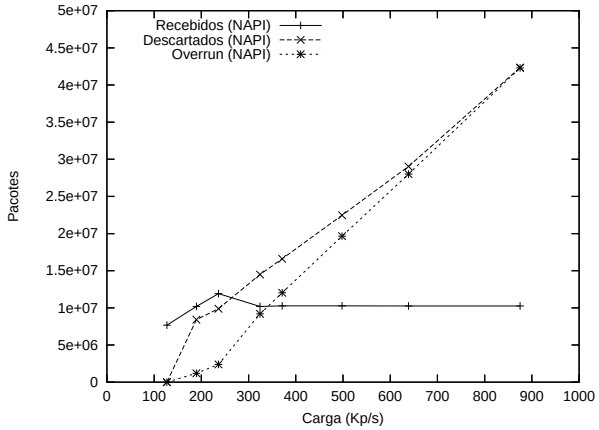
(b) SS_2 (Athlon 2,8GHz): nível de carga no intervalo [60s..120s] igual à N_5 (371Kp/s).



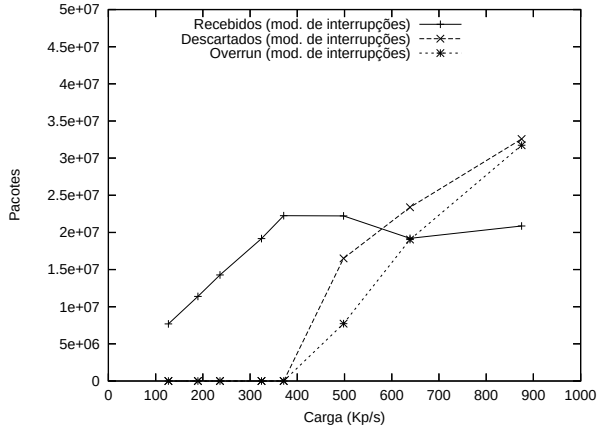
(c) SS_3 (Pentium IV 3GHz): nível de carga no intervalo [60s..120s] igual à N_7 (639Kp/s).

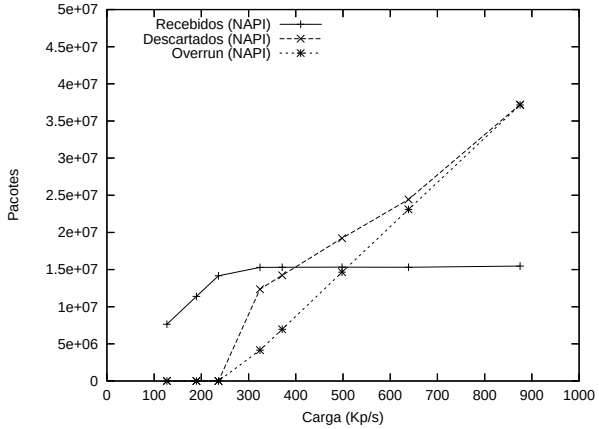
Figura 4.2: Padrão do *jitter* ao longo do tempo para os testes com moderação de interrupções (gráficos à esquerda) e NAPI (gráficos à direita)



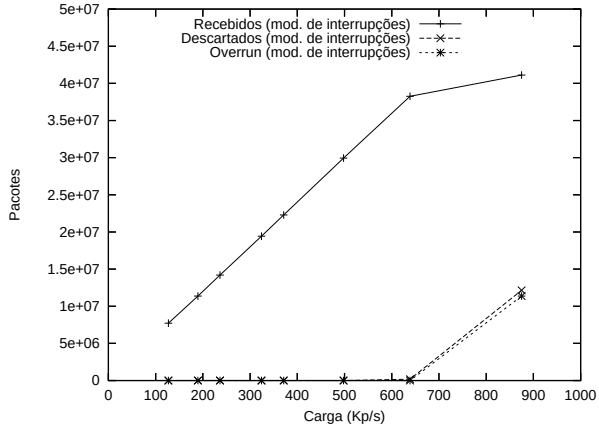


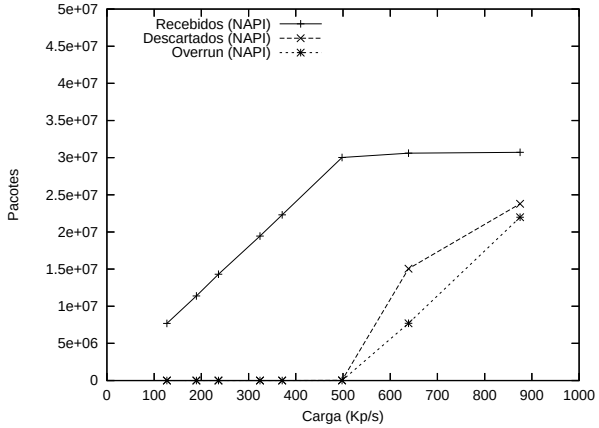
(a) SS_1 (Duron 750MHz)





(b) SS_2 (Athlon 2,8GHz)





(c) SS_3 (Pentium IV 3GHz)

Figura 4.3: Número de pacotes recebidos, descartados e *overrun* para testes com moderação de interrupções (gráficos à esquerda) e NAPI (gráficos à direita).

nível de carga gerado no qual a máquina SS_i não apresenta perdas, enquanto a fração $\frac{1}{N_{p-1}}$ fornece o tempo entre chegadas de cada pacote para o nível N_{p-1}, o qual pode ser visto como uma estimativa do tempo médio de processamento de pacotes que SS_i pode realizar.

