Análise e comparação de técnicas e aceleradores de processamento de pacotes

Leonardo da C. Marcuzzo[1], Carlos R. P. dos Santos[1]

[1]Departamento de Computação Aplicada - Universidade Federal de Santa Maria (UFSM)
Avenida Roraima nº 1000 - 91.105-900 - Santa Maria - RS - Brasil

{lmarcuzzo,csantos}@inf.ufsm.br

Abstract. *The major advances in network interfaces of servers were not accompanied by the network stack of traditional operating systems. While Network Interface Cards (NICs) are able to achieve up to 100 Gbit/s of traffic, operating systems can achieve only 1/10 of this performance. Due to this, techniques for faster packet processing are being developed in order to reduce performance loss. This paper presents some of these techniques and frameworks where these are implemented, and shows a performance comparison of those frameworks against traditional operating systems, to justify their usage.*

Resumo. *Os grandes avanços no hardware de interfaces de rede de servidores comuns não foram acompanhados pela pilha de rede de sistemas operacionais tradicionais. Enquanto interfaces de rede são capazes de atingir até 100 Gbits/s de tráfego, sistemas operacionais não conseguem processar 1/10 desta quantidade de tráfego. Desta forma, algumas técnicas para acelerar o processamento de pacotes estão sendo desenvolvidas com o objetivo de mitigar esta perda de desempenho. Este artigo apresenta algumas técnicas, bem como frameworks que as implementam, e realiza uma comparação com sistemas operacionais tradicionais, justificando o uso destes aceleradores.*

1. Introdução

A demanda crescente pela capacidade de roteamento e processamento de pacotes, devido a universalização do acesso à Internet, ao uso de cada vez mais conteúdo midiático e pela disponibilidade de conexões mais rápidas requer tecnologias capazes de suportar este crescimento [ITU 2016]. Em *datacenters* e *Points-of-Presence* (PoPs), esta demanda é suprida através do uso de equipamentos proprietários especialmente desenvolvidos para o encaminhamento e processamento de pacotes (*i.e., middleboxes*).

No entanto, a utilização destes equipamentos resulta em um ecossistema inflexível, dominado por tecnologias proprietárias, com altos custos de compra e manutenção, o que impede a entrada de empresas menores no setor [Sherry et al. 2012]. Uma das alternativas ao uso de *middleboxes* é a utilização de servidores genéricos (*Commercial off-the-self* - CoTS) para executarem essas funcionalidades, aproveitando-se da disponibilidade de interfaces de rede capazes de enviar e receber uma quantidade de tráfego comparável a *middleboxes*. No entanto, a pilha de rede de sistemas operacionais tradicionais não é capaz de processar esta quantidade de dados, mesmo em arquiteturas *multi-core*, por serem desenvolvidos com foco em generalidade e compatibilidade, ao invés de alto desempenho. Como exemplo, testes realizados em [Salopek et al. 2014]

mostram que a vazão da pilha de rede do Linux é inferior a 10Gbit/s, mesmo utilizando a *New API* [Salim 2005].

Considerando esta defasagem dos sistemas operacionais, e impulsionados por novas tecnologias de rede (*e.g.*, Redes Definidas por Software - SDN e Virtualização de Funções de Rede - NFV) que podem se beneficiar destes sistemas, técnicas focadas na melhoria de desempenho no encaminhamento e processamento de pacotes em sistemas tradicionais estão sendo desenvolvidas e implementadas através de *frameworks*, com o objetivo de tornar servidores CoTS uma alternativa de menor custo com relação as soluções proprietárias existentes [García-Dorado et al. 2013]. Estes *frameworks* de aceleração de pacotes são capazes de contornar ou substituir a pilha de rede de sistemas tradicionais e utilizar essas técnicas para que aplicações no *userspace* possam ter um desempenho comparável ao de *middleboxes*, permitindo que funcionalidades exclusivas desses possam, enfim, serem implementadas em sistemas tradicionais.

Dado os benefícios que podem ser obtidos ao utilizar aceleração de pacotes, este artigo tem como objetivo apresentar uma visão geral sobre algumas das técnicas mais utilizadas, como processamento em lotes, pré-alocação de recursos e *kernel bypass*. Dois *frameworks* onde essas são implementadas, DPDK [Intel 2014] e Netmap [Rizzo 2012], terão seu desempenho avaliado com relação a vazão, latência e utilização de recursos e comparados com a pilha de rede tradicional do Linux, com o objetivo de demonstrar sua eficiência em um cenário com alto consumo de recursos de rede.

