Os Dois Lados de Containers: Explorando o Balanço Entre Isolamento e Desempenho de Funções de Rede Virtualizadas

Nicolas Silveira Kagami[1], **Luciano Paschoal Gaspary**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

nskagami@inf.ufrgs.br

Abstract. *Network Function Virtualization (NFV) has been suggested as a solution to the telecommunications industry's dependence on deploying dedicated physical devices to provide new functionality. Before NFV is widely adopted by the industry, it is essential that its robustness and performance aren't far from what is observed in traditional middlebox solutions. The use of containers has been recently studied as a lightweight alternative to the vastly employed hardware virtualization. However, most studies don't examine the effect of exerting isolation in the performance of containers. This factor can be decisive and should not be overlooked, specially considering that implementing isolation is a more intricate endeavour in containers than in hardware virtualization. This paper aims to characterize the trade-off relationship between isolation and performance in container based NFV environments, exploring technologies regarding isolation, communication, network functions and anomalies.*

Resumo. *Network Function Virtualization (NFV) tem sido sugerida como solução à histórica dependência da indústria de telecomunicações em implantar dispositivos físicos dedicados para prover novas funcionalidades. Para que NFV seja amplamente adotada pela indústria, é fundamental que a robustez e o desempenho não estejam muito longe daqueles observados no contexto de middleboxes tradicionais. A utilização de containers tem sido estudada recentemente como uma alternativa mais leve e com melhor desempenho do que a vastamente empregada virtualização de hardware. No entanto, a maioria das investigações não explora o custo de exercer isolamento no desempenho de containers. Tal consideração pode ser decisiva e não pode ser negligenciada, uma vez que a implementação do isolamento em containers é mais intrincada que em virtualização de hardware. Neste artigo propõe-se caracterizar a relação de compromisso entre isolamento e desempenho em ambientes NFV baseados em containers. Mais especificamente, explora-se eixos de tecnologias de isolamento, comunicação, funções de rede e anomalias.*

1. Introdução

A indústria de telecomunicações tem se apoiado historicamente na implantação de dispositivos físicos dedicados para prover novas funcionalidades, como *middleboxes*. Esse tipo de equipamento frequentemente requer um encadeamento específico com outros serviços, incumbindo não só em restrições topológicas da rede como na localização física dessas funções componentes. Tais restrições, juntamente com a exigência dos usuários por qualidade, estabilidade e custo baixo, acarretaram ciclos de produto mais longos e uma forte

dependência em hardware especializado [Mijumbi et al. 2016], culminando num enrijecimento topológico e funcional.

Network Function Virtualization (NFV) tem sido sugerida como solução a essas complicações. NFV é um conceito emergente no qual funções de rede podem ser implantadas como instâncias de software, abstraindo os componentes físicos subjacentes. Essa abordagem permite maior flexibilidade e escalabilidade dos serviços, uma vez que os mesmos podem ser provisionados dinamicamente em resposta a mudanças momentâneas na rede, e relocados a qualquer hardware compatível com sua tecnologia de virtualização.

Múltiplas questões relevantes devem ser adequadamente respondidas antes que NFV seja amplamente adotada pela indústria. Um aspecto fundamental é que a robustez e o desempenho observado em soluções NFV não esteja muito longe daqueles observados no contexto de *middleboxes* tradicionais. Essas questões estão fortemente atreladas às tecnologias de virtualização e comunicação em cima das quais a solução NFV é desenvolvida.

Existem duas vertentes principais na tecnologia de virtualização que têm atraído atenção da academia recentemente, a virtualização baseada em *hypervisors* e a virtualização baseada em *containers*. A virtualização por *hypervisors*, também conhecida como virtualização por hardware, se caracteriza por abstrair o hardware subjacente, garantindo isolamento de recursos de cada máquina virtual ao custo de um *overhead* de acesso. A virtualização por *containers*, também conhecida como virtualização por software, pode ser vista como uma alternativa mais leve, uma vez que apresenta um sistema operacional compartilhado que serve de base para todos *containers*, acarretando um menor custo de desempenho.

O emprego de *containers* em NFV pode trazer múltiplos benefícios em relação ao uso de máquinas virtuais, como um menor *overhead* em acesso ao hardware, maior escalabilidade, além de mais rapidez e flexibilidade no estabelecimento de funções novas. Essas vantagens vêm ao custo de isolamento potencialmente mais precário, feito através de construtos do próprio sistema operacional, enquanto que as instâncias de virtualização por *hypervisor* ocorrem em sistemas operacionais distintos. Soluções de comunicação para *containers* ainda estão sendo desenvolvidas e avaliadas e serão determinantes para a aceitação de *containers* no âmbito de NFV.

Alguns estudos têm sido feitos na direção de entender o que tecnologias de virtualização leve podem oferecer em um contexto de NFV, sugerindo alguns tópicos de pesquisa que requerem atenção, como tecnologias de comunicação e soluções de isolamento. Neste trabalho propomos uma avaliação das características de isolamento e desempenho de *containers*, tentando responder as seguintes questões de pesquisa.

- Qual o desempenho sustentado que pode ser obtido por *containers* em um contexto NFV, sem perder de vista questões de segurança e isolamento?
- Como as tecnologias de comunicação disponíveis afetam o balanço de desempenho e isolamento?

Este trabalho apresenta uma investigação do comportamento de *Virtual Network Functions* (VNFs) implementadas em *containers*, com o objetivo de averiguar como escolhas de tecnologias de comunicação e isolamento influenciam no desempenho quando uma função vizinha apresenta comportamento errático. Mais especificamente, avalia-se as capacidades de rede, que são essenciais no contexto de NFV.