Máquinas		N_{p-1}	N_p	$\frac{1}{N_{p-1}}$
SS_1	Mod. de interrupções	236Kp/s	324Kp/s	4,24μs
	NAPI	127Kp/s	189Kp/s	7,87μs
SS_2	Mod. de interrupções	371Kp/s	498Kp/s	2,70μs
	NAPI	236Kp/s	324Kp/s	4,24μs
SS_3	Mod. de interrupções	639Kp/s	875Kp/s	1,56μs
	NAPI	498Kp/s	639Kp/s	2,0μs

Tabela 4.2: Valores de N_{p-1} , N_p e $\frac{1}{N_{p-1}}$ para a máquina SS_i operando com moderação de interrupções e NAPI.

A análise dos valores da Tabela 4.2 mostra que para as máquinas SS_1 e SS_2 operando com moderação de interrupções, as perdas começam a acontecer somente dois níveis de carga acima do que as mesmas máquinas utilizando NAPI. Um resultado semelhante é notado para a máquina SS_3, porém em um grau de magnitude menor. Também é observado que, sob o ponto de vista de $\frac{1}{N_{p-1}}$, a NAPI insere aumentos significativos no tempo de processamento de pacotes, principalmente para as máquinas SS_1 e SS_2, nas quais os valores chegam próximos ao dobro dos valores do mecanismo de moderação de interrupções.

Os resultados descritos acima, se considerados isoladamente, podem sugerir que, em termos quantitativos, a moderação de interrupções é superior a NAPI quando se considera o número de pacotes recebidos e as perdas. Entretanto, nota-se que para a máquina SS_i, utilizando o mecanismo de moderação de interrupções, $N_p = N_l$, ou seja, as perdas só ocorrem quando a máquina entra em *livelock*. Observe também que, segundo os resultados da vazão e utilização de CPU vistos anteriormente, mesmo no nível N_{l-1} o desempenho de SS_i já está bastante comprometido. Logo, o número maior de perdas que a NAPI

provoca é justificado e, ao mesmo tempo, evidencia uma de suas principais técnicas, o descarte antecipado de pacotes, como um meio eficiente de proteger o sistema da carga excedente.

Utilizando os resultados obtidos pela análise de perdas, é possível ainda estimar o valor da Taxa de Chegada Máxima Livre de Perdas (TCMLP). Considerando N_t como sendo igual a TCMLP, tem-se que $N_{p-1} \leq N_t < N_p$. Esta formulação é válida para SS_i operando tanto com a NAPI quanto com a moderação de interrupções. Contudo, se valendo do limite superior definido pela NAPI para o número de pacotes recebidos, é possível calcular um valor aproximado de N_t para SS_i.

Então, supondo que o limite estabelecido pela NAPI não se altera consideravelmente para cargas maiores que N_8, calcula-se o valor de N_t dividindo o número de pacotes recebidos no nível de maior intensidade de carga (N_8) pelo tempo de duração do fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$, que é igual a 60 segundos. Realizando este cálculo, obtém-se os valores aproximados da N_t para as máquinas SS_1, SS_2 e SS_3 como sendo, respectivamente, 171Kp/s ($\sim$ 88Mb/s), 258Kp/s ($\sim$ 132Mb/s) e 512Kp/s ($\sim$ 262 Mb/s). Note ainda que os valores estimados são coerentes com a formulação $N_{p-1} \leq N_t < N_p$ para cada máquina SS_i.

Uma vez que se tenha o valor aproximado da TCMLP, é possível traçar graficamente o comportamento da vazão do subsistema de rede em função da carga de entrada, o que é mostrado pela Figura 4.4. Conforme previsto pelos resultados analíticos mostrados na Seção 2.1, sistemas que não possuem mecanismos eficientes para descartar a carga excedente entram em colapso para cargas além da sua capacidade, o que é provado pelo comportamento das curvas para a moderação de interrupções. Por outro lado, as máquinas SS_i com NAPI acompanham a carga até a TCMLP, ponto a partir do qual mantém a vazão constante independente da carga de entrada.