O restante do artigo está organizado da seguinte forma: A seção dois apresenta uma visão detalhada das técnicas e *frameworks* que as implementam. Após, na seção três, é apresentada a metodologia de testes e os resultados obtidos, bem como uma discussão sobre esses. Por fim, na seção quatro são apresentadas as conclusões dos autores.

2. Processamento Rápido de Pacotes

Até a versão 2.6 do *kernel* do Linux, a pilha de rede funcionava primariamente por interrupções. Desta forma, toda vez que um pacote é recebido pela interface de rede, ele é colocado em uma fila circular (*i.e., rings*), e chama uma interrupção. Após, o pacote é copiado desta fila para uma área de memória interna, conhecida como *packet kernel buffer*, que copia novamente para a pilha de rede, onde é processado, para que só assim possa ser disponibilizado para o *userspace* (através de outra cópia). A implementação da *New API* otimizou alguns destes processos para casos onde há uma grande quantidade de tráfego, mitigando interrupções (após a primeira interrupção, é feito um *pooling* para os pacotes seguintes) e, quando o sistema está sobrecarregado, permitindo que pacotes sejam descartados ainda na interface de rede, evitando cópias desnecessárias.

No entanto, ainda existem diversas limitações nessa nova implementação, como por exemplo a alocação e desalocação de recursos para cada pacote, o acesso serial, cópias múltiplas entre o *kernel* e aplicações no *userspace* e trocas de contexto. Em suma, é possível dizer que, embora a *New API* seja um avanço com relação ao estado anterior da pilha de rede, ele ainda não é adequado para ser usado em ambientes com grande quantidade de pacotes a serem processados.

Dado estas limitações, várias técnicas foram propostas com o objetivo de otimizar o desempenho dos sistemas operacionais quando trabalham com grande quantidade de dados.

2.1. Técnicas de aceleração de pacotes

A seguir, algumas das técnicas mais utilizadas serão apresentadas e detalhadas, apresentando seus benefícios e deficiências [Hermsmeyer et al. 2009]. Em [Tsiamoura et al. 2014], algumas destas técnicas foram analisadas com o objetivo de mostrar sua utilização no contexto de utilização com roteadores virtuais. Neste artigo, o foco da análise foi sobre encaminhamento e processamento de pacotes de máquinas virtuais.

- **Pré-alocação e reuso de recursos:** Tipicamente os recursos para o recebimento e processamento de pacotes são alocados na hora em que o pacote é recebido (ou enviado), e liberados após o envio para a próxima camada. A utilização de um espaço de memória alocado em um momento prévio (*e.g.*, na inicialização do *driver*) para o armazenamento de pacotes, e o reuso desta memória para armazenar os pacotes seguintes eliminam o gasto de processamento causado por esses métodos. A desvantagem é que isto resulta em um consumo maior de memória, já que mesmo que nenhum pacote esteja sendo processado ou encaminhado, a memória deve se manter alocada.
- **Suporte a múltiplos rings:** Interfaces de rede modernas possuem múltiplos *rings* de recebimento e envio, que podem ser utilizados de forma independente. Isto permite que a carga seja balanceada em sistemas *multi-core*, com aplicações, núcleos ou canais de memória diferentes escrevendo em *rings* de forma paralela. No entanto, o uso de múltiplos núcleos para recebimento e envio de pacotes reduz os recursos disponíveis para outras aplicações, além de pacotes poderem chegar fora de ordem, ocasionando problemas de ordenação em algumas aplicações.
- **Processamento em lotes:** Para reduzir a quantidade de cópias entre a interface de rede e o restante do sistema, os pacotes podem ser copiados em conjunto. Esta técnica é bastante efetiva por reduzir a quantidade de acessos ao *hardware* da interface de rede e a quantidade de cópias do *kernel* ou *framework* para a aplicação do usuário. Porém, isto causa um aumento no *jitter* e latência nos pacotes, pois estes devem esperar o processamento de todos os pacotes, ou um *timer* expirar, caso não haja pacotes suficientes no lote.
- **Kernel bypass:** Finalmente, a pilha de rede pode ser inteiramente ignorada pela interface de rede, que pode entregar diretamente os pacotes para a aplicação do usuário, criando um *fastpath* entre os descritores da placa e a aplicação. No entanto, para que esta técnica funcione, a aplicação deve possuir *drivers* próprios e ser capaz de processar os pacotes como vieram da interface (*e.g.*, desencapsular o Frame).