O restante deste artigo é organizado da seguinte forma. A Seção 2 apresenta trabalhos relacionados e o estado da arte no estudo de *containers*. A Seção 3 caracteriza a aplicação de uma função virtualizada de rede no contexto de virtualização por *containers*. A Seção 4 descreve o ambiente de avaliação experimental, detalhando e justificando as escolhas de ferramentas, métricas, cenários e cargas de trabalho. A Seção 5 relata os resultados dos experimentos executados. A Seção 6 desenvolve pontos de análise sobre os resultados obtidos e os relaciona com questões de isolamento e desempenho em *containers* assim como suas implicações para NFV. Por fim, a Seção 7 versa sobre a contribuição do artigo e elabora sobre trabalhos futuros.

2. Trabalhos Relacionados

Em um amplo estudo dos atuais desafios de pesquisa em NFV [Mijumbi et al. 2016], Mijumbi *et al* afirmam que *containers* podem ser mais apropriados do que *hypervisors* dependendo da função. Este trabalho aponta ainda que *containers* podem trazer até 30% de economia em custos de servidor se comparados a virtualização por hardware. Em uma investigação similar em escopo [Han et al. 2015], Han afirma que as soluções atuais de NFV em máquinas virtuais (*hypervisor*) têm problemas de memória e tempo de instanciação. Além disso, sustenta que a indústria requer implementações mais leves, remanescendo como um tópico de pesquisa em aberto. Ambas investigações destacaram o potencial de *containers*, e ressaltaram a necessidade de estudo na área.

Em um estudo da aplicação de NFV ao fornecimento dinâmico e elástico de acesso público à internet, Heideker e Kamienski observaram vantagens de desempenho e tempo de instanciação na aplicação de *containers* [Heideker and Kamienski 2016]. *Containers* também foram utilizados em um ambiente dedicado à submissão de cargas de trabalho, a ferramenta Rufus, cujo artigo ressalta que *containers* apresentam uma alternativa efetiva e de baixo custo [Souza et al. 2016]. O impacto de virtualização por *containers* também foi considerado positivo para a aplicação em *Data Centers* [Netto et al. 2016] e computação massivamente paralela e distribuída [Barbosa et al. 2014].

Em uma avaliação das tecnologias disponíveis de *hypervisors* e *containers* [Morabito et al. 2015], Morabito concluiu que *containers* apresentam melhor desempenho, com um *overhead* quase imperceptível para a maioria das métricas. No entanto, o autor condiciona essas vantagens ao custo de segurança e isolamento, salientando a carência de estudos sobre esses aspectos. Em um outro estudo [Felter et al. 2015], Felter comparou implementações das tecnologias de virtualização e concorda que *containers* geralmente apresentam menor *overhead*. O autor sugere que *containers* devem oferecer melhor desempenho e boa adequação a IaaS (*Infrastructure as a Service*). Porém, enfatiza que algumas questões de comunicação ainda devem ser investigadas. Há consenso entre Morabito e Felter sobre o melhor desempenho de *containers* em relação a *hypervisors*. Ambos, no entanto, apontam que esforços devem ser feitos para prover o isolamento, que seria garantido na virtualização por *hardware*.

Em uma investigação sobre comunicação de *containers* aplicada a NFV, Anderson confirma as vantagens de *containers* em relação a *Virtual Machines* (VMs) e afirma que, especificamente, latência e *jitter* requerem mais estudo, uma vez que atrasos regulares são importantes na cadeia de serviço de *Virtual Network Functions* (VNF) [Anderson et al. 2016]. Um estudo do desempenho de módulos de *kernel* como tecnologias de comunicação de *containers* salienta que *containers* e comunicação *inter-containers* requerem muito estudo [Claassen et al. 2016]. O estudo chegou a conclusão de que *macvlan* é o módulo de *kernel* com melhor desempenho, porém, a versão analisada não incluiu opções de isolamento disponíveis. Os estudos focados em comunicação de *containers* reforçam suas capacidades e explicitam a necessidade de estudo na área.

Considerando os trabalhos previamente mencionados, é possível concluir que *containers* têm o potencial para diminuir custos em NFV devido a seu baixo impacto. Em geral, *containers* apresentam melhor desempenho do que soluções equivalentes por *hypervisors*, porém, a sua capacidade de isolamento não é garantida. Esse é um aspecto que requer estudo, uma vez que a virtualização leve não providencia isolamento inerente, diferentemente da virtualização de *hardware*, em que o isolamento é consequente do substrato de virtualização. Nesse sentido, uma questão em aberto é quanto o esforço de *containers* para manter o isolamento afeta seu desempenho final. Também foi proposto que as tecnologias de comunicação de *containers* sejam investigadas, não somente por seu papel no desempenho e no isolamento, mas principalmente por ser um aspecto essencial para garantir a qualidade de funções de rede.

3. Função Virtualizada de Rede em Containers

Esta seção caracteriza os diferentes mecanismos disponíveis para virtualização de *containers* e seu potencial proveito no processo de implantação de funções virtualizadas de rede. Mais especificamente, busca-se analisar como diferentes combinações dessas tecnologias podem contribuir na direção de um equilíbrio adequado entre desempenho e isolamento, duas características fundamentais para NFV.

Soluções de virtualização por *containers* permitem a interação com o *kernel* hospedeiro, o que acarreta menores perdas de desempenho, uma vez que efetivamente elimina a necessidade de emulação de dispositivos físicos e *drivers*. Por esse mesmo motivo, *containers* não garantem intrinsecamente o isolamento apresentado por soluções de virtualização a nível de hardware, nas quais a camada de virtualização mantém um sistema operacional dedicado entre as possíveis interações inadequadas e o *kernel* hospedeiro.

As capacidades de isolamento de *containers* provêm de um leque de mecanismos de software, muitos dos quais são soluções maduras originárias da comunidade de sistemas operacionais. A Figura 1 ilustra como *containers* estruturam seus recursos para prover o ambiente de virtualização. Particularmente no nível de *kernel* estão ferramentas como *cgroups*, para delimitar uso de recursos, *AppArmor*, para estabelecer controle de acesso e *namespaces*, para confinar o escopo de nomes, as quais edificam o ambiente de isolamento. A configurabilidade desses mecanismos possibilita que um administrador de sistema equilibre requisitos de desempenho e isolamento, que podem ser essenciais para calibrar o comportamento de uma função virtualizada de rede.

