Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Virtualização de funções de redes para processamentos de pacotes em comutadores OpenFlow heterogêneos

João Victor Guimarães de Oliveira

JUIZ DE FORA
MARÇO, 2021

Virtualização de funções de redes para processamentos de pacotes em comutadores OpenFlow heterogêneos

João Victor Guimarães de Oliveira

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Luciano Jerez Chaves

Co-orientador: Alex Borges Vieira

JUIZ DE FORA

MARÇO, 2021

VIRTUALIZAÇÃO DE FUNÇÕES DE REDES PARA PROCESSAMENTOS DE PACOTES EM COMUTADORES OPENFLOW HETEROGÊNEOS

João Victor Guimarães de Oliveira

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Luciano Jerez Chaves
Mestre em Ciência da Computação

Edelberto Franco Silva
Doutor em Ciência da Computação

Roberto Massi de Oliveira
Doutor em Engenharia Elétrica

JUIZ DE FORA

19 DE MARÇO, 2021

Aos meus amigos e irmãos.

Aos pais, pelo apoio e sustento.

Resumo

Diante da grande ascensão das tecnologias em redes de computadores na busca de um melhor desempenho das aplicações, e também da evolução dos paradigmas de virtualização das funções de rede e de redes definidas por software, neste trabalho buscamos uma técnica para aperfeiçoar o balanceamento do tráfego e a escalabilidade de funções virtuais de rede. Assim, implementamos uma função virtual de rede através de regras distribuídas entre comutadores SDN programáveis, o que possibilita a adoção eficiente da virtualização das funções de rede em cenários que necessitem de processamento escalável de pacotes. Nesse sentido, propomos dois mecanismos para gerência dinâmica da função virtual: um mecanismo de escalabilidade do plano de dados e um mecanismo de balanceamento de carga. O primeiro monitora os recursos e ajusta o número de comutadores de acordo com a demanda da rede. Já o mecanismo de balanceamento identifica os novos tráfegos e instala as regras de fluxos nos comutadores adequados. Avaliamos o mecanismo de balanceamento experimentalmente e os resultados mostraram um aumento na vazão agregada de aproximadamente 27% quando comparado com uma política simples. Já o mecanismo de escalabilidade foi avaliado por meio de simulações, sendo que os resultados mostram como a combinação desses mecanismos permite otimizar o uso dos recursos e atender toda a demanda da rede com qualidade e eficiência.

Palavras-chave: Redes de computadores, Redes Definidas por Software, Balanceamento de fluxos, Virtualização das Funções de Rede.

Abstract

With the progress of computer network technologies searching for improved performance of applications and the evolution of the Network Function Virtualization and Software Defined Networking, we propose techniques to enhance the load balancing and the scalability of virtual network functions. Thus, we implemented a virtual network function with rules distributed among programmable SDN switches, enabling efficient virtualization of network functions in scenarios that require scalable packet processing. In this sense, we propose two mechanisms for dynamic management of the virtual function: a data plan scalability mechanism and a load balancing mechanism. The former monitors the resources and adjusts the number of switches according to the demand of the network. The latter identifies new traffic flows and installs the flow rules in the appropriate switches. We evaluated the load balancing mechanism experimentally, and the results revealed an increase in the throughput of aggregate throughput by approximately 27% compared to a simple policy. Simulations evaluated the scalability mechanism, and the results highlighted how the combination of both mechanisms optimizes the use of resources while meeting the network demand with quality and efficiency.

Keywords: Computer networks, Software Defined Networking, Load balancing, Network Function Virtualization.

Agradecimentos

Aos meus pais, por todo amor, incentivo e confiança nas minhas escolhas.

Aos professores Luciano Jerez Chaves e Alex Borges Vieira pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

A minha namorada, Talita Valverde, por todo carinho e o apoio nos momentos difíceis durante esses anos.

Agradeço aos meus amigos, em especial ao Pedro Bellotti pelo apoio durante o desenvolvimento desse trabalho.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

#EuAmoMais

"A persistência é o caminho do êxito"

Charles Chaplin

Conteúdo

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Lista de Abreviações

API	Application Program Interfaces
CPU	Central Process Unit
DL	DownLink
DPDK	Data Plane Development Kit
ETSI	European Telecommunication Standards Institute
HW	Hardware
IP	Internet Protocol
LTS	Long-Term Support
MANO	Management and Orchestration
ns-3	Network Simulator 3
NFV	Network Function Virtualization
ONF	Open Networking Foundation
OXM	OpenFlow eXtensible Match
P-GW	Packet GateWay
QoS	Qualidade de Serviço
SDN	Software Defined Network
SFC	Service Function Chaining
SW	Software
TCAM	Ternary Content-Addressable Memory
UDP	User Datagram Protocol
UL	UpLink
VNF	Virtual Network Function
VM	Virtual Machine

1 Introdução

A crescente demanda por conectividade à Internet tem forçado a evolução das tradicionais arquiteturas de redes de computadores para um modelo mais sustentável. De acordo com (CISCO SYSTEMS., 2020), estima-se que em 2023 existam 5,3 bilhões de usuários conectados a Internet, cerca de 66% da população mundial. Logo, as redes devem ser capazes de oferecer acesso ubíquo aos usuários sem aumentar significativamente o custo de implantação e operação para os provedores de infraestrutura. Dentre as principais limitações da arquitetura tradicional está a dependência de *hardware* dedicado para a implementação de funções de rede, como roteadores, *firewalls*, *gateways*, balanceadores de carga, etc. (CHAVES et al., 2015). Este modelo requer um alto investimento para a aquisição e instalação de equipamentos, normalmente com recursos superdimensionados capazes de atender ao picos de tráfego e acomodar a demanda em curto prazo.

Com o objetivo de diminuir os custos e aumentar a flexibilidade, o paradigma de Virtualização das Funções de Rede (NFV, do Inglês *Network Functions Virtualization*) tem se mostrado uma alternativa atraente, recebendo atenção tanto da academia como da indústria (ETSI, 2012). Através da separação entre o *software* responsável pela função de rede e o *hardware* onde ele é executado, o NFV permite o uso compartilhado de recursos computacionais de propósito geral para a execução de instâncias independentes de Funções Virtuais de Rede (VNFs, do Inglês *Virtual Network Functions*). A abordagem convencional consiste em implementar as VNFs como *software* rodando em máquinas virtuais (VMs, do Inglês *Virtual Machines*) ou *containers* que são instanciados em servidores na nuvem. Dessa forma, as VNFs podem ser personalizadas e atualizadas com mais agilidade enquanto os recursos de *hardware* passam a ser melhor utilizados, visto que as VNFs podem ser alocadas dinamicamente de acordo com a demanda da rede (YI et al., 2018).

Como um facilitador para a implantação do NFV, o paradigma das Redes Definidas por *Software* (SDN, do Inglês *Software Defined Networks*), juntamente com o protocolo OpenFlow, apresentam uma abordagem moderna para o gerenciamento de tráfego em uma rede de comutadores (ONF, 2012). A inteligência e o controle da rede são logi-

camente centralizados em *software* enquanto o plano de dados é fisicamente distribuído em *hardware* programável e otimizado para o encaminhamento de pacotes. Dessa forma, as tomadas de decisão são facilitadas pela visão global da rede, permitindo ao SDN direcionar os pacotes até as VMs para oferecer o encadeamento adequado do tráfego entre as VNFs e prover serviços de forma dinâmica através das Cadeias de Funções de Serviço (SFCs, do Inglês *Service Function Chains*) (KREUTZ et al., 2015).