2.2. Frameworks de aceleração de pacotes

As técnicas citadas anteriormente podem funcionar em conjunto para que um desempenho melhor possa ser obtido. Assim, *frameworks* de aceleração de pacotes apresentam várias técnicas implementadas em diferentes camadas do sistema (*e.g., drivers, kernel, userspace*), sendo que todos estes *frameworks* possuem como foco a otimização de aplicações no *userspace*, por ser o ambiente onde a grande maioria das aplicações são implementadas, e podem ser otimizadas de forma segura. Em alguns casos, estes *frameworks* podem substituir inteiramente a pilha de rede, desde que as aplicações possam tratar os pacotes da forma que chegam da interface física.

Assim, foram identificados três *frameworks* de código aberto popularmente utilizados na literatura, e embora utilizem técnicas semelhantes, possuem arquiteturas diferentes. Nota-se que existem esforços de grandes empresas (e.g., Cisco, Solarflare) neste mesmo sentido, porém são soluções proprietárias.

- **Packet_mmap:** [Linux 2005]: Desenvolvido como uma forma mais efetiva de monitorar o tráfego de interfaces de rede no *kernel* do Linux. É implementado através de modificações nos *sockets* para que os *buffers* dos pacotes sejam alocados em uma área de memória compartilhada entre o *kernel* e o *userspace*, reduzindo o número de cópias necessárias. No entanto, os pacotes ainda precisam ser processados pela pilha de rede do sistema, o que impede que outras técnicas para a aceleração de pacotes possam ser utilizadas em conjunto.
- **netmap** [Rizzo 2012]: O *netmap* é um *framework* desenvolvido com o objetivo de reduzir o consumo de recursos necessários para encaminhar tráfego entre o *hardware* e o *userspace*. Interfaces de rede associadas com o *netmap* são controladas por um *driver* próprio, enquanto que a pilha de rede do *kernel* pode funcionar em paralelo com outras interfaces, perdendo o acesso às controladas pelo *netmap*. Das técnicas apresentadas anteriormente, o *netmap* utiliza *kernel bypass, batching* e suporte a multi-queues (se disponível no hardware), além de possuir proteções com relação a programação direta do *hardware* e do uso da memória (que precisam ser validados pelo sistema). Recentemente, foi adicionado suporte à criação de *pipes* internos (e.g, para comunicação entre Virtual Machines (VMs) no mesmo host). No entanto, sua alocação de *buffers* não é feita de forma dinâmica, o que impede que *zero-copy* seja usado de maneira efetiva. Sua implementação possui também um comutador virtual próprio (VALE) e, segundo os autores, é capaz de atingir 14.88 milhões de pacotes (i.e., a maior quantidade possível de pacotes de 64B em uma interface de 10GbE) utilizando apenas um único núcleo.
- **DPDK** [Intel 2014]: Desenvolvido pela Intel, o Data Plane Development Kit (DPDK) é um *framework* comparável ao *netmap* que utiliza técnicas semelhantes, além de possuir alocação dinâmica de *buffers* e mais funcionalidades no *userspace*. Da mesma forma que o *netmap*, o DPDK controla diretamente os descritores das interfaces, permitindo que aplicações do usuário se comuniquem diretamente com o *hardware*, diminuindo o *overhead*. Um dos diferenciais do DPDK com relação ao *netmap* é um suporte melhor para a manipulação de pacotes em arquiteturas *multi-core*, sendo capaz de distribuir de maneira otimizada o processamento nos canais de memória RAM. Como exemplo, é possível configurar o DPDK para utilizar um núcleo inteiramente para o *pooling* e encaminhamento de pacotes , deixando os outros para realizar outras funções (modo *pipeline*), ou então dividir igualmente a carga entre eles (modo *run-to-completion*).

A Figura 1 apresenta a arquitetura de alto nível destes *frameworks*, de modo a mostrar diferenças entre suas arquiteturas. É possível ver que o DPDK roda completamente no *userspace*, utilizando uma *thread* para fazer o encaminhamento dos pacotes. VMs e aplicações são conectadas a esta *thread*, que por sua vez encaminha os pacotes para o seu destino de maneira eficiente. Já o *netmap* é implementado como um módulo do *kernel* que disponibiliza uma área de memória compartilhada com o *userspace*, criando um *fastpath*.