O último conjunto de resultados é exibido pela Figura 4.5 e mostra os gráficos que ilustram o comportamento do tempo médio de resposta para cada máquina SS_i. Os gráficos mostram que o tempo médio de resposta é praticamente idêntico para moderação de interrupções e NAPI durante o primeiros níveis de carga, ficando muito próximo a

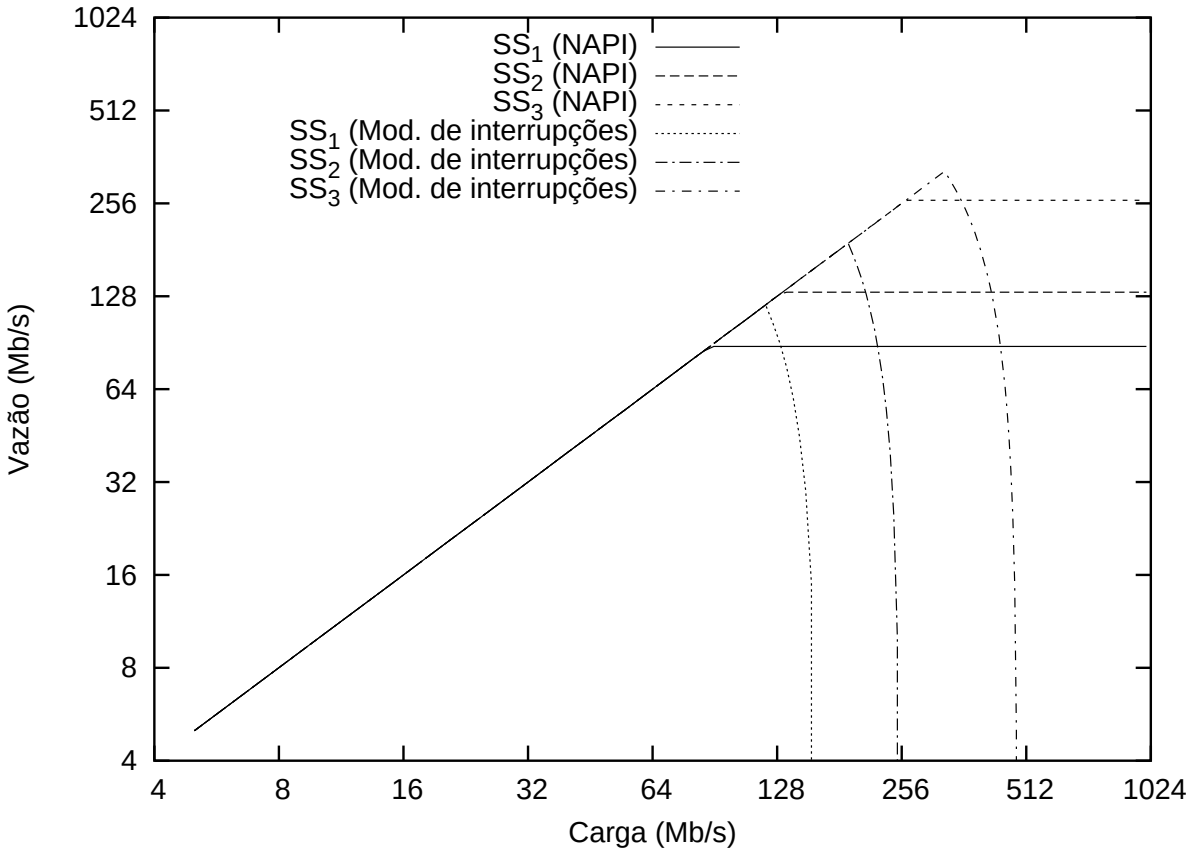
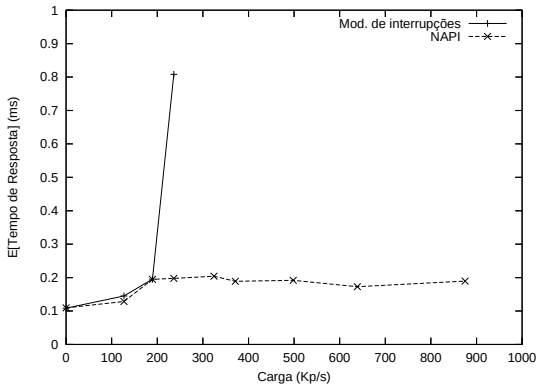


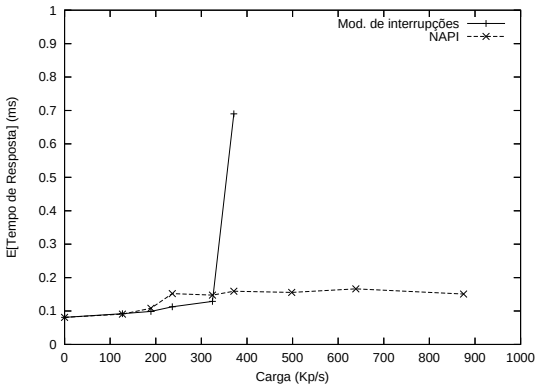
Figura 4.4: Vazão do subsistema de rede em função da carga de entrada para cada máquina SS_i operando com moderação de interrupções e NAPI.