Entretanto, a flexibilidade do NFV vem acompanhada de novos desafios, dentre eles a escalabilidade (WANG et al., 2016). Algumas VNFs requerem servidores de alto desempenho para lidar com intensas cargas de trabalho, principalmente quando precisam realizar inspeção individual de pacotes em grandes fluxos de dados. Mesmo com os recentes avanços tecnológicos, o processamento de pacotes por servidores de propósito geral ainda apresenta desempenho moderado quando comparado às plataformas de *hardware* de propósito específico (NGUYEN et al., 2016). Nesse contexto, uma das estratégias comumente adotada em trabalhos na literatura para aumentar a escalabilidade é replicar as VNFs e utilizar o SDN para balancear a carga de trabalho entre as instâncias disponíveis (LAGHRISSI; TALEB, 2019).

Visto que a natureza de algumas VNFs está estritamente relacionada com a inspeção massiva de pacotes e possíveis ações sobre eles, (CHAVES; GARCIA; MADEIRA, 2017) propuseram uma abordagem diferente para a virtualização desta categoria de funções de rede: substituir o *software* em uma VM por um conjunto de regras programadas em comutadores OpenFlow, de modo a explorar o alto desempenho do *hardware* dedicado. Como um único comutador pode não ser suficiente para processar todo o tráfego, a escalabilidade desta VNF pode ser alcançada com a distribuição das regras de fluxo entre vários comutadores e o adequado balanceamento do tráfego entre eles.

Como uma das contribuições deste trabalho, foi evoluída a proposta original de (CHAVES; GARCIA; MADEIRA, 2017) para incorporar um plano de dados heterogêneo, que combina comutadores em *hardware* e em *software* na instanciação da VNF, de modo a explorar as características distintas destas plataformas, como o tamanho das tabelas de fluxos, o atraso médio no processamento dos pacotes e a vazão máxima suportada. Essa abordagem oferece flexibilidade ao provedor da infraestrutura: ele pode optar por comu-

tadores OpenFlow com *hardware* dedicado ao processamento de pacotes para explorar o alto desempenho do dispositivo sem perder a programabilidade oferecida pelo mesmo; e/ou pode optar por comutadores OpenFlow em *software*, que flexibiliza a instanciação da VNF em servidores de propósito geral, mesmo que isso implique em alguma redução na capacidade de processamento da função virtual. Um cenário heterogêneo como esse poderia ser adotado em um ambiente onde o provedor da infraestrutura mantenha um (ou alguns) comutadores físicos em constante operação, garantindo continuidade no oferecimento do serviço, mas instancie outros comutadores virtuais para atender demandas temporárias de tráfego.

Dois mecanismos para gerência dinâmica desta VNF foram propostos. Um mecanismo de *escalabilidade do plano de dados*, que decide qual o tipo e quantos comutadores OpenFlow que a VNF necessita, e de um mecanismo de *balanceamento de carga*, que distribui fluxos de pacotes entre os comutadores heterogêneos, considerando suas especificações. Com o auxílio do controlador SDN local, o mecanismo de escalabilidade do plano de dados monitora o tráfego da rede e os recursos disponíveis nos comutadores. Quando necessário, ele ajusta o número de comutadores ativos na infraestrutura para atender à demanda da rede. Por sua vez, o mecanismo de balanceamento de carga identifica reativamente a chegada de novos tráfegos e instala com precisão as novas regras nos comutadores adequados, considerando as particularidades dos comutadores em *hardware* e *software*. Além da alocação inicial dos tráfegos, o mecanismo de balanceamento de carga também lida com sua realocação posterior, tendo como objetivo o uso aprimorado dos recursos da rede.

Parte desta proposta foi avaliada em um pequeno ambiente de testes utilizando equipamentos reais e comparando o mecanismo de balanceamento dinâmico de carga com uma política estática simples, sem escalabilidade do plano de dados. A *política estática* distribui novos fluxos aleatoriamente entre os comutadores. Já a *política dinâmica* considera as especificidades dos comutadores de *hardware* e *software* na alocação inicial dos fluxos e posterior realocação dos mesmos, de modo a extrair o máximo que cada tipo de infraestrutura pode oferecer. Os resultados do experimento mostraram que, em uma rede sobrecarregada, a política de balanceamento de carga dinâmica aumentou a vazão da

VNF em aproximadamente 27% em comparação com a política estática. Posteriormente, foi implementado o cenário completo no simulador de rede ns-3 para investigar a operação conjunta dos mecanismos de balanceamento de carga e de escalabilidade dos dados. Os resultados da simulação atestam, para todas as cargas de trabalho que foram avaliadas, que a combinação desses mecanismos permite que a VNF atenda a demanda da rede com qualidade enquanto economiza recursos.

Os próximos capítulos deste trabalho estão organizados da seguinte maneira: o Capítulo 2 resume os fundamentos teóricos acerca das tecnologias utilizada; o Capítulo 3 apresenta uma revisão da literatura sobre o tema; o Capítulo 4 revela o cenário adotado neste trabalho e detalha os dois mecanismos propostos; o Capítulo 5 descreve a metodologia e apresenta os resultados para um cenário real não escalável; o Capítulo 6 descreve a metodologia e apresenta os resultados obtidos para um cenário simulado escalável; e, por fim, o Capítulo 7 expõe as conclusões deste trabalho.

2 Fundamentação Teórica

Neste capítulo apresento a fundamentação teórica relacionada a este trabalho. Especificamente, abordamos os paradigmas de Virtualização da Funções de Rede e Redes Definidas por Software, além de detalhar o protocolo OpenFlow.

2.1 Virtualização das Funções de Rede

O conceito de virtualização permite uma abstração entre serviços e recursos físicos. Essa é uma maneira de reduzir despesas e aumentar a eficiência de ambientes computacionais. Logo, com a virtualização é possível que diversas aplicações executem simultaneamente sobre o mesmo *hardware* físico compartilhado, contribuindo assim para uma melhor utilização dos recursos computacionais.

Diversas funções de redes tradicionais, como *gateways*, *firewalls* e balanceadores de carga, utilizam *hardware* dedicado com implementações definidas pelo fabricante do equipamento. Essa condição dificulta o desenvolvimento de novos serviços e a possibilidade de melhoria na rede, pois não é possível modificar o funcionamento do *hardware*, exceto por configurações pré-estabelecidas pelos fabricantes. O paradigma de Virtualização das Funções de Rede (NFV) surgiu para melhorar essa questão. O NFV foi apresentado pelo *European Telecommunication Standards Institute* (ETSI), e pode ser definido como a separação do *software* que realiza o processamento das funções de rede do *hardware* dedicado que executa este *software*. Com essa separação, o NFV permite consolidar diversas funções de rede em uma infraestrutura de *hardware* compartilhada (MIJUMBI J. SERRAT; BOUTABA, 2015). Essa otimização reduz custos de manutenção e de operação, pois o *hardware* dedicado pode ser substituído por computadores de menor custo (LI; CHEN, 2015).

A Figura 2.1 apresenta uma visão em alto nível da arquitetura do NFV. No topo da arquitetura encontramos as Funções Virtuais de Rede (VNFs). Cada VNF representa um elemento da rede que pode ser virtualizado, como roteadores, *firewalls*, entre outros

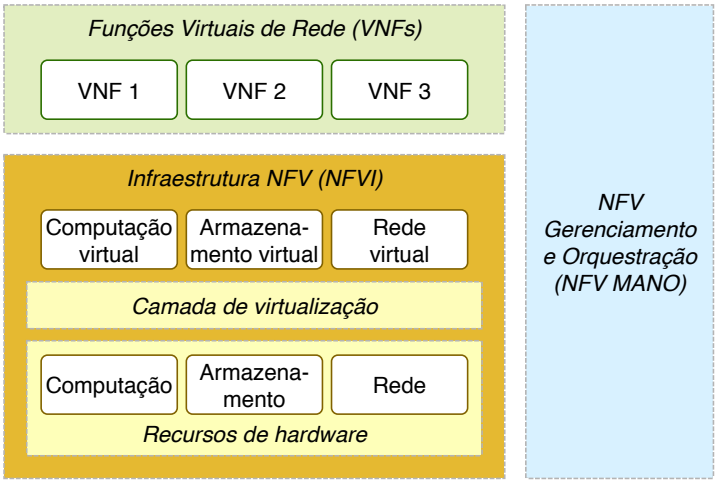


Figura 2.1: Arquitetura NFV.