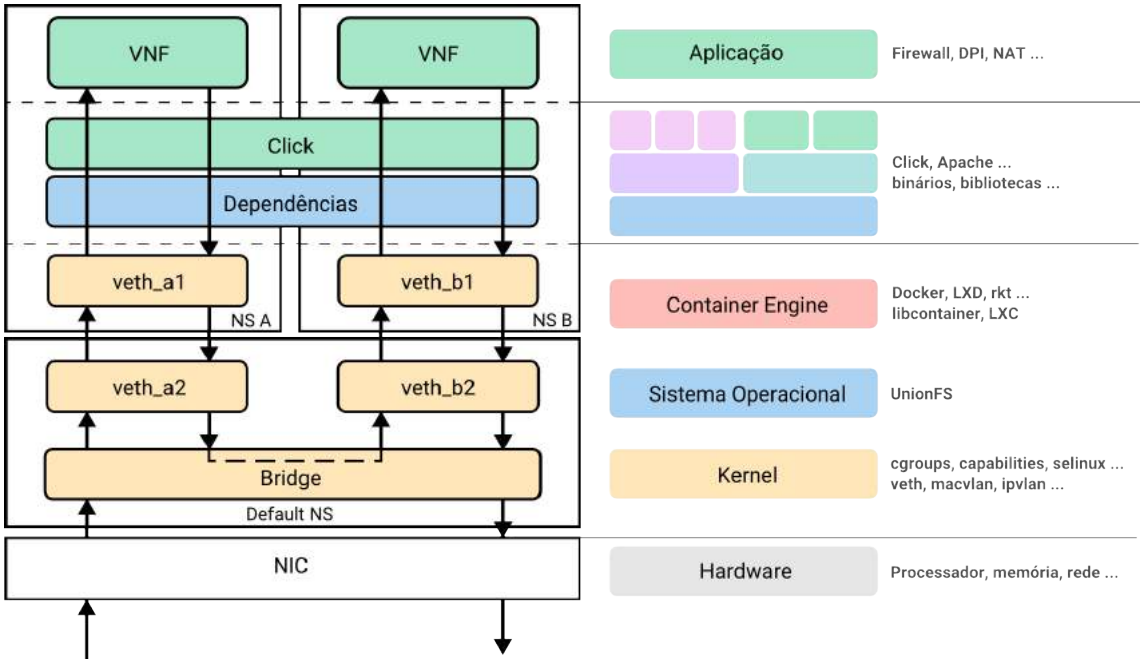

Figura 1. Representação estrutural de *containers* de aplicação.

Algumas soluções de *containers* (como Docker e rkt) têm se especializado em comportar aplicações, aproveitando dependências compartilhadas para economizar recursos. Essas soluções têm sido denominadas *containers* de aplicação, em contraste aos *containers* de sistema, que representam o conceito original de *container*. Outro esforço para contornar ineficiências foi o desenvolvimento da biblioteca *libcontainer*, que disponibiliza o uso dos recursos de isolamento do *kernel* linux sem a necessidade de componentes do espaço de usuário como LXC, libvirt e systemd-nspawn.

Containers de aplicação, desbravados pelo Docker, apresentam *union filesystems*, que permitem que arquivos e diretórios de *filesystems* diferentes se sobreponham transparentemente. Além disso, o sistema *Copy-on-write* (CoW) possibilita que múltiplos *containers* acessem um mesmo dado, mantendo a capacidade de possuir suas próprias versões, em caso de modificação. Essas duas funcionalidades são integrais ao sistema de imagens e camadas, apresentado na Figura 1, o que diminui consideravelmente o consumo de recursos quando utilizando múltiplos *containers* que compartilham as mesmas camadas. Somando isso ao fato de que é possível implementar funções virtualizadas de rede distintas utilizando um arcabouço estático de dependências (como *click modular router*), pode-se alcançar um nível de densidade de funções de rede muito alta.

Dentre as diversas tecnologias de comunicação disponíveis a *containers*, os módulos de *kernel* representam a configuração padrão distribuída pelas soluções atuais. O módulo de *kernel veth* estabelece um par de interfaces de rede, uma no *namespace* do *container* e outra no *namespace* padrão, deixando a comunicação do *container* restrita ao que flui entre esses pontos. O Docker, por padrão, cria pares *veth* para seus *containers* e os conecta por uma *bridge* a sua rede principal. Esses níveis de indireção afetam o desempenho fortemente, mas facilitam isolamento, uma vez que toda comunicação com o *container* pode ser facilmente configurada.

O módulo de *kernel macvlan* tem sido usado como alternativa em casos de necessidade de desempenho. O *driver* da controladora de rede fica sob comando (*slave*) das interfaces *macvlan* presentes nos *namespaces* dos *containers*, cada uma associada a um endereço MAC. Seu ganho em desempenho em relação a *veth* é compensado com menos isolamento inerente, considerando que, a princípio, todas interfaces compartilham o mesmo domínio de *broadcast*. No entanto, diferentes modos de operação de *macvlan* permitem ajustar isolamento e desempenho, sendo possível criar um canal de comunicação especial entre *containers* ou vetar sua intercomunicação inteiramente. Esse atalho de comunicação *inter-container* pode ser apropriado para diminuir latência no encadeamento de VNFs, mas também permite que um *container* comprometido afete seus vizinhos.