0,1ms. Essa situação se mantém apenas até o nível de carga N_{l-1}, quando a curva do tempo médio de resposta referente à moderação de interrupções apresenta um aumento exponencial. Contraditoriamente, o pior caso desse aumento é observado para a máquina de maior capacidade (máquina SS_3, Figura 4.5(c)), na qual a diferença entre o menor e o maior valor do tempo médio de resposta chega a quase a 1,2ms, o que corresponde a um aumento de uma ordem de grandeza.

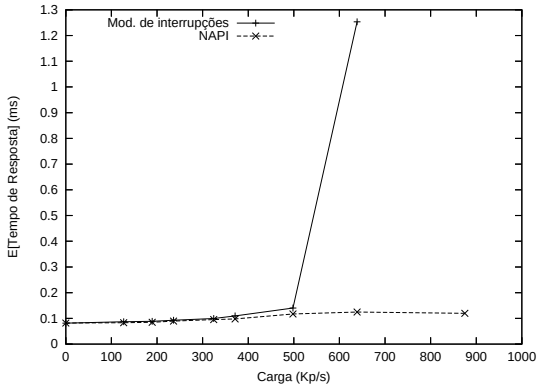
Um estudo das perdas reportadas pelo comando `ping` confirma o desempenho ruim da máquina SS_3 com relação ao tempo médio de resposta. Enquanto o comando `ping` executado na máquina AL indica uma porcentagem de perdas igual a 31% e 22% para, respectivamente, as máquinas SS_1 e SS_2 durante os testes com o nível de carga N_{l-1} (as máquinas estavam operando com o mecanismo de moderação de interrupções), as perdas para a máquina SS_3 chegam a 80% para o mesmo nível de carga. Os testes de tempo de resposta foram repetidos diversas vezes para a máquina SS_3 operando com moderação de



(a) SS_1 (Duron 750MHz)



(b) SS_2 (Athlon 2,8GHz)



(c) SS_3 (Pentium IV 3GHz)

Figura 4.5: Tempo médio de resposta.

interrupções. Todos os resultados mostraram que SS_3 continua a apresentar um desempenho pior do que as outras máquinas para o nível de carga N_{l-1}. A razão exata deste comportamento não foi bem compreendida e certamente demanda mais investigações.

A comparação dos resultados dos testes de vazão com os resultados dos testes de tempo de resposta mostra que os processos executados no nível do kernel do sistema são tão prejudicados pela sobrecarga quanto aqueles que são executados no nível do usuário. No caso dos testes de vazão, o programa Iperf (nível de usuário) executado em SS_i foi o principal prejudicado, enquanto nos testes de tempo de resposta, as rotinas específicas do protocolo ICMP (nível de kernel) presentes em SS_i, as quais são responsáveis pelas respostas às requisições ICMP feitas pelo comando `ping` executado em AL, foram as prejudicadas.

De forma geral, as análises apresentadas nesta seção indicam que a NAPI é mais eficiente do que a moderação de interrupções em vários aspectos, incluindo estabilidade do sistema em situações de tráfego extremo e utilização da CPU. A NAPI também se apresentou mais eficiente em diversas configurações de máquinas, incluindo máquinas de baixa capacidade, sendo capaz de impedir a ocorrência de *livelock* em todos os casos. O mesmo não foi verificado para a moderação de interrupções, que permitiu a ocorrência de *livelocks* em certos níveis de carga.

A análise de perdas, aliada aos resultados da vazão e utilização de CPU, mostrou a eficiência de uma das principais técnicas utilizadas pela NAPI para proteger o sistema da carga, o descarte antecipado de pacotes. Entretanto, a falta de controle de *qual* pacote será descartado pode ser um problema para sistemas que devem implementar características de qualidade de serviço.

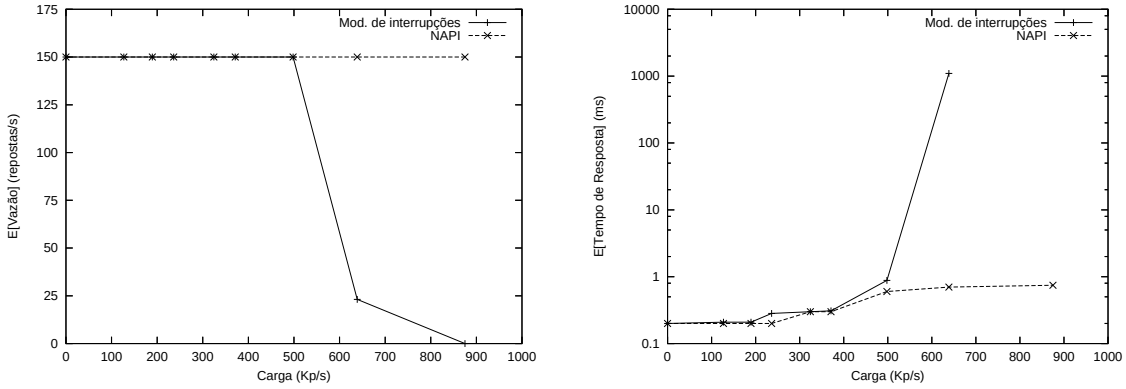
Por exemplo, considere um serviço *on-line*. Este serviço, que atende a um grupo específico de clientes, está hospedado em uma máquina de configuração semelhante a máquina SS_2, cuja conexão com a Internet é feita através de um enlace gigabit, ou seja, a máquina tem uma apenas uma interface de rede. Considere que, durante o horário de maior acesso de clientes, a máquina sofre um ataque de negação de serviço cuja intensidade é igual ao nível de carga N_8. A máquina não entrará em *livelock* se estiver usando a NAPI. Ainda assim os clientes poderão ser prejudicados, pois a grande maioria da memória reservada aos pacotes que chegam ao sistema (DMA *ring*) estará preenchida com os pacotes do ataque. Tal situação abre a proposta para mais engenharia na implementação da NAPI, talvez até mesmo incluindo algum tipo de "inteligência" no descarte de pacotes.