(LI; CHEN, 2015). Logo abaixo encontramos os recursos virtuais, que é a união de recursos de *software* e *hardware*, criando um ambiente que permita a instanciação de uma VNF. Para administrar todos os elementos da arquitetura, temos o componente de Gerenciamento e Orquestração (MANO, do Inglês *Management and Orchestration*), que é responsável por garantir que exista processamento, armazenamento e recursos disponíveis para cada VNF (ABDELWAHAB et al., 2016).

2.2 Redes Definidas por Software

Redes Definidas por *Software* (SDN) é um paradigma criado pela *Open Networking Foundation* (ONF) que objetiva a separação entre os planos de dados e de controle da rede. Isso permite centralizar o plano de controle em software, substituindo diversos algoritmos distribuídos tradicionais, que são executados individualmente em cada nó da rede. Assim, com a camada de controle centralizada, o processo de tomadas de decisão é melhorado (YOUSAF et al., 2017).

A Figura 2.2 apresentada a arquitetura SDN. Ela é composta por três camadas que são acessadas através de Interfaces de Programação de Aplicativos (APIs, to Inglês *Application Program Interfaces*). A *camada de aplicação* contém os serviços que operam sobre a rede. A *camada de controle* é responsável pela tomada de decisão e supervisão dos recursos da infraestrutura. A *camada de infraestrutura* contém os dispositivos e elementos de rede responsáveis pelo encaminhamento dos pacotes.

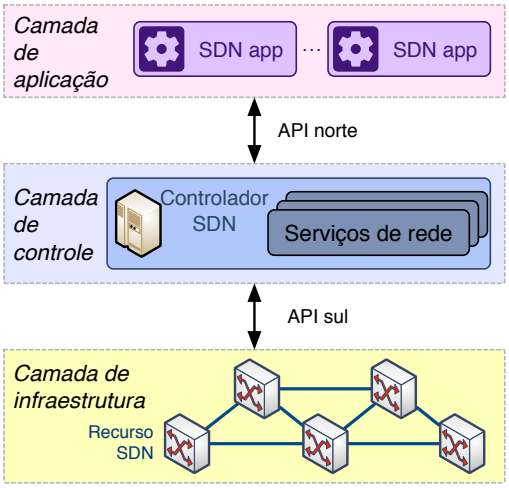


Figura 2.2: Arquitetura SDN.

No SDN, o controle da rede é realizado através de um *software* centralizado, que pode analisar o estado da rede em tempo real de modo a otimizar as rotas de encaminhamento dos pacotes. Isso aumenta a flexibilidade de configuração e de gerenciamento da rede.

Tanto o SDN quanto o NFV possuem como objetivo aumentar a flexibilidade, reduzir custos e permitir a escalabilidade dos serviços. Ambos paradigmas são independentes entre si, mas podem ser usados simultaneamente. O NFV pode virtualizar elementos do SDN, como por exemplo o controlador. Por outro lado, o SDN permite programar com facilidade as conexões entre as VNFs (LI; CHEN, 2015).

2.3 Protocolo OpenFlow

O OpenFlow é um protocolo criado para viabilizar a comunicação entre as camadas de controle e infraestrutura em redes SDN (MCKEOWN et al., 2008). Ele é implementado tanto nos dispositivos da infraestrutura da rede quanto no controlador. O OpenFlow é baseado em regras, que podem ser programadas pelo controlador nos comutadores para identificar os fluxos de dados na rede e realizar ações específicas sobre os pacotes.

A implementação atual do protocolo OpenFlow abrange os componentes e as funções básicas de um comutador, como pode ser visto na Figura 2.3. O canal OpenFlow realiza a conexão entre o comutador e o controlador. Essa interface permite que o controlador possa administrar o comutador, recebendo e enviando pacotes para a rede. O

canal de controle do comutador pode ser administrado por um ou por vários controladores simultaneamente. Todas as mensagens trocadas nestes canais devem seguir o padrão especificado pelo protocolo.

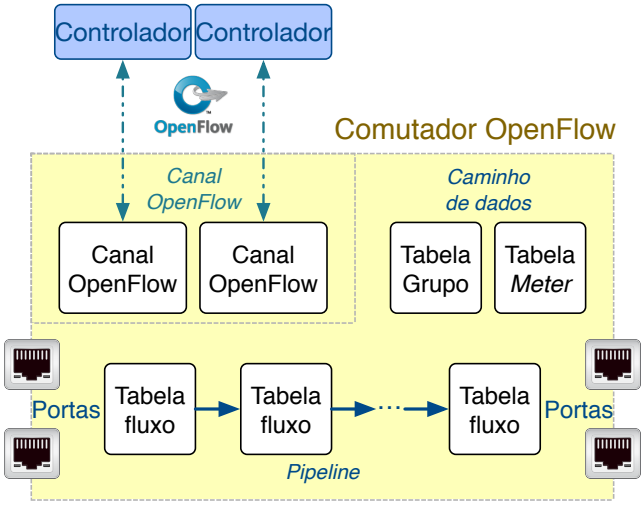


Figura 2.3: Componentes de um comutador OpenFlow.

O caminho de dados do comutador OpenFlow é composto de uma ou mais tabelas de fluxo, uma tabela de grupo e uma tabela de medições (*meter*). As tabelas de fluxo inspecionam e encaminham pacotes de acordo com as regras instaladas pelo controlador. Cada regra é composta de campos do tipo *OpenFlow eXtensible Match* (OXM), que analisam valores nos cabeçalhos dos pacotes para identificar o fluxo de dados correspondente. Além desses campos, as regras de fluxos possuem contadores de estatísticas e um conjunto de instruções que serão aplicados aos pacotes de um fluxo de dados.

O processamento dos pacotes no comutador OpenFlow começa na primeira tabela de fluxo, seguindo a prioridade das regras instaladas, podendo continuar pelas tabelas seguintes. Caso seja encontrado uma correspondência na tabela de fluxos, as instruções daquela regra são realizadas. Caso contrário, o pacote é enviado ao controlador, que extrai as informações necessárias para a tomada de decisão. As instruções nas regras de fluxo podem conter ações de encaminhamento de pacotes para as portas de saída do comutador, modificações no conteúdo do pacote ou mesmo direcionamento do pacote para processamento nas tabelas de *meter* e grupo.

A tabela de grupo consiste de um conjunto de instruções para a execução de políticas de encaminhamento mais sofisticadas, como balanceamento de carga entre enlaces, inundação da rede, recuperação em caso de enlaces com falhas, etc. Já a tabela de *meter* possui regras usadas para monitorar a vazão dos fluxos e, caso necessário, descartar pacotes excedentes para implementar mecanismos como limitadores de vazão.

3 Trabalhos relacionados

É fato que o NFV é uma alternativa atraente para reduzir os custos e aumentar a flexibilidade de redes (YI et al., 2018; MIJUMBI et al., 2015). Porém, essa flexibilidade contrapõe desempenho e escalabilidade. Assim, para lidar com intensas cargas de trabalho, pode ser necessário combinar poderosos servidores de processamento para replicar as VNFs e realizar o balanceamento de carga entre elas (WANG et al., 2016).

(CEROVIC et al., 2018) e (FEI et al., 2020) fizeram extensas revisões sobre tecnologias recentes para acelerar o manuseio de pacotes em ambientes virtuais. Entre estas tecnologias, destacamos o FD.io[1]: uma coleção de bibliotecas para viabilizar serviços programáveis e flexíveis em uma plataforma de *hardware* genérica. O componente chave do FD.io é a biblioteca de processamento vetorial de pacotes, que permite aos desenvolvedores especificar os fluxos de processamento de pacotes através de grafos variados. O FD.io aproveita dos recursos do Conjunto de Desenvolvimento de Plano de Dados da Intel (DPDK, do Inlgês *Data Plane Development Kit*)[2] para acelerar o plano de dados das VNFs por meio de uma camada de abstração. Esta camada fornece uma interface padrão de programação para bibliotecas, aceleradores de *hardware* e outros elementos do sistema operacional. Outras soluções construídas com o DPDK incluem comutadores OpenFlow em *software*, como é o caso do Open vSwitch[3] (PFAFF et al., 2015), e também de plataformas como o NetVM e OpenNetVM (WOOD et al., 2015).