É importante notar que a virtualização por *containers* representa uma alternativa que não só oferece maior desempenho no caso ideal, mas principalmente oferece a capacidade de configurar funcionalidades de isolamento específicas, determinando o grau de impacto no desempenho. A eficiência de recursos também pode ser beneficiada, possibilitando maior densidade de funções de rede do que a obtida em soluções de virtualização por *hypervisor*. As investigações atuais têm se concentrado no desempenho ideal de *containers*, muitas vezes desconsiderando o efeito de exercer restrições de isolamento. Para que a comparação com soluções de virtualização de hardware seja pertinente e para que a comunidade de desenvolvimento de *containers* possa adequá-los ao ecossistema de NFV, é necessário avaliar o efeito no desempenho que essas ferramentas podem acarretar quando exercendo suas funções.

4. Ambiente de Avaliação

O objetivo desta seção é descrever o ambiente de avaliação experimental, detalhando e justificando as escolhas de ferramentas, métricas, cenários e cargas de trabalho. O conjunto de experimentos visa a caracterização do desempenho de uma função virtualizada de rede estabelecida em uma *Service Function Chain* (SFC) durante a ocorrência de uma anomalia em um *container* vizinho, como ilustrado na Figura 2. Determina-se o desempenho por meio de um fluxo que origina e culmina nas funções de rede fonte e destino, considerando especificamente a comunicação *inter-container* característica de uma SFC que compartilha o substrato físico de virtualização. A anomalia representa o comportamento errático de uma instância de função de rede, forçando o sistema a exercer seus mecanismos de isolamento, o que pode comprometer o desempenho da função de rede.

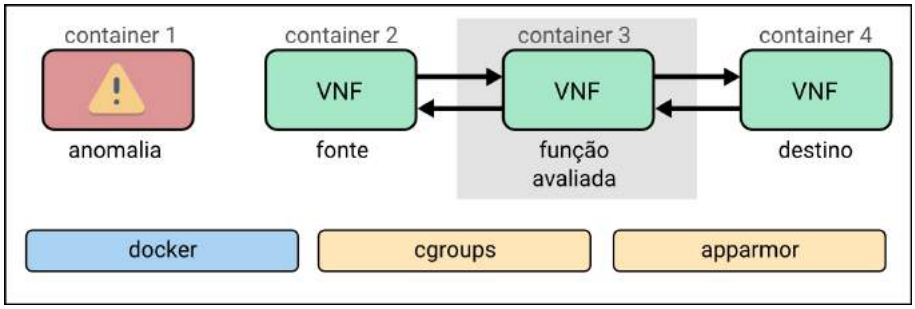

Figura 2. Fluxo de teste utilizado, representando a comunicação *inter-containers* em uma *Service Function Chain*.

Das tecnologias de virtualização atualmente disponíveis, *Docker*[1] foi escolhido por sua ampla utilização e reconhecimento. Além disso, representa a tendência de reformulação de *containers* para acomodar aplicações de forma mais eficiente. As tecnologias de isolamento incluem ferramentas do *kernel linux* utilizadas tanto por *libcontainer*[2] quanto *liblxc*, caracterizando tanto *containers* de aplicação como de sistema. A ferramenta *cgroups*[3] é utilizada para restringir o uso de recursos de *containers*, enquanto que o módulo de *kernel AppArmor*[4] é empregado para delimitar controle de acesso, ambas as quais trabalham por padrão com o *Docker*. O perfil de execução do *cgroups* foi composto de forma a reservar os recursos necessários para os *containers* envolvidos no fluxo de avaliação, limitando recursos como CPU, memória, número de operações de I/O por segundo e capacidade de vazão de disco. Os recursos necessários para o funcionamento do encadeamento de funções de rede foram determinados a partir do consumo observado em uma execução prévia do ambiente de avaliação, o qual ocorreu na ausência de anomalias. O perfil escolhido para o AppArmor foi o *docker-default*, que é o padrão para *containers Docker*.

Para implementar as funções virtualizadas de rede, foi escolhido o *Click Modular Router*[5], uma arquitetura de *software* para programar roteadores configuráveis. *Click* disponibiliza um conjunto de elementos processadores de pacotes a partir do qual múltiplas funções de rede podem ser compostas. Além disso, a estrutura de camadas e sistemas de arquivos de *containers* de aplicação favorece implementações que compartilham dependências, tornando configurações como esta particularmente econômicas de instanciar. As funções de rede escolhidas foram *Firewall*, *Deep Packet Inspection* (DPI) e *Network Address Translation* (NAT), de forma que os aspectos mais comuns do manuseio de pacotes estejam representados nos casos avaliados. O *Firewall* é implementado por um elemento do *Click* que simplesmente filtra pacotes de acordo com padrões pré-definidos, representando uma função leve de filtragem. O DPI é composto por um elemento que inspeciona e redireciona pacotes filtrados para caminhos diferentes, representando a capacidade de manipulação de fluxos do *Click*. A função de NAT seleciona pacotes e reescreve parte de seus cabeçalhos, representando a modificação de pacotes.

As tecnologias de comunicação avaliadas abrangem os módulos de *kernel veth* e *macvlan*, em diferentes configurações. Esses módulos foram escolhidos por sua popularidade e documentação, além de abranger múltiplos níveis de desempenho e isolamento. A configuração denominada *veth bridge*, visível na Figura 3(a), é composta de um par *veth* para cada *container*, conectados a uma *bridge* no *namespace* padrão. Esse é o padrão de comunicação do *Docker* e permite tanto comunicação externa como interna.

A Figura 3(b) ilustra *veth direct*, cujas interfaces virtuais *veth* encontram-se nos *namespaces* de dois *containers*, efetivamente estabelecendo uma ligação direta. Tal configuração não possibilita comunicação externa, mas sua comunicação interna é menos onerosa que do que a *veth bridge*, uma vez que múltiplos níveis de indireção são substituídos por uma conexão única.