4.2 Desempenho do Servidor Web Apache

Esta seção mostra os resultados dos experimentos definidos na Seção 3.3, nos quais os efeitos da carga sobre uma máquina de alta capacidade (SS_3) são avaliados em termos de duas métricas de um servidor Web executado na máquina saturada: vazão e tempo de resposta.

A Figura 4.6(a) mostra que o servidor mantém a vazão constante para a maioria das

cargas aplicadas sobre máquina a SS_3. A vazão média começa a ser afetada somente a partir do nível de carga N_7 (639Kp/s) para a máquina SS_3 operando com moderação de interrupções, efeito que é expresso por uma perda de desempenho de aproximadamente 64%, enquanto o tempo de resposta (Figura 4.6(b)) aumenta praticamente em 1000 vezes. No nível N_8, a taxa de resposta da máquina SS_3 chega a zero quando o mecanismo de moderação de interrupções é utilizado, indicando o estado de *livelock*, fato que também é indicado pela ausência de um valor para o tempo de resposta para este nível.



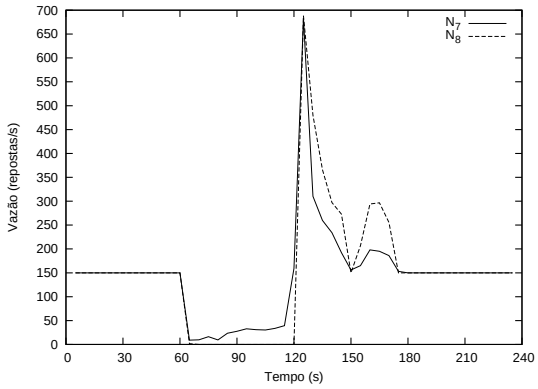
(a) Vazão média (respostas/s).

(b) Tempo médio de resposta (ms).

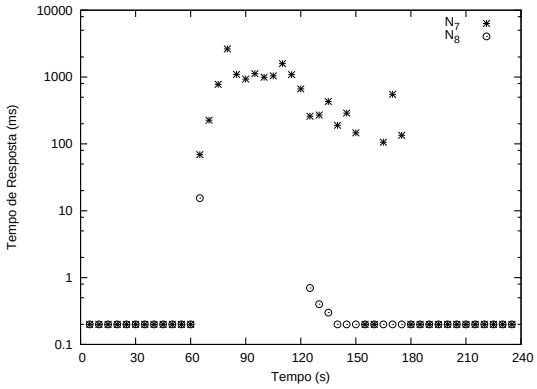
Figura 4.6: Desempenho do servidor Web Apache para a máquina SS_3 (Pentium IV 3GHz)

Os efeitos da carga sobre o desempenho do servidor quando a NAPI está sendo utilizada não são evidentes quando apenas a vazão é considerada. Logo, para esta avaliação, o tempo de resposta é o mais indicado. A Figura 4.6(b) mostra que o tráfego gerado pelas máquinas LG começa a influenciar o tempo de resposta a partir do nível de carga N_3 (236Kp/s), causando um sutil aumento de 0,2ms para 0,3ms. A partir deste ponto, acréscimos mais significativos são percebidos apenas para os três maiores níveis de carga, nos quais o tempo de resposta chega a 0,7ms.

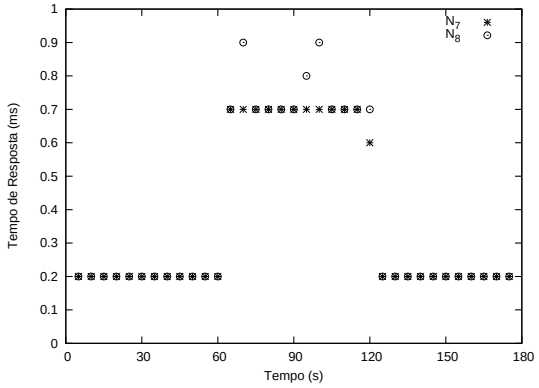
A Figura 4.7 exibe, em vez da média, os valores individuais da taxa de resposta do servidor Web Apache (reportados pela ferramenta `httperf` em AL a cada cinco segundos) ao longo do tempo, para os testes com os níveis de carga N_7 (639Kp/s) e N_8 (875Kp/s), ambos atuando no intervalo [60s..120s]. Nestes testes, a máquina SS_3 utiliza o mecanismo de moderação de interrupções nas Figuras 4.7(a) e 4.7(b) e NAPI na Figura 4.7(c). Em



(a) Vazão (moderação de interrupções).