Outro ponto amplamente estudado na literatura é a escalabilidade em NFV. Algumas categorias de VNFs necessitam de alto desempenho para realizar a inspeção individual dos pacotes em grandes fluxos de dados. Uma das estratégias de escalabilidade neste contexto é replicar as VNFs e balancear a carga de trabalho entre as réplicas disponíveis. O balanceamento de carga é a técnica que divide uma demanda por vários recursos computacionais em uma rede, como enlaces e servidores, de modo a otimizar a utilização dos mesmos. Este é um tópico atraente e bastante explorado na literatura,

[1] ⟨https://www.fd.io⟩
[2] ⟨https://www.dpdk.org⟩
[3] ⟨https://www.openvswitch.org⟩

principalmente quando envolve os paradigmas SDN e NFV.

A existência de uma infraestrutura SDN atua como um facilitador para a implementação das técnicas de balanceamento de carga, dada a fácil programabilidade destas redes. Os trabalhos de (LAGHRISSI; TALEB, 2019) e (CARPIO; DHAHRI; JUKAN, 2017) apresentam discussões sobre o problema de posicionamento de funções virtuais de rede em ambientes que adotam NFV e SDN. Os autores dedicam atenção especial ao discutir soluções de posicionamento de VNFs que sejam cientes do balanceamento da carga nos enlaces, incluindo aspectos em relação à replicação de instâncias destas funções.

Alternativamente, o SDN também pode ser usado para implementar diretamente as funções virtuais de rede, principalmente aquelas que demandam de inspeção individual de pacotes. (KAUR; KAUR; GUPTA, 2017) usam uma rede de comutadores OpenFlow para implementar um *firewall* (sem estado) distribuído, espalhando as regras de processamento de fluxos entre os comutadores da rede e eliminando o ponto único de falha da arquitetura tradicional. Outro cenário pode ser visto em (RODRIGUES et al., 2015), onde os autores propõem a adoção de um comutador OpenFlow ligado à um controlador centralizado para substituir os equipamentos especializados de balanceamento de carga em nível de aplicação. Com a arquitetura proposta, é possível implementar diferentes políticas para distribuir requisições entre servidores Web, oferecendo escalabilidade ao serviço. Num contexto semelhante, (AN; KIESS; PEREZ-CAPARROS, 2014) propõem a virtualização do plano de dados do *Gateway* de Pacotes (P-GW, do Inglês *Packet Gateway*) utilizado na arquitetura das redes móveis 4G. Preocupados com a escalabilidade desta VNF, os autores adotam um conjunto local de réplicas e utilizam um par de comutadores OpenFlow para distribuir o tráfego entre as instâncias ativas.

Como uma alternativa ao trabalho de (AN; KIESS; PEREZ-CAPARROS, 2014), a proposta de (CHAVES; GARCIA; MADEIRA, 2017) prevê a virtualização do plano de dados do *gateway* P-GW unicamente através de comutadores SDN. Para isso, eles implementam as tarefas sob responsabilidade do *gateway* como regras de fluxo OpenFlow, incluindo aquelas necessárias para a classificação individual dos fluxos de pacotes nos túneis usados pela rede 4G, além de limitadores de vazão e contabilização de estatísticas para fins de cobrança. Como a quantidade de regras cresce proporcionalmente com o

número de fluxos ativos na rede e, considerando as limitações no número máximo de regras em comutadores OpenFlow de prateleira, os autores sugerem que as regras que implementam o P-GW sejam distribuídas entre um conjunto de comutadores idênticos para aumentar a escalabilidade. Para isso, eles propuseram um mecanismo adaptativo para ativar/desativar os comutadores e mover as regras de fluxos entre eles de acordo com a demanda da rede.

Para flexibilizar a proposta de (CHAVES; GARCIA; MADEIRA, 2017), é interessante que a expansão de capacidade da VNF possa acontecer com a agregação de comutadores OpenFlow heterogêneos, incluindo também comutadores virtualizados. De fato, conforme afirmado por (GEISSLER et al., 2020), infraestruturas com comutadores OpenFlow heterogêneos são uma realidade, e lidar com esta diversidade é essencial para otimizar o uso dos recursos, diminuir custos e viabilizar a próxima geração das arquiteturas de rede. É exatamente neste contexto que este trabalho foi desenvolvido.

4 Cenário heterogêneo para VNFs

Virtualizar funções de rede em comutadores OpenFlow para tomada de decisão seletiva de pacote é uma abordagem atraente, pois a inspeção de cabeçalhos, o encaminhamento e o descarte de pacotes são ações inerentes aos comutadores OpenFlow. No entanto, como o número de regras necessárias para implementar um VNF pode aumentar consideravelmente com a carga da rede e a complexidade da função, um único comutador OpenFlow pode não ser capaz de lidar com todo o tráfego de forma eficiente. A solução imediata é combinar vários comutadores OpenFlow e dividir a carga de trabalho entre eles. Nesse sentido, e parcialmente inspirado pelo trabalho de (CHAVES; GARCIA; MADEIRA, 2017), a Figura 4.1 mostra a topologia para uma VNF genérica que adotamos neste trabalho.

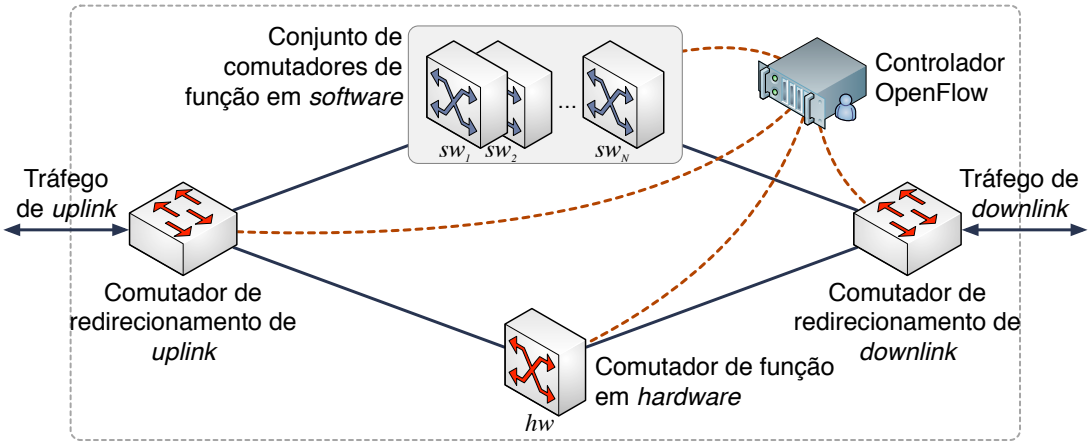


Figura 4.1: Comutadores heterogêneos implementando uma única VNF.

Nessa topologia genérica há um conjunto de *comutadores de função* em *software* (sw_i, com $i \in \{1, 2, ..., n\}$) e um único *comutador de função* em *hardware* (hw). As regras OpenFlow que estão distribuídas entre esses comutadores heterogêneos de função são aquelas que realmente executam as tarefas da VNF de maneira escalável. Por sua vez, os *comutadores de redirecionamento* de *uplink* (UL) e *downlink* (DL) atuam simultaneamente como pontos de entrada e saída do tráfego na VNF. Esses dois comutadores de redirecionamento também são responsáveis por efetivamente implementar o balanceamento da carga entre os comutadores de função. Por último, o controlador OpenFlow

local gerencia o tráfego dentro da VNF, instalando, atualizando e removendo regras de fluxo nos comutadores de função e redirecionamento de forma a implementar as políticas de controle definidas pelo operador da rede.