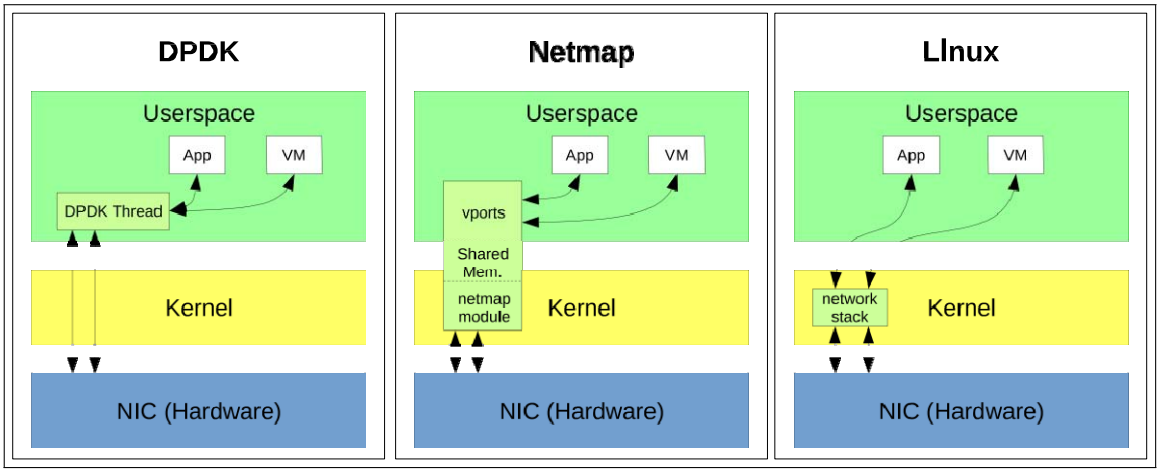

Figura 1. Comparação entre a arquitetura dos sistemas testados

Neste caso, cada VM ou aplicação têm direito a uma parte desta área onde realiza escrita e leitura. O módulo por sua vez é responsável por gerenciar e encaminhar os pacotes nesta área, fazendo cópias em lote entre áreas diferentes. Por fim, na pilha de rede tradicional do Linux, as interfaces das VMs são conectadas ao módulo *vhost*, que delega ao Linux o roteamento e encaminhamento de pacotes para ser processado pelo *kernel*. Pelo *Packet_mmap* apenas fazer alterações de baixo nível na arquitetura do Linux, sua arquitetura de alto nível é igual a do Linux.

3. Metodologia e Resultados

Dado os *frameworks* descritos anteriormente, optou-se pela criação de um cenário virtualizado, composto por duas máquinas virtuais, cada uma com uma interface de rede, um núcleo e 1Gb de RAM, executando o sistema operacional Debian 8 e virtualizadas através do *hypervisor* KVM (*Kernel-based Virtual Machine*). O *host* destas máquinas virtuais possui um processador Intel Core i5 2.2Ghz, com dois núcleos (quatro *threads*) e 8Gb de RAM DDR3@1066Mhz.

O cenário interconecta as duas VMs através do *host*, utilizando três métodos diferentes. O primeiro método é a interconexão das máquinas através de Linux Bridges. Tomou-se este como base por ser a forma padrão de conectar VMs entre si e entre a rede externa no Linux, além de utilizar a New API. Considerando que o packet_mmap é apenas uma extensão da pilha de rede do Linux, ele não foi considerado nos testes por já utilizar-se a pilha tradicional. O segundo método utiliza o *netmap* para a interconexão das VMs, o que foi feito através da inserção do módulo no *kernel* e a criação de interfaces paravirtualizadas do tipo `ptnetmap`, que por sua vez são conectadas ao comutador VALE disponibilizado pelo módulo. O terceiro método utiliza o exemplo `l2fwd` disponibilizado pelo DPDK para interconectar as VMs, definindo-se as interfaces das VMs com o tipo `dpdkvhostuser`.

Foram escolhidas as formas mais simples de interconexão nos três métodos, para que não houvesse disparidade no consumo de outros recursos do *host* e não foram realizadas nenhum tipo de otimizações nos métodos (*e.g.*, afinidade de cpu).