(b) Tempo de resposta (moderação de interrupções).



(c) Tempo de resposta (NAPI).

Figura 4.7: Valores individuais das métricas do servidor Web Apache ao longo do tempo para os testes com os níveis de carga N_7 (639Kp/s) e N_8 (875Kp/s), ambos atuando no intervalo [60s..120s]. A máquina em questão é SS_3 (Pentium IV 3GHz), a qual opera com o mecanismo de moderação de interrupções (Figuras 4.7(a) e 4.7(b)) e NAPI (Figura 4.7(c)).

particular, a duração do fluxo $(AL, eth0) \rightarrow (SS_i, eth1)$ foi aumentada para 240 segundos nos testes com a moderação de interrupções a fim de evidenciar os efeitos da sobrecarga ao longo do tempo. A vazão para a NAPI não foi considerada visto que se manteve constante durante toda duração do teste.

Nota-se que o servidor Web Apache, executado em SS_3 com moderação de interrupções, não mostra qualquer dificuldade para manter a taxa de resposta constante enquanto não há a presença do tráfego provindo das máquinas LG. Contudo, a partir do instante em que o fluxo $(LG, eth0) \rightarrow (SS_i, eth0)$ é iniciado (segundo 60), a situação muda. No caso do nível de carga N_7, a vazão do servidor cai radicalmente, porém não

chega a zero, apresentando um ligeiro crescimento ao longo dos 60 segundos seguintes. O mesmo não acontece para o nível de carga N_8, o qual derruba a vazão a zero (*live-lock*), mantendo-a neste patamar durante toda a sua duração. O oposto é observado para o tempo de resposta, que aumenta consideravelmente no período de sobrecarga. Até mesmo o tempo de resposta para máquina operando com NAPI apresenta aumentos neste período.

Um comportamento interessante da vazão do servidor, operando com moderação de interrupções, é percebido quando o tráfego das máquinas LG cessa (Figura 4.7(a), segundo 120). A partir deste instante, a vazão apresenta um súbito aumento, atingindo um valor cerca de quatro vezes maior do que a vazão inicial. O primeiro aumento forma um pico de vazão cujo valor corresponde a 683 e 688 respostas por segundo, para os níveis N_7 e N_8, respectivamente. O primeiro pico de vazão é acompanhado por um segundo pico de menor intensidade (respectivamente 198 e 296 respostas por segundo para os níveis N_7 e N_8), para só então a vazão se estabilizar. O mesmo se observa para o tempo de resposta (Figura 4.7(b), que demora alguns segundos para voltar ao regime de estabilidade.

A origem para tal comportamento oscilatório da vazão reside no período de sobrecarga (segundos 60 a 120). Neste período, visto que máquina SS_3 está à beira de um colapso devido a carga gerada pelas máquinas LG, o processo do servidor Apache já não consegue tratar as várias requisições que chegam, o que implica na lotação das filas do entrada do subsistema de rede associadas ao servidor. Da mesma forma, a respostas às requisições também se acumulam nas filas de saída. Contudo, quanto o tráfego enviado pelas máquinas LG é finalizado, o conteúdo das filas de recepção/transmissão é processado à taxa máxima pelo sistema. Porém, enquanto o servidor está atendendo as tarefas acumuladas no período de sobrecarga, novas tarefas chegam ao sistema, as quais têm que esperar para serem atendidas, causando um novo acúmulo. Esse ciclo é uma possível explicação para os padrões vistos na vazão do servidor.

Outro aspecto interessante sobre o padrão da vazão observado para a máquina SS_3, operando com moderação de interrupções, após o período de sobrecarga é o tempo que o sistema permanece instável antes de voltar ao regime normal de operação, que neste

caso se mostrou equivalente ao tempo de duração da carga gerada pelas máquinas LG, ou seja, 60 segundos. Estes resultados mostram que os efeitos nocivos do tráfego excessivo de entrada persistem por um longo período de tempo após o término deste. Logo, o simples fato da carga de entrada ter cessado não garante que o sistema volte imediatamente ao seu estado normal.

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho apresentou a avaliação de dois mecanismos de controle de *livelock*, conhecidos como moderação de interrupções e NAPI, atuando com suas respectivas configurações padrão. Ambos os mecanismos se propõem a proteger a máquina em que operam de entrar em colapso em uma situação de grande tráfego.