Sem perder a generalidade, é sugerido como um *firewall* sem estado poderia ser implementado com essa topologia. Nesse cenário, os comutadores de redirecionamento garantem a conexão da VNF com a rede local e a Internet. As regras de fluxo que estão distribuídas entre os comutadores de função implementam as tarefas de *firewall* como, por exemplo, identificar os endereços IP de origem e destino para encaminhar ou descartar adequadamente os pacotes. Além disso, com a ajuda imediata do controlador OpenFlow local dentro da VNF, poderíamos estender este *firewall* para uma versão com armazenamento de estado, salvando informações sobre as conexões ativas no *software* centralizado de controle.

Um diferencial importante do cenário adotado neste trabalho em relação ao cenário proposto por (CHAVES; GARCIA; MADEIRA, 2017) é a utilização de comutadores de função com recursos distintos, caracterizando uma abordagem heterogênea composta por um comutador de função em *hardware* e por um ou mais comutadores de função em *software*. Como discutido em (COSTA et al., 2017), comutadores OpenFlow implementados em *hardware* são otimizados para o encaminhamento de pacotes, suportam altas cargas de trabalho e introduzem baixo atraso nos fluxos de dados. Entretanto, sofrem de uma limitação importante: o tamanho reduzido das tabelas de fluxo. As tabelas pequenas são consequência do alto custo e elevado consumo energético das Memórias de Conteúdo Ternário Endereçável (TCAM, do Inglês *Ternary Content-Addressable Memory*), utilizadas para a implementação eficiente das regras OpenFlow que envolvem máscaras nos campos de busca. Por outro lado, implementações em *software* de comutadores OpenFlow (como o Open vSwitch), que podem ser instanciadas em servidores de prateleira, são facilmente configuráveis e possuem tabelas de fluxos consideravelmente maiores. Entretanto, o atraso no encaminhamento de pacotes num ambiente virtualizado pode ser até uma ordem de grandeza maior em relação ao atraso de um comutador em *hardware*, o que limita consideravelmente sua capacidade máxima de processamento.

A combinação entre comutadores de função heterogêneos é benéfica, pois o pro-

vedor da infraestrutura de virtualização pode manter um ou mais comutadores OpenFlow em *hardware* sempre disponíveis para oferecer continuidade no serviço da VNF, enquanto que novas instâncias temporárias de comutadores em *software* podem ser ativadas sob demanda para atender aos picos de tráfego na rede.

A seguir, são apresentados os dois mecanismos propostos neste trabalho para a gerência dinâmica desta VNF com recursos heterogêneos. Primeiro, na Seção 4.1, é considerado o uso do balanceamento de carga para distribuir os tráfegos entre os comutadores de função. Como os comutadores de redirecionamento da VNF estão sujeitos à grandes cargas de tráfego, então eles devem dispor de *hardware* de alto desempenho, com possíveis restrições no tamanho de suas tabelas de fluxo. Portanto, a estratégia de balanceamento de carga deve contar com um número moderado de entradas de fluxos nestes comutadores. Além disso, na Seção 4.2, é examinada a escalabilidade do cenário, onde o controlador OpenFlow monitora a carga da VNF para ajustar o número de comutadores de função em *software* ativos. Nesse caso, o mecanismo de balanceamento de carga deve estar ciente das mudanças na topologia da VNF, redistribuindo a carga de trabalho entre os comutadores de função quando necessário.

4.1 Mecanismo de balanceamento de carga

O mecanismo de balanceamento dinâmico de carga aqui proposto leva em consideração os recursos disponíveis nos comutadores da função virtual. Ele tem como objetivo aumentar o desempenho da VNF, explorando melhor o tamanho da tabela de fluxos dos comutadores de função em *software* e a alta capacidade de processamento dos comutadores de função em *hardware*. Em linhas gerais, a estratégia de funcionamento deste mecanismo é instalar novos tráfegos sempre nos comutadores em *software* (para explorar as grandes tabelas de fluxos) e, periodicamente, mover os tráfegos com maior vazão para o comutador em *hardware* (para explorar a alta capacidade de processamento). Este mecanismo é detalhado na sequência.

Sempre que um novo tráfego (identificado por um pacote que não corresponde a nenhuma regra existente na tabela de fluxo dos comutadores de função) chega na VNF, o protocolo OpenFlow envia uma mensagem com este pacote para o controlador. O

controlador então executa o Algoritmo 1 para a admissão do novo fluxo de dados. Durante este processo de admissão, o controlador verifica os indicadores de uso das tabelas de fluxo e da capacidade de processamento de todos os comutadores em *software*, procurando pelos valores críticos indicados aqui por α e β. Se α ou β exceder o limiar κ, o controlador bloqueia a solicitação, dado a falta de recursos na VNF. Caso contrário, o controlador instala uma cópia da nova regra de processamento da função virtual em cada um dos comutadores em *software* e aceita o tráfego. Neste trabalho adotamos $\kappa = 95\%$.

Algoritmo 1: Admissão de novos fluxos de dados

Entradas: Os metadados do novo fluxo de dados
O atual número n de comutadores em *software* ativos
Os indicadores de uso das tabelas de fluxos $T(sw_i), \forall i \in \{1, ..., n\}$
Os indicadores de uso de CPU $C(sw_i), \forall i \in \{1, ..., n\}$
Saída : Verdadeiro, caso o fluxo possa ser aceito, Falso caso contrário

```
1  início
       /* Verifica os recursos disponíveis no comutador em software */
2      α ← max( T(sw_i), ∀i ∈ {1, ..., n} )
3      β ← max( C(sw_i), ∀i ∈ {1, ..., n} )
4      se α ≥ κ ou β ≥ κ então
5          retorna Falso
6      fim
       /* Instala regras de função nos comutadores em software */
7      para i ← 1 até n faça
8          Instala as novas regras de função em sw_i
9      fim
10     retorna Verdadeiro
11 fim
```

A estratégia de instalar uma cópia da nova regra em cada um dos comutadores em *software* facilita o processo de redirecionamento dos tráfegos para o efetivo balanceamento da carga de trabalho entre eles. Neste contexto, podemos utilizar uma única regra OpenFlow de grupo nos comutadores de redirecionamento para encaminhar os pacotes aleatoriamente para os comutadores em *software*. Os grupos OpenFlow representam conjuntos de ações para semânticas complexas de encaminhamento. Nesse trabalho foram utilizadas as regras de grupo do tipo *select* para distribuir uniformemente os pacotes (e consequentemente, a carga) entre os comutadores de função em *software*. Neste caso, é importante frisar que todos os comutadores de função em *software* devem ter cópias das regras para processamento de todos os tráfegos ativos, de modo que nenhum tráfego seja prejudicado com perda de pacotes. Note que replicar as regras pelos comutadores

em *software* não é um problema, pois o tamanho da tabela de fluxos destes comutadores tende a ser consideravelmente grande.

Essa abordagem reativa do balanceador de carga (em que o primeiro pacote de cada fluxo é enviado para o controlador), permite a adoção da VNF de maneira transparente na rede, dispensando eventual sinalização entre o controlador e as aplicações cliente/servidor. Além disso, o controlador OpenFlow e os comutadores de função e redirecionamento são todos locais para cada VNF, então o atraso incorrido pelo envio de um único pacote ao controlador torna-se desprezível.