[1] `https://www.docker.com` - Acessado em Março 2017
[2] `https://github.com/opencontainers/runc` - Acessado em Março 2017
[3] `https://wiki.archlinux.org/index.php/cgroups` - Acessado em Março 2017
[4] `https://wiki.archlinux.org/index.php/AppArmor` - Acessado em Março 2017
[5] `http://read.cs.ucla.edu/click` - Acessado em Março 2017

A configuração *macvlan bridge* é representada na Figura 3(c), onde interfaces *macvlan* localizadas nos *containers* controlam uma interface *slave* no *namespace* padrão. A comunicação *inter-container* é possibilitada por uma *pseudobridge* que liga diretamente os *containers* via memória, oferecendo alto desempenho ao custo de isolamento. Outros modos de operação de *macvlan* não permitem comunicação direta *inter-container*, como o modo *private*, ou implementam tal comunicação de forma mais indireta, como o modo VEPA, que requer um *switch* servindo como refletor.

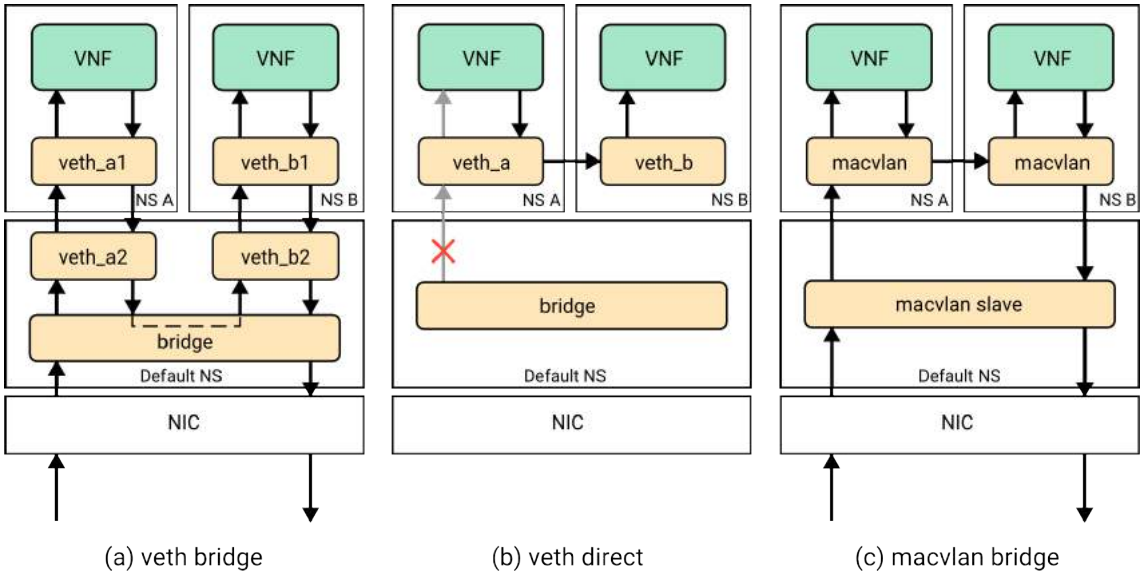

Figura 3. Configurações das interfaces de comunicação utilizadas.

As anomalias são compostas de comportamentos irregulares específicos, separados em estratégias que afetam recursos diferentes, como CPU, memória, I/O e disco. Para simular esses comportamentos foi empregada a ferramenta *stress*[6], na qual a anomalia de CPU é simulada invocando múltiplas *threads* que executam *sqrt()*. A anomalia de memória consiste em *threads* que alocam memória via *malloc()*, tocando posições em um intervalo de 4096 *bytes* e então desalocando via *free()*. Por fim, as anomalias de I/O e disco são provocadas pela utilização de *sync()* e *write() / unlink()*, respectivamente.

As métricas de avaliação consideradas são focadas em características cruciais para a qualidade de funções de rede, como capacidade de vazão, latência (ou *Round Trip Time*) e *jitter*. A ferramenta *iperf3*[7] é utilizada para gerar as cargas de trabalho, medindo fluxo TCP para determinar a capacidade de vazão e fluxo UDP para determinar *jitter*. A ferramenta *ping* é parametrizada para medir latência a cada 100 ms, resultando nas medidas de latência média e latência máxima. A latência máxima, ou pico de latência, é determinada como o maior valor de RTT encontrado em uma instância de avaliação e ajuda a estimar a capacidade de impacto instantâneo das anomalias avaliadas. A Tabela 1 ilustra as métricas de avaliação juntamente com suas respectivas ferramentas de medição, duração e unidades de medida.

[6] `http://people.seas.harvard.edu/~apw/stress/` - Acessado em Março 2017
[7] `http://software.es.net/iperf` - Acessado em Março 2017

Métrica	Ferramenta	Unidade	Duração
Vazão	*iperf3 TCP*	Mbps	15 s
Latência Média	*ping*	ms	15 s
Latência Máxima			
Jitter	*iperf3 UDP*	ms	15 s

Tabela 1. Métricas de avaliação e ferramentas de medição.

O sistema de avaliação empregado é um servidor Dell PowerEdge R420, com um processador Intel Xeon E5-2400, 32 GB de RAM 1333 MHz, executando Ubuntu 16.04 *kernel* 4.4.0. O experimento foi executado estilo *Full Factorial*, permutando todos os parâmetros descritos na Tabela 2 e repetindo o experimento 12 vezes, totalizando 2160 instâncias de teste. Cada instância tem uma duração de 45 segundos, divididos entre os testes *iperf3 TCP*, *iperf3 UDP*, e ping, separados por 5 segundos. Além disso, as instâncias aguardam 5 segundos após a aplicação das anomalias para avaliar a função de rede.

Parâmetros	Valores
Tecnologia de Comunicação	*veth bridge*, *veth bridge*, *macvlan bridge*
Tecnologia de Isolamento	Nenhuma, *cgroups*, *AppArmor*, *cgroups & AppArmor*
Função de Rede Virtualizada	*Firewall*, DPI, NAT
Anomalia	Nenhuma, CPU, Memória, Disco, I/O

Tabela 2. Parâmetros utilizados em uma avaliação *Full Factorial*.