Os dois mecanismos foram testados experimentalmente em três máquinas com capacidades entre 750MHz e 3GHz (Tabela 3.2), as quais estavam conectadas em uma rede gigabit. Estas máquinas foram submetidas a diversos níveis de carga que variaram entre 127Kp/s ($\sim$ 65Mb/s) e 875Kp/s ($\sim$ 488Mb/s).

Os resultados mostram que a NAPI é mais eficiente do que a moderação de interrupções em vários aspectos, principalmente no que se refere à estabilidade do sistema em situações de tráfego extremo. Também foi verificado que o mecanismo de moderação de interrupções não é capaz de evitar o *livelock* nas máquinas consideradas. Neste caso, mesmo a máquina de maior capacidade (Pentium IV 3GHz) é completamente saturada por uma carga que não chega a utilizar metade da capacidade nominal da rede gigabit.

Testes que avaliaram a capacidade das máquinas de manter constantes métricas como vazão, utilização de CPU e tempo de resposta em situação de tráfego intenso confirmam o bom desempenho da NAPI. Contudo, a análise de pacotes recebidos e perdas indica que a NAPI pode impor restrições a sistemas que demandem características de qualidade e diferenciação de serviço devido ao descarte indiscriminado de pacotes.

Os experimentos realizados com uma aplicação real, o servidor Web Apache, executada na máquina de maior capacidade mostram que, caso se utilize a NAPI, o atendimento a uma taxa constante de requisições não é consideravelmente afetado mesmo quando a máquina em questão é submetida à grandes cargas de entrada. Além disso, os testes com uma aplicação complexa como um servidor Web mostraram que os efeitos da sobrecarga sob um sistema que não opere com um mecanismo de controle adequado persistem mesmo após o término do sobrecarga. Os resultados indicam que o tempo que tal sistema permanece instável após o fim da sobrecarga é proporcional ao tempo de duração do tráfego de entrada que o saturou.

Uma vez que os testes descritos neste trabalho utilizaram os mecanismos de controle de *livelock* apenas em suas configurações padrão, a proposta de um possível trabalho futuro seria avaliar os efeitos de alterações nestas configurações sobre o desempenho do sistema. Por exemplo, se o valor do parâmetro que controla o número de pacotes processados no laço de espera ocupada da NAPI for aumentado, certamente um número menor de perdas será observado. Também é bem provável que um ajuste fino dos diversos parâmetros oferecidos pelo *driver* da interface de rede e1000 produzam melhores resultados para a moderação de interrupções. Um exemplo de tal ajuste é descrito em [10].

Por fim, muito foi falado neste trabalho sobre mecanismos de controle de admissão. Em especial, alguns dos mecanismos descritos atuam no nível do kernel, controlando a taxa de pacotes despachada para as aplicações [20, 40]. Uma proposta de trabalho interessante seria avaliar a eficiência destes mecanismos em situações de tráfego extremo, ou seja, avaliar se eles são eficientes contra *livelocks*. Porém, a restrição de alguns autores quanto ao fornecimento dos mecanismos desenvolvidos em seus trabalhos para pode ser um problema para tal avaliação.

Referências Bibliográficas

[1] T. F. Abdelzaher, K. G. Shin, e N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] G. Banga e P. Druschel. Measuring the Capacity of a Web Server Under Realistic Loads. *World Wide Web*, 2(1-2):69–83, 1999.

[3] N. Bhatti e T. F. Abdelzaher. Web Content Adaptation to Improve Server Overload Behavior. *Proceedings of the 8th international conference on World Wide Web*, pp. 1563–1577, 1999.

[4] R. B. Bloom. *Apache Server 2.0: The Complete Reference*. McGraw-Hill, 2002.

[5] J. Brustolini, A. Silberschatz, e A. Singh. Signaled Receiver Processing. *Proceedings of the 2000 USENIX Annual Technical Conference*, 2000.

[6] J. Carlström e R. Rom. Application-aware Admission Control and Scheduling in Web Servers. *Proceedings of the IEEE Infocom 2002 Conference*, pp. 506–515, 2002.

[7] H. Chen e P. Mohapatra. Overload Control in QoS-aware Web Servers. *Computer Networks*, 42(1):119–133, 2003.

[8] H. Chen, P. Mohapatra, e X. Chen. An Admission Control Scheme for Predictable Server Response Time for Web Accesses. *Proceedings of the 10th international conference on World Wide Web*, pp. 545–554, 2001.

[9] L. Cherkasova e P. Phaal. Session Based Admission Control: A Mechanism for Improving Performance of Commercial Web Sites. *Proceedings of the 7th International Workshop on Quality of Service*, pp. 226–235, 1999.

[10] Intel Corporation. Interrupt Moderation Using Intel Gigabit Ethernet Controllers. http://www.intel.com/design/network/applnots/ap450.htm, Setembro de 2003, Application Note (AP-450).

[11] M. Crovella, R. Frangioso, e M. Harchol-Balter. Connection Scheduling in Web Servers. *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, 1999.

[12] M. E. Crovella e L. Lipsky. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pp. 160–169, 1996.