Assim, para a execução dos testes, foi instalado em ambas VMs a

aplicação NetPIPE [Turner and Chen 2002], para realizar testes de vazão e latência, métricas comumente utilizadas para desempenho e definidas na RFC 2544 [Bradner and McQuaid 1999]. A ferramenta foi escolhida por realizar testes com diferentes tamanhos de blocos, variando continuamente seu tamanho, de modo a simular um uso mais realista das conexões.

3.1. Resultados e Discussão

Após a realização dos testes e coleção das métricas, verificou-se que os dois *frameworks* se mostraram superiores a implementação da pilha tradicional do Linux, comprovando assim a efetividade das técnicas descritas anteriormente.

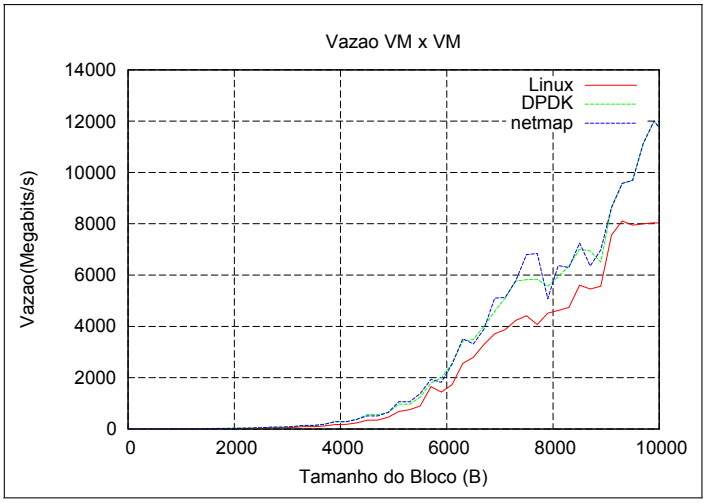

Figura 2. Teste de vazão entre duas VMs

A Figura 2 apresenta o resultado da vazão para o cenário utilizado. É possível ver que tanto o DPDK como o *netmap* possuem uma vazão muito parecida, sendo que ambas são superiores a implementação original do Linux. O desempenho de ambas foi de 12 Gbit/s para o tamanho máximo dos pacotes (10kB), enquanto que o Linux chegou a 8 Gbit/s. Mesmo que o *netmap* precise realizar cópias entre a memória, isto não afetou o seu desempenho em comparação com o DPDK.

Nota-se que o desempenho do Linux foi avaliado utilizando a tecnologia *vhost* (*i.e.,* algumas das operações de rede são movidas para o *kernel*, que já é capaz de reduzir latência, número de cópias e CPU em comparação a implementação original), mesmo assim ficando abaixo do desempenho dos frameworks.

Já a Figura 3 apresenta os resultados de latência para o cenário. Novamente, o desempenho de ambos os *frameworks* foi superior ao do Linux. No entanto, o desempenho do DPDK foi superior ao do *netmap*. Isto pode ser explicado pelo fato do DPDK executar inteiramente no *userspace* enquanto que o *netmap* precisa que seja realizada uma cópia para uma área de memória do kernel.

Tanto para vazão como para latência, o resultado dos *frameworks* foi superior ao do Linux. Verificou-se também que o consumo de CPU e memória, tanto das VMs quanto do host nunca chegou ao máximo, o que poderia impactar a validade dos resultados. Vale ressaltar que tanto o DPDK como o *netmap* necessitam que configurações adicionais sejam feitas no *host* e nas VMs, e o consumo de memória e CPU é maior do que o *vhost*.