5. Resultados

Esta seção relata os resultados obtidos a partir da execução do ambiente de avaliação previamente descrito. Todas as observações abaixo foram estatisticamente consideráveis a um nível de significância $1 - \alpha = 95\%$. A seguir estão quantificadas as observações mais interessantes com o objetivo de caracterizar como cada um dos fatores influencia no balanço de desempenho e isolamento. Apresenta-se o efeito das interfaces de comunicação, seguido da influência das anomalias e do isolamento e, por fim, das funções de virtualização.

Considerando o cenário de aplicação de anomalias, a Figura 4 ilustra as diferentes capacidades de vazão e picos de latência das interfaces avaliadas, também apresentando o efeito da aplicação de isolamento de recursos *cgroups*. Em média, *macvlan* apresentou 4,74% mais vazão que *veth direct*, que, por sua vez, alcançou 22,29% mais que a configuração *veth bridge*. A relação da latência foi mais intrincada, na qual a *macvlan* apresentou o maior aumento de latência durante anomalias de memória. A interface *veth direct* exibiu as menores médias de latência, demonstrando valores 16,17% menores que

a implementação tradicional de *veth bridge*. No entanto, é importante salientar que a anomalia de memória afetou fortemente todas as interfaces avaliadas.

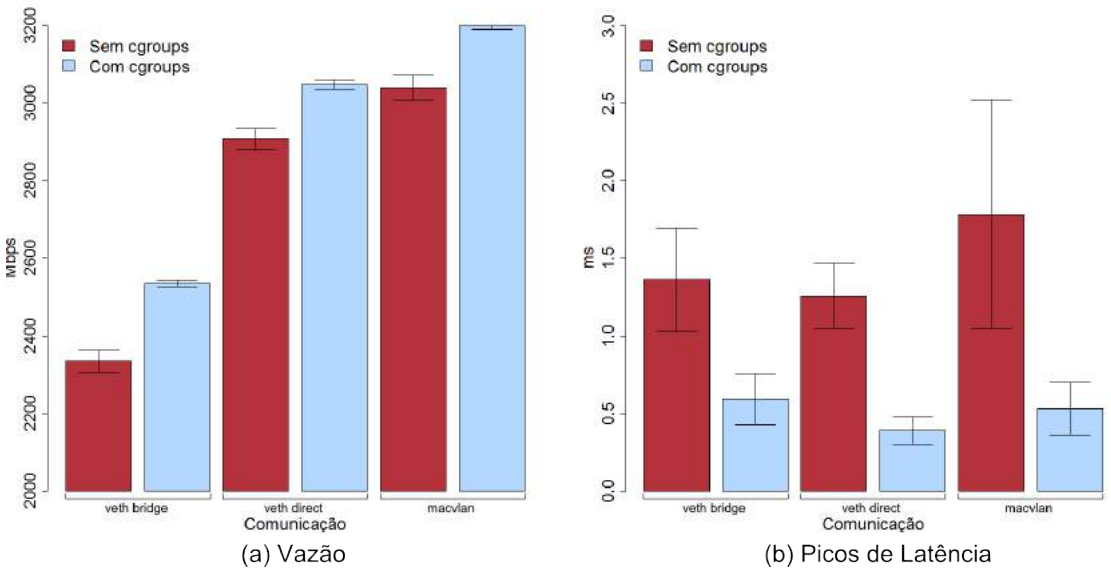

Figura 4. Desempenho das interfaces de comunicação no cenário de anomalias e efeito do *cgroups* na capacidade de vazão (à esquerda) e nos picos de latência (à direita).

Com o objetivo de determinar o custo inerente de desempenho encontrado em situações de extenuação de recursos pode-se considerar os valores observados na presença de anomalias e ausência de ferramentas de isolamento e contrastá-los com os valores encontrados no cenário ideal. Nesse caso, as anomalias deduziram, em média, 312,39 Mbps da capacidade de vazão e acrescentaram 1,034 ms nos picos de latência, representando uma variação de -10,39% e +129,78% em relação aos seus respectivos valores no caso ideal.

Em contraste, o impacto das anomalias na presença de isolamento pode ser determinado examinando a capacidade de vazão encontrada nesse cenário e comparando-a com o caso ideal. Na presença de ambos mecanismos de isolamento a perda de vazão média ocasionada pelas anomalias avaliadas foi de 115,31 Mbps, ou o equivalente a 3,81% da vazão apresentada na ausência de anomalias. Os respectivos valores de *jitter*, latência média e máxima correspondentes não foram significativos.

Em uma análise das ferramentas específicas de isolamento, o *AppArmor* manifestou consistentemente um impacto negativo de 32,48 Mbps na capacidade de vazão, representando um decréscimo de 1,073%, quando comparado com os valores obtidos aplicando anomalias. Em contrapartida, o isolamento de recursos *cgroups* teve um efeito majoritariamente positivo, mitigando em média o impacto das anomalias em 62,59% na capacidade de vazão e 60,26% nos picos de latência, enquanto que a latência média foi reduzida em 0,0459 ms. O único efeito negativo perceptível da aplicação do *cgroups* foi um leve aumento de 4,8 us no *jitter*, quando contraposto com o caso ideal (livre de anomalias).

Considerando individualmente as anomalias avaliadas, a Figura 5 apresenta os diferentes valores de capacidades de vazão e picos de latência observados juntamente com o efeito da aplicação do isolamento de recursos *cgroups*. Como é possível constatar abaixo, a privação de memória foi a anomalia que infligiu o maior impacto quando desprovido de isolamento, atingindo 602,85 Mbps de perda em capacidade de vazão e 2,18 ms de aumento nos picos de latência. Entretanto, o isolamento de recursos do *cgroups* foi capaz de reivindicar mais desempenho da anomalia de memória do que de qualquer outra, reduzindo o impacto negativo em 94,38% na capacidade de vazão. As anomalias manifestadas em I/O e Disco tiveram o menor impacto sem isolamento, mas, em compensação, a aplicação do isolamento teve o menor efeito nesses casos.