[13] P. Druschel e G. Banga. Lazy Receiver Processing (LRP): a Network Subsystem Architecture for Server Systems. *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, pp. 261–275. ACM Press, 1996.

[14] S. Floyd e V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking (TON)*, 1993.

[15] Apache Software Foundation. The Apache Web Server. http://www.apache.org.

[16] S. D. Gribble. Robustness in Complex Systems. *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, 2001.

[17] J. D. Hansen e E. Jul. A Scheduling Scheme for Network Saturated NT Multiprocessors. *Proceedings of the USENIX Windows NT Workshop*, 1997.

[18] J. L. Hennessy e D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3rd edition, 2003.

[19] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[20] H. Jamjoom e J. Reumann. QGuard: Protecting Internet Servers from Overload. Relatório Técnico CSE-TR-427-00, University of Michigan, 2000.

[21] G. Jin e B. L. Tierney. System Capability Effects on Algorithms for Network Bandwidth Measurement. *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, 2003.

[22] B. Krishnamurthy e J. Rexford. *Redes para a Web: HTTP/1.1, Protocolos de Rede, Caching e Medição de Tráfego* . Campus, 2001.

[23] D. Kristol e L. Montulli. RFC 2965 – HTTP State Management Mechanism, Outubro de 2000.

[24] E. Lemoine, C. Pham, e L. Lefèvre. Packet Classification in the NIC for Improved SMP-based Internet Servers. *Proceedings of the IEEE International Conference on Networking (ICN 2004)*, 2004.

[25] D.L. Mills. RFC 958 – Network Time Protocol (NTP), Setembro de 1985.

[26] J. Mogul e K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-Driven Kernel. *Proceedings of the 1996 USENIX Technical Conference*, 1996.

[27] J. C. Mogul. The Case for Persistent-Connection HTTP. *Proccedings of the SIGCOMM*, 1995.

[28] K. Moore e N. Freed. RFC 2964 – Use of HTTP State Management Mechanism, Outubro de 2000.

[29] E. Nahum, S. Elnikety, J. Tracey, e W. Zwaenepoel. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. *Proceedings of the 13th international conference on World Wide Web*, pp. 276–286, 2004.

[30] National Laboratory for Applied Network Research (NLANR). Iperf. http://dast.nlanr.net/Projects/Iperf/.

[31] R. Olsson. pktgen The Linux Packet Generator. *Proceedings of the 11th International Linux System Technology Conference*, 2004.

[32] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *Measurement and Modeling of Computer Systems*, pp. 151–160, 1998.

[33] R. Prasad, M. Jain, e C. Dovrolis. Effects of Interrupt Coalescence on Network Mesurements. *Proceedings of the Passive & Active Measurement Workshop (PAM 2004)*, 2004.

[34] K. K. Ramakrishnan. Scheduling Issues for Interfacing to High Speed Networks. *Proceedings of the Global Telecommunications Conference*, 1992.

[35] K. Salah e K. El-Badawi. Evaluating System Performance in Gigabit Networks. *Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks (LCN'03)*, 2003.

[36] J. H. Salim, R. Olsson, e A. Kuznetsov. Beyond Softnet. *Proceedings of the 5th Annual Linux Showcase & Conference*, 2001.

[37] B. Schroeder e M. Harchol-Balter. Web Servers Under Overload: How Scheduling Can Help. *Proceedings of the 18th International Teletraffic Congress (ITC–18)*, 2003.

[38] A. Silberschatz e P. B. Galvin. *Operating System Concepts*. John Wiley & Sons, 5th edition, 1997.

[39] SRI International and USC/ISI Postel Center for Experimental. TG Traffic Generator. http://www.postel.org/tg/tg.htm.

[40] T. Voigt e P. Gunningberg. Kernel-based Control of Persistent Web Server Connections. *ACM SIGMETRICS Performance Evaluation Review*, 29(2):20–25, 2001.

[41] T. Voigt, R. Tewari, D. Freimuth, e A. Mehra. Kernel Mechanisms for Service Differentiation in Overloaded Web Servers. *USENIX Annual Technical Conference*, 2001.

[42] M. Welsh e D. Culler. Overload Management as a Fundamental Service Design Primitive. *Proceedings of the 10th ACM SIGOPS European Workshop*, 2002.

[43] M. Welsh e D. Culler. Adaptive Overload Control for Busy Internet Servers. *Proceedings of the 4th USENIX Conference on Internet Technologies and Systems*, 2003.

[44] M. Welsh, D. Culler, e E. Brewer. SEDA: An Architecture for Well-conditioned, Scalable Internet Services. *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[45] S. Zander, D. Kennedy, e G. Armitage. KUTE – A High Performance Kernel-based UDP Traffic Engine. Relatório Técnico 050118A, Swinburne University of Technology, Janeiro de 2005.

MARCO ANTONIO JONACK

LIMITES DE CAPACIDADE E PROTEÇÃO DE SERVIDORES EM REDES GIGABIT

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dra. Cristina Duarte Murta

CURITIBA

2005