Comutadores em *software* geralmente possuem tabelas de fluxo grande o suficiente para acomodar todos os novos tráfegos na VNF, eliminando os bloqueios por falta de espaço. Apesar desta vantagem, esses comutadores apresentam um atraso consideravelmente maior na manipulação de pacotes (COSTA et al., 2021), o que resulta numa capacidade máxima de processamento reduzida, quando comparado aos comutadores em *hardware*. Como consequência de instalar todos os novos tráfegos nos comutadores em *software*, eles podem atingir rapidamente o limite máximo da sua capacidade de processamento, resultando em bloqueio de requisições e eventuais perdas de pacotes. Para lidar com este problema, o Algoritmo 2 complementa a política de balanceamento de carga dinâmico com uma realocação periódica de alguns fluxos de pacotes.

O controlador da VNF verifica a disponibilidade de recursos (ocupação da tabela de fluxos e capacidade de processamento) no comutador de função em *hardware* e também monitora as estatísticas dos fluxos de pacotes em todos os comutadores de função a cada 10 segundos. Posteriormente, o controlador calcula a vazão média para cada fluxo de pacotes atualmente instalado nos comutadores de função em *software*, classificando esses fluxos em ordem decrescente de vazão. Essas estatísticas servem de entrada para o Algoritmo 2 de realocação de fluxos. Enquanto houver recursos disponíveis no comutador de função em *hardware*, este algoritmo move as regras correspondentes aos fluxos de maior vazão dos comutadores em *software* para o comutador em *hardware*, de modo a ocupar todo o espaço disponível no comutador em *hardware*. Assim, os fluxos "elefantes", que exigem um maior processamento de pacotes, serão alocados no comutador de função em *hardware*, enquanto os inúmeros outros fluxos "ratos", que não exigem muito

Algoritmo 2: Realocação dos fluxos de dados

Entradas: As estatísticas de vazão de todos os fluxos de dados ativos
Os indicadores de uso das tabelas de fluxos $T(hw)$
Os indicadores de uso de CPU $C(hw)$

```
1  início
2      E ← cópia das regras de fluxos atualmente instaladas em sw_1
3      enquanto T(hw) < κ e C(hw) < κ faça
4          e ← fluxo em E com maior vazão
5          Move para o Hardware (e)
6          Remove e de E
7      fim
8  fim
9  procedimento Move para o Hardware (regra de fluxo e) faça
10     Instala uma cópia da regra de fluxo e em hw
11     Instala regras de fluxos com maior prioridade nos comutadores de
         redirecionamento para encaminhar o corresponde fluxo de dados para hw
12     para i ← 1 até n faça
13         Escalona a remoção da regra de fluxo e de sw_i
14     fim
15 fim
```

processamento individual, permanecerão nos comutadores de função em *software*.

A transferência dinâmica de cada fluxo é feita através da instalação no comutador de função em *hardware* de uma regra OpenFlow exatamente igual à existente nos comutadores de função em *software*. Na sequência, é necessário que regras específicas (e de maior prioridade do que a regra geral de grupo) sejam instaladas nos comutadores de redirecionamento para encaminhar apenas os pacotes deste fluxo para o comutador de função em *hardware*. Após a instalação da nova regra de função e da atualização dos comutadores de redirecionamento, as várias cópias antigas da regra de função podem ser finalmente removidas dos comutadores de função em *software* (esta etapa é opcional, pois é possível deixar que as regras expirem naturalmente com o temporizador de ociosidade). A ordem de execução dessas operações é importante, pois minimiza as perdas de pacotes durante a realocação dos fluxos.

Em suma, o balanceamento dinâmico de carga explora a maior capacidade de processamento do comutador em *hardware*, concentrando nele os fluxos de maior vazão mas sem extrapolar sua restrita tabela de fluxos. Como consequência, apenas os fluxos com menor vazão individual continuarão instalados no comutador em *software*, reduzindo sua demanda agregada por processamento e consequente perdas de pacotes.

4.2 Mecanismo de escalabilidade do plano de dados

O mecanismo de escalabilidade do plano de dados aqui proposto visa ajustar o número de comutadores de função em *software* que estão ativos em um determinado momento, de acordo com a demanda da rede. O tamanho da tabela de fluxo dos comutadores de função em *software* não é fator responsável por bloqueio de novos tráfegos. Porém, ainda é possível que o tráfego agregado nos comutadores de função em *software* atinja a capacidade máxima. Portanto, propomos o mecanismo de escalabilidade do plano de dados para aumentar e diminuir dinamicamente o número de comutadores de função em *software* ativos para atender à demanda da rede. O Algoritmo 3 detalha este mecanismo.

Algoritmo 3: Escalabilidade do plano de dados

Entradas: O atual número n de comutadores em *software* ativos
O número máximo de comutadores em *software* σ
Os indicadores de uso de CPU $C(sw_i)$, para todo $i \in \{1, \dots n\}$

início
  / Calcula a média de uso de CPU μ dos comutadores em software */*
  $\mu \leftarrow \frac{1}{n} \sum_{i=1}^{n} C(sw_i)$
  / Instancia um novo comutador em software */*
  se $\mu > \theta_i$ **e** $n < \sigma$ **então**
    $n \leftarrow n + 1$
    Instancia um novo comutador sw_n
    para cada regra de fluxo e atualmente instalada em sw_1 **faça**
      Instala uma cópia da regra de fluxo e em sw_n
    fim
    Atualiza a regra de grupo nos comutadores de redirecionamento
  fim
  / Desativa um ou mais comutadores em software */*
  senão se $n > 1$ **então**
    $\mu' \leftarrow \frac{1}{n-1} \sum_{i=1}^{n-1} C(sw_i)$
    enquanto $\mu' < \theta_d$ **e** $n > 1$ **faça**
      Desativa o comutador em *software* sw_n
      $n \leftarrow n - 1$
      Atualiza a regra de grupo nos comutadores de redirecionamento
      $\mu' \leftarrow \frac{1}{n-1} \sum_{i=1}^{n-1} C(sw_i)$
    fim
  fim
fim

O controlador da VNF monitora periodicamente os n comutadores de função em *software* ativos para calcular o indicador de uso de CPU individual $C(sw_i)$. Esta informação permite ao Algoritmo 3 calcular o uso médio de CPU μ para todos os co-

mutadores em *software* ativos. Se μ exceder o limiar de incremento θ_i e não estivermos com o número máximo de comutadores em *software* suportados pela infraestrutura σ ativos, o controlador instancia um novo comutador de função em *software*. Neste caso, ele copia todas as regras atualmente instaladas em qualquer outro comutador em *software* para este novo (arbitrariamente, foi optado por copiar as regras de sw_1). Finalmente, ele atualiza a regra de grupo nos comutadores de redirecionamento com o novo número n de comutadores em *software* ativos.

Este algoritmo também verifica se é possível diminuir o número atual n de comutadores de função em *software* ativos. Para isto, ele calcula o uso médio da CPU μ' para uma possível situação com $n-1$ comutadores. Se μ' for inferior ao limiar de decremento θ_d, o controlador desativa um comutador em *software*. Ele repete esse processo até que μ' fique acima de θ_d ou até que apenas um comutador em *software* permaneça ativo. Se ocorrer qualquer ajuste em n, o controlador atualiza as regras de grupo nos comutadores de redirecionamento.

5 Balanceamento de carga com plano de dados não escalável

Neste capítulo foi avaliado como o balanceamento de carga adequado entre comutadores heterogêneos pode otimizar o uso de recursos e melhorar o desempenho geral da VNF, aumentando a vazão agregada e diminuindo a taxa de bloqueio.

5.1 Metodologia de avaliação

É considerada aqui uma pequena instância da nossa topologia genérica com exatamente um comutador de função em *hardware* (*hw*), um comutador de função em *software* (*sw*) e sem o mecanismo de escalabilidade do plano de dados. A Figura 5.1 ilustra esta topologia.