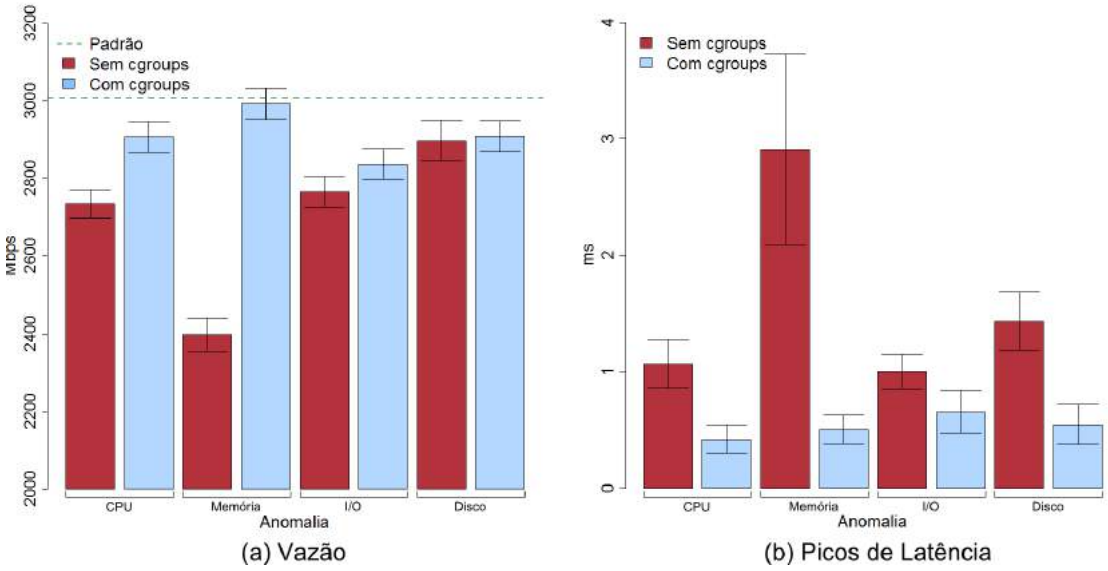

Figura 5. Efeito do *cgroups* nas anomalias, na capacidade de vazão (à esquerda) e nos picos de latência (à direita).

As três funções de rede analisadas obtiveram taxas muito similares de capacidade de vazão, no entanto, foi observada uma relação de ordem estatisticamente significativa na latência média e máxima. A latência observada na função NAT foi, em média, 0.0352 ms maior que a observada em DPI, a qual foi 0.0231 ms maior que a latência do *Firewall*. Nenhuma das anomalias avaliadas teve relação significativa com quaisquer das VNFs estudadas.

6. Discussão

Esta seção desenvolve pontos de análise sobre os resultados obtidos e os relaciona com questões de isolamento e desempenho em *containers* assim como suas implicações para NFV. Primeiramente, avalia-se as tecnologias de isolamento quanto à sua viabilidade, seguido de uma elucidação sobre a diversidade de resultados das anomalias e funções de rede. Por fim, elabora-se um panorama das principais propriedades das configurações de comunicação relativas à aplicação no encadeamento de funções virtualizadas.

A tecnologia de isolamento de recursos *cgroups* se mostrou mais eficaz na redução do impacto de anomalias de CPU e memória, nas quais sua contribuição mais expressiva ocorreu na diminuição dos picos de latência e aumento na capacidade de vazão. Contrastando com o caso ideal (livre de anomalias), seu único custo estatisticamente significativo, ainda que diminuto, ocorreu em *jitter* (4,8 us), mas em vista da apreciável diminuição dos picos de latência, tal observação não representa um empecilho à aplicação de *cgroups* a NFV.

Considerando a tecnologia de isolamento de segurança *AppArmor*, o único custo observável foi verificado na capacidade de vazão. Tal custo pode ser considerado diminuto em comparação com o típico custo de desempenho apresentado por virtualização de *hardware*. A escolha de tecnologias como *AppArmor* e *SELinux* se vê facilmente justificada em luz da proteção disponibilizada às possíveis brechas de segurança *zero day* presentes no *kernel* hospedeiro.

Das anomalias escolhidas, a que apresentou maior impacto na ausência de isolamento foi a privação de memória. No entanto, a aplicação do isolamento de recursos resultou numa redução quase completa desses efeitos (94,38%). Pode-se concluir que a anomalia de memória representa a maior vulnerabilidade de recursos mitigada por *cgroups*, no contexto avaliado. Por outro lado, a aplicação do isolamento de recursos teve seu menor efeito nas anomalias de IO e disco, cujo impacto é derivado de chamadas de sistema (*sync* e *write*, respectivamente). A capacidade dos *containers* de fazer chamadas de sistema ao *kernel* hospedeiro da máquina representa um ponto de vulnerabilidade, não só expondo recursos, mas principalmente oportunizando lacunas de segurança. É importante frisar que um *container* deve receber apenas o acesso estritamente necessário para seu funcionamento. Nesse caminho, o uso de *capabilities* pela *libcontainer* e *lxc* representa um passo na direção certa, mas ainda existem requisitos de segurança relativos a *containers* a serem atendidos.

Os resultados permitem discernir as funções de rede escolhidas, considerando que os mecanismos com requisitos de processamento mais onerosos (como alteração de cabeçalhos e redirecionamento interno) resultaram em diferenças perceptíveis no quesito de latência. Apesar de não ocasionar correlação significativa com as anomalias avaliadas, essa diversidade de mecanismos serviu para adicionar representatividade aos resultados, uma vez que diferentes mecanismos ativos foram avaliados.