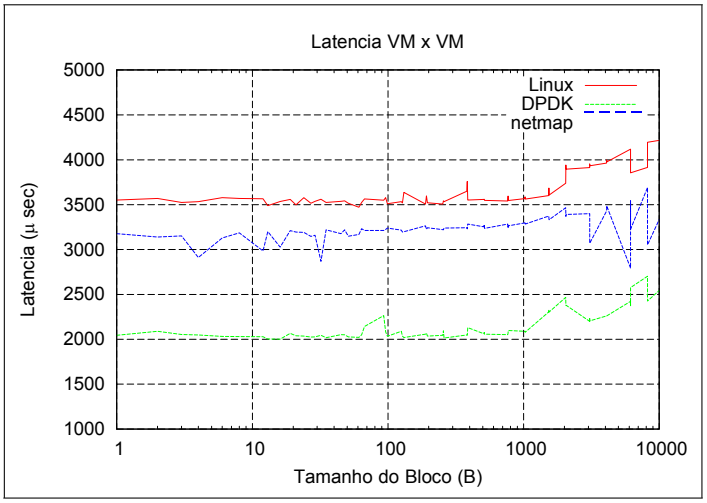

Figura 3. Teste de latência entre duas VMs

Os resultados de latência foram na ordem dos microssegundos, devido ao fato de todas máquinas e aplicações executarem no mesmo *host*. Mesmo assim, nota-se claramente um desempenho superior dos *frameworks*, principalmente do DPDK, que reduziu a latência quase pela metade quando comparado ao Linux. Acredita-se que isto se deve ao fato de este ser executado inteiramente no *userspace*, o que reduz trocas de contexto e outras operações que custam ciclos ao processador. O seu uso elevado de recursos, causado pela necessidade de *HugePages* e um núcleo inteiramente dedicado ao *pooling* devem ser considerados na hora da implementação. No caso dos testes realizados com o DPDK, um dos núcleos utilizou sempre 100% de processamento enquanto que os núcleos utilizados pelas máquinas virtuais se mantiveram com utilização semelhante aos dos outros testes. Com o Netmap, verificou-se um uso maior de CPU nas máquinas virtuais, possivelmente causado pela utilização dos drivers especiais do Netmap.

4. Conclusão

Esta pesquisa teve como objetivo a apresentação, análise e comparação de aceleradores de processamento de pacotes. Foi possível identificar claramente vantagens e desvantagens no uso dos *frameworks* para aplicações com alto consumo de recursos de rede. Como vantagem, pode-se considerar o desempenho superior para as métricas avaliadas, enquanto que as desvantagens são relacionadas a dificuldade de configuração e um consumo maior de recursos do sistema como um todo.

Desta forma, acredita-se que, no caso de funções virtualizadas de rede e serviços de alto desempenho, *frameworks* se mostram como uma opção superior quando comparado à pilha do Linux.

Já para o uso com aplicações que não necessitam de alto desempenho de rede, deve-se avaliar as vantagens e desvantagens em usar estes *frameworks*, que podem consumir recursos que não são necessários nestes cenários, diminuindo a quantidade restante para ser usada por outras aplicações. Estas tecnologias podem contribuir para que sistemas tradicionais funcionem em conjunto com *middleboxes*, em ambientes de *datacenter*.

Por fim, como trabalhos futuros, o teste com outros cenários (*e.g.*, máquinas físicas) pode identificar o ganho de desempenho em ambientes diferentes. No entanto

isto requer uma infraestrutura que esteja preparada para receber e implantar aceleradores de processamento de pacotes.

Referências

Bradner, S. and McQuaid, J. (1999). Benchmarking methodology for network interconnect devices. RFC 2544, RFC Editor.

García-Dorado, J. L., Mata, F., Ramos, J., Santiago del Río, P. M., Moreno, V., and Aracil, J. (2013). *High-Performance Network Traffic Processing Systems Using Commodity Hardware*, pages 3–27. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hermsmeyer, C., Song, H., Schlenk, R., Gemelli, R., and Bunse, S. (2009). Towards 100g packet processing: Challenges and technologies. *Bell Labs Technical Journal*, 14(2):57–79.

Intel (2014). Data plane development kit. *URL http://dpdk.org*. Acesso em 27 abr. 2017.

ITU (2016). Ict facts and figures 2016. *URL http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf*. Acesso em 03 set. 2016.

Linux (2005). Packet mmap. *URL kernel.org/doc/Documentation/networking/packet mmap.txt*.

Rizzo, L. (2012). netmap: A novel framework for fast packet i/o. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 101–112, Boston, MA. USENIX Association.

Salim, J. H. (2005). When napi comes to town. In *2005 Linux Conf*.

Salopek, D., Vasić, V., Zec, M., Mikuc, M., Vašarević, M., and Končar, V. (2014). A network testbed for commercial telecommunications product testing. In *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 372–377.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., and Sekar, V. (2012). Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24, New York, NY, USA. ACM.

Tsiamoura, K., Wohlfart, F., and Raumer, D. G. (2014). A survey of trends in fast packet processing. In *Proceedings of the Seminars Future Internet (FI), and Innovative Internet Technologies and Mobile Communication Networks (IITM)*, pages 41–48.

Turner, D. and Chen, X. (2002). Protocol-dependent message-passing performance on linux clusters. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pages 187–194. IEEE.