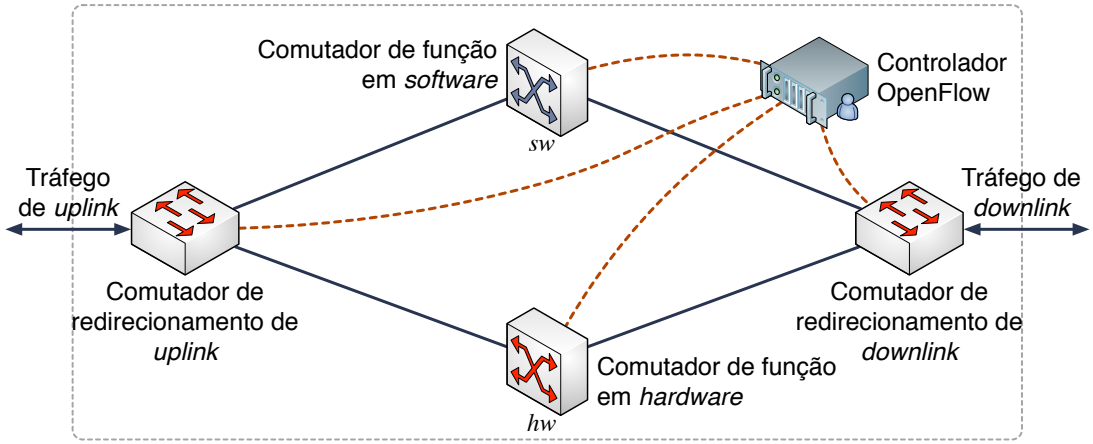


Figura 5.1: Topologia VNF com um comutadores em *hardware* e em *software*.

Foi utilizado um pequeno ambiente de testes com sete computadores dedicados para implementar esta topologia. Para emular os comutadores internos da VNF foram utilizadas quatro máquinas independentes, cada uma executando uma instância do comutador virtual de código aberto *Open vSwitch*. Um quinto computador atua como controlador OpenFlow executando o *software* *POX*[4]. Por fim, um par de computadores foi utilizado como cliente e servidor, gerando tráfego na rede com o auxílio da ferramenta

[4] ⟨https://github.com/noxrepo/⟩

Iperf[5]. Nesta topologia, todos os enlaces de comunicação entre as máquinas são *Gigabit Ethernet*.

Neste ambiente de testes, a heterogeneidade entre os comutadores em *hardware* (redirecionamento e função) e o comutador de função em *software* deve-se à diferença nas configurações dos equipamentos utilizados. Para os comutadores em *hardware* foram adotadas três máquinas com processador Intel Core i3-540, 2 GB de memória e sistema operacional Ubuntu 18.04 LTS. Para o comutador de função em *software* foi adotada uma máquina antiga, com processador Intel Celeron D, 512 MB de memória e sistema operacional Lubuntu 16.04 LTS. Essa discrepância nas configurações resultou em uma capacidade máxima de processamento 2,5 vezes maior no comutador em *hardware* quando comparado ao comutador em *software*. Note que, ao se utilizar comutadores reais, essa diferença pode ser ainda maior. Assim, os resultados apresentados na sequência podem ser subestimados, servindo como limite inferior de ganho.

Para refletir a diferença no tamanho das tabelas de fluxo, o comutador de função em *hardware* foi ajustado para suportar no máximo 200 regras simultâneas. Este valor foi definido de acordo com as dimensões do experimento, de modo a analisar o impacto do tamanho da tabela no bloqueio dos tráfegos. Para o comutador de função em *software*, a tabela de fluxos foi mantida com seu tamanho padrão.

Neste ambiente de testes não é possível incluir mais clientes e servidores, visto que todas as interfaces de rede nos comutadores de redirecionamento já estão em uso. Tal restrição implica em uma vazão teórica máxima de 1 Gbps em cada direção. Entretanto, este não é fator limitante para os experimentos, já que a capacidade reduzida de processamento do comutador em *software* ($\approx$ 400 Mbps) e o tamanho reduzido da tabela de fluxos do comutador em *hardware* (200 regras) são, de fato, os gargalos deste cenário.

O *software Iperf* foi configurado com os parâmetros da Tabela 5.1 para gerar fluxos de dados UDP unidirecionais com duração uniforme e vazão exponencial. A taxa de chegada de novos fluxos segue um processo de *Poisson*, onde três valores distintos de λ foram adotados para definir intensidades de carga *baixa*, *média* e *alta* nos experimentos. Por fim, os atrasos de pacotes nos comutadores de função foram medidos utilizando uma

[5]⟨https://iperf.fr⟩

versão modificada do software *UDPPing*[6], com sondas em paralelo ao tráfego gerado.

Tabela 5.1: Parâmetros para geração dos tráfegos no ambiente experimental.

Parâmetro	**Distribuição**	**Carga**		
		Baixa	**Média**	**Alta**
Taxa de chegada de fluxos	*Poisson*	$\lambda = 3,\overline{3}$	$\lambda = 5$	$\lambda = 10$
Duração do fluxo	Uniforme	min = 5 s e max = 100 s		
Vazão do fluxo	Exponencial	média = 1024 Kbps		

Este ambiente de teste foi utilizado para avaliar o desempenho do mecanismo de balanceamento de carga considerando nossa política dinâmica apresentada na Seção 4.1 em comparação com uma política estática. Na *política estática*, os fluxos são balanceados de forma simples entre os comutadores de função em *hardware* e em *software*, levando em consideração apenas o número da porta UDP do tráfego. Em outras palavras, o controlador envia fluxos de pacotes originados em porta UDP ímpar para o comutador de função em *software* e fluxos de pacotes originados em porta UDP par para o comutador de função em *hardware*. Esta política estática de balanceamento de carga não considera os recursos computacionais disponíveis nos comutadores de função. Ele apenas distribui fluxos uniformemente entre os comutadores, balanceando a carga a longo prazo.

Note que a utilização do número da porta UDP para a tomada de decisão não é a ideal. De fato, na prática, tal decisão não possui escalabilidade, demandando a instalação de inúmeras regras nos comutadores de redirecionamento. Porém, essa política serve como caso base de comparação, e foi adotada neste trabalho apenas por conta da limitação de um único par de cliente/servidor no cenário de experimentação. Em cenários com mais clientes e servidores, é possível realizar as decisões baseadas no endereço IP do cliente, utilizando regras OpenFlow com máscaras para identificar valores pares/ímpares, o que reduziria consideravelmente o número de regras nos comutadores de redirecionamento.

Para cada combinação entre as *políticas de balanceamento* × *carga de trabalho*, realizamos 15 execuções de testes. Os resultados apresentados na Seção 5.2 a seguir são a média das 15 execuções, considerando as análises no período estável do experimento, com um nível de confiança de 95%.

[6] ⟨https://github.com/wangyu-/UDPping⟩

5.2 Resultados e Discussões

Inicialmente, foi avaliado o percentual de bloqueios de fluxos, considerando as diferentes cargas de trabalho. Os bloqueios acontecem quando o comutador de função escolhido pela política de balanceamento não tem recursos para acomodar o novo fluxo. No cenário avaliado, isso acontece apenas quando a política estática está em uso e não há mais espaço disponível nas tabelas de fluxo do comutador de função em *hardware*. De fato, a política dinâmica não gera bloqueios, visto que novos fluxos são sempre instalados no comutador de função em *software*, que possui tabela de fluxos com tamanho demasiadamente grande.

A Figura 5.2 mostra o percentual de bloqueios ao se utilizar a política de balanceamento estática, em uma rede com baixa, média e alta carga de tráfego. Em baixa carga, o percentual de bloqueio do comutador em *hardware* é negligenciável. Porém, em uma rede com carga média de trabalho, pouco mais de 3% dos fluxos são bloqueados. Esse número pode superar 16% quando a rede está em alta carga de trabalho. Esse valor mostra a importância da política dinâmica e de políticas que exploram o conhecimento dos recursos computacionais disponíveis nos elementos de rede.