Dentre as interfaces de comunicação avaliadas, *macvlan* em modo *bridge* apresentou as maiores capacidades de vazão, como previamente relatado. No entanto, é importante salientar que a reformulação de interfaces *veth* para comunicação direta entre *containers* manifestou capacidade comparável e obteve os melhores valores de latência média e de picos de latência, em especial nos casos de disputa de memória. Esses resultados indicam que boa parcela do déficit de desempenho apresentado por interfaces *veth* em estudos anteriores [Claassen et al. 2016] esteja mais relacionada aos níveis de indireção associados a sua mais comum implementação do que a alguma característica intrínseca de interfaces *veth*. Enquanto isso, *macvlan* apresentou os maiores picos de latência durante ataques à memória. Esse efeito pode ser justificado pelo fato de que a *pseudobridge* utilizada na comunicação *macvlan inter-container* é implementada via RAM, deixando-a possivelmente mais vulnerável à privação de memória principal.

Uma interessante relação de compromisso se apresenta na escolha entre essas interfaces de comunicação. Em termos de desempenho, diferentes prioridades entre capacidade de vazão e características de latência podem levar a diferentes escolhas de interfaces, em particular na configuração de cadeias de funções de rede. Em termos de isolamento, a configuração *veth direct* limita a capacidade de comunicação apenas ao *container* possuidor da interface pareada, enquanto que *macvlan bridge* permite a comunicação direta com todas interfaces que compartilhem a *bridge*. Em termos de configurabilidade, *veth direct* requer a instanciação de um par de interfaces para cada possível canal de comunicação *inter-container*, por outro lado, uma instância de *macvlan slave* possibilita comunicação entre múltiplos *containers*. Além disso, configurações dedicadas de *veth* facilitam a limitação de tráfego via *tc*, enquanto que tal tarefa é mais complicada com múltiplas interfaces se comunicando via *macvlan bridge*.

7. Conclusão

Este artigo apresentou uma análise das características de isolamento e desempenho de *containers* considerando múltiplas tecnologias de isolamento, comunicação, funções de rede e anomalias de execução. Com base nesses experimentos foi possível averiguar o impacto de exercer isolamento no desempenho de funções de rede quando o sistema está sob extenuação de recursos. Os experimentos realizados não só evidenciaram um baixo custo no desempenho de rede para o emprego de isolamento, mas, notavelmente, demonstraram quanto o isolamento de recursos pode contribuir para o desempenho em situações anômalas. Os resultados sugerem que *containers* conseguem prover um ambiente propício para o desempenho de NFV sem perder de vista questões de segurança e isolamento.

Além disso, foi apresentada uma avaliação de como as tecnologias de comunicação mais habitualmente utilizadas em *containers* influenciam na relação de isolamento e desempenho.Tal avaliação incluiu a reformulação de uma interface de comunicação que resultou em observações inovadoras sobre suas capacidades de desempenho e relações de compromisso na configuração de cadeias de funções de rede.

Como trabalho futuro, pretende-se explorar alguns fatores de teste adicionais. Em termos de tecnologia de comunicação, *overlay networks*, mecanismos de comunicação externa e tecnologias como DPDK representam aspectos promissores que poderiam se beneficiar de estudos semelhantes. Tais fatores podem desvendar propriedades de comunicação entre *containers* localizados em diferentes dispositivos físicos de virtualização, de forma complementar às tecnologias avaliadas neste artigo. Em termos de plataformas de virtualização, *containers* de sistema, particularmente o LXD, merecem o mesmo método de análise. Considerando tecnologias de isolamento, outros mecanismos como *SELinux* e *Smack* representam alternativas interessantes. Um dos desafios no contexto de isolamento é referente a como ataques de segurança podem influenciar no desempenho das funções vizinhas. Tal cometimento se vê dificultado pela variedade significativa de estratégias de segurança e de invasão, cada qual com seu espectro de impacto. Em termos de anomalias, outros artifícios podem ser usados para simular cargas de trabalho mais específicas, as quais possibilitam a descoberta de novas interações com as tecnologias usuais. Em específico, avaliar a resposta de um ataque interno de *Denial of Service* pode elucidar características essenciais das tecnologias de comunicação e isolamento, as quais podem ser determinantes para a adequação de *containers* em NFV.

Referências

Anderson, J., Hu, H., Agarwal, U., Lowery, C., Li, H., and Apon, A. (2016). Performance considerations of network functions virtualization using containers. *2016 International Conference on Computing, Networking and Communications, ICNC 2016*.

Barbosa, J., Oliveira, V., Bandini, M., Schulze, B., and Mury, A. (2014). N-Clusters : Ferramenta para a Gerência de Ambientes de Computação Massivamente Paralela e Distribuída. *XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 1003–1010.

Claassen, J., Koning, R., and Grosso, P. (2016). Linux containers networking: Performance and scalability of kernel modules. *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 713–717.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and Linux containers. *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Heideker, A. and Kamienski, C. (2016). Gerenciamento Flexível de Infraestrutura de Acesso. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Mijumbi, R., Serrat, J., Gorricho, J.-l., Bouten, N., Turck, F. D., and Member, S. (2016). Network Function Virtualization : State-of-the-Art and Research Challenges. *IEEE Communications Surveys and Tutorials*, 18(1):236–262.

Morabito, R., Kjällman, J., and Komu, M. (2015). Hypervisors vs. lightweight virtualization: A performance comparison. *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, pages 386–393.

Netto, H., Lung, L. C., Correia, M., and Luiz, A. F. (2016). Gerenciamento Flexível de Infraestrutura de Acesso. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Souza, J., Santos, A., Bandini, M., and Kl, H. (2016). Rufus : Ferramenta para o Gerenciamento de Infraestrutura para a Execução de Aplicações em Containers. *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.