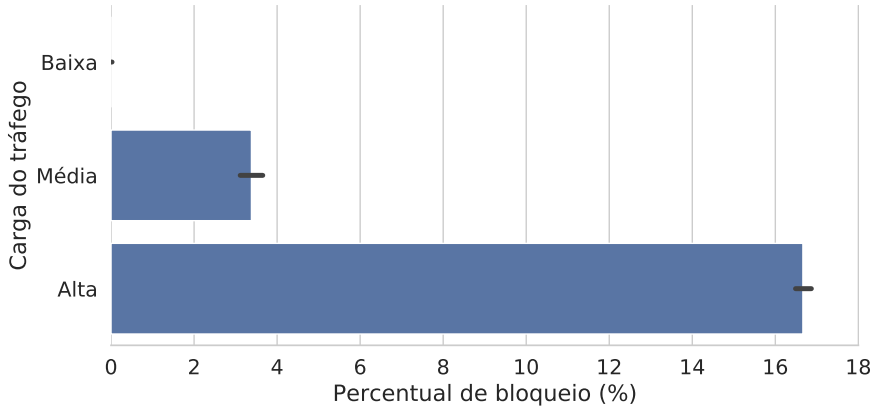


Figura 5.2: Percentual de bloqueio no comutador de função em *hardware*.

A vazão média da rede depende da demanda dos fluxos e, também, da política adotada pela VNF no balanceamento da carga. A Figura 5.3 apresenta a vazão média da rede para as diferentes configurações dos experimentos realizados. Para as cargas baixa e média, não há diferença estatística significativa entre as políticas. Claramente, há recursos computacionais disponíveis nesses cenários, onde a política de balanceamento tem pouca influência. A necessidade do balanceamento de carga e a importância da política

apropriada fica mais evidente com a alta carga de trabalho. A diferença média de vazão entre a política estática e a dinâmica, nesse cenário, é aproximadamente 27%.

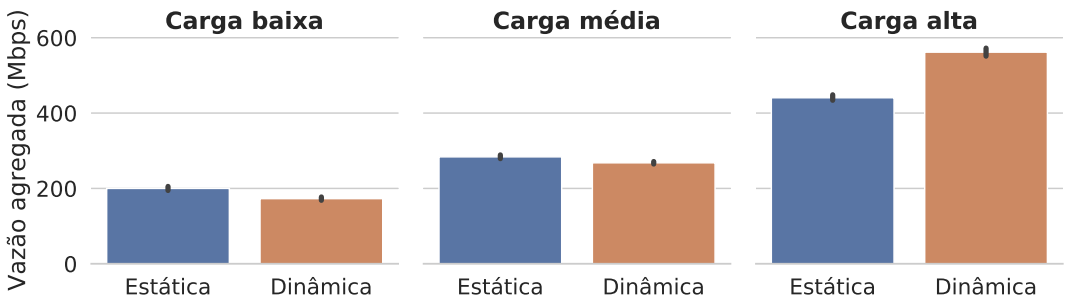


Figura 5.3: Vazão média da rede por política no intervalo estável.

A Figura 5.4 detalha a vazão da rede e mostra a carga atribuída a cada comutador de função. Para as cargas baixa e média, quando os comutadores de função têm recursos disponíveis em excesso, a política estática divide a demanda da rede entre os dois comutadores de modo estatisticamente igual. Cada comutador fica com uma vazão próxima a 100 Mbps. Nesses dois cenários, a política dinâmica apresentou maior vazão no comutador de função em *hardware*. Ou seja, quando há recursos disponíveis, a política dinâmica concentra a maioria dos fluxos no comutador em *hardware*, privilegiando o seu melhor desempenho no processamento de pacotes. O comutador em *hardware*, para as cargas baixa e média, ficou responsável por ≈ 140 Mbps e ≈ 200 Mbps de vazão, respectivamente.

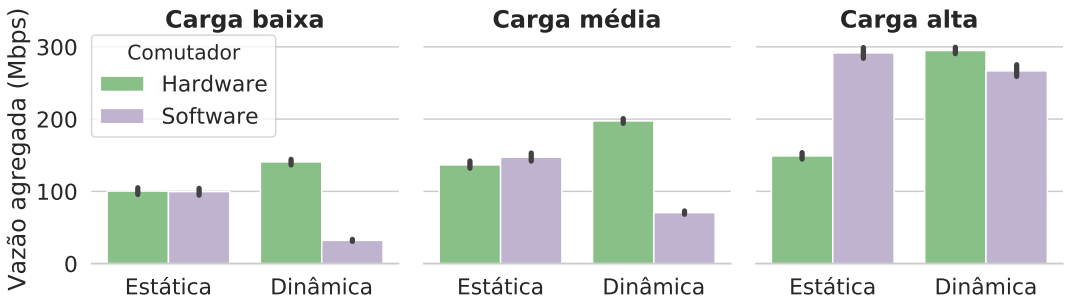


Figura 5.4: Vazão média por comutador no intervalo estável.

Por outro lado, quando os recursos de rede estão escassos (carga alta), a política estática continua atribuindo fluxos igualmente aos comutadores de função. Como o comutador em *hardware* tem limitação no tamanho da tabela, ele começa a negar serviço e eleva o percentual de bloqueio (Figura 5.2). Com isso, a vazão no comutador em *hardware* fica em ≈ 150 Mbps, o que é bem menor do que a vazão do comutador em *software*, que excede

290 Mbps. A política dinâmica evita os bloqueios e a disputa de recursos no comutador em *hardware*. Ela encaminha apenas os fluxos com maior vazão para este comutador, que tem maior poder de processamento, respeitando assim o seu limite no tamanho da tabela. Dessa forma, mesmo com uma quantidade menor de fluxos, o comutador em *hardware* alcança uma vazão de rede tão alta quanto o comutador em *software*.

A Figura 5.5 corrobora com a discussão anterior ao mostrar o número médio de regras instaladas nos comutadores de função. A distribuição das regras, em cargas baixa e média, é igualitária entre os comutadores pela política estática (assim como acontece com a vazão, já apresentada na Figura 5.4). Por outro lado, nessas cargas (média e baixa), a política dinâmica já explora prioritariamente o comutador em *hardware*. Mesmo com o número total de regras limitado em 200 neste comutador, a política dinâmica sempre tenta redirecionar os maiores fluxos para ele. Em carga alta, a limitação na tabela de fluxos do comutador em *hardware* torna-se crítica para ambas as políticas. Nesse caso, porém, os efeitos na vazão são mais evidentes, como demonstrado anteriormente.

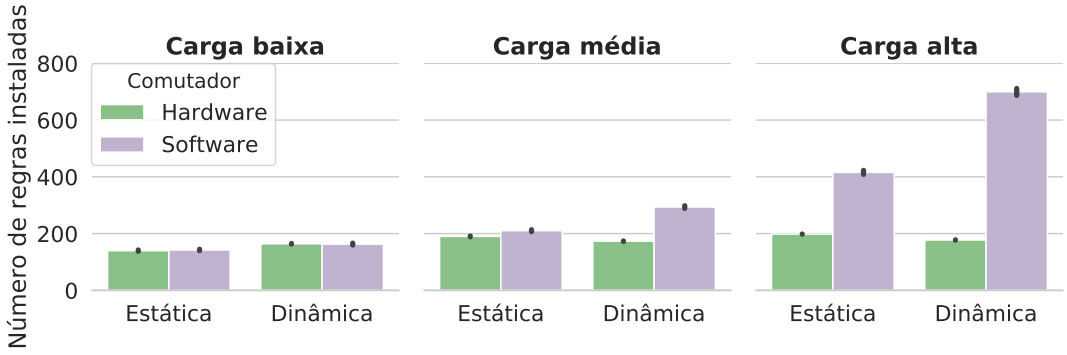


Figura 5.5: Número médio de regras instaladas por comutador no intervalo estável.

Notadamente, a política dinâmica apresenta uma vantagem com relação ao melhor aproveitamento dos recursos da VNF, resultando no aumento de sua vazão. Analisamos também um outro indicador de qualidade: a latência fim-a-fim dos pacotes. A Figura 5.6 apresenta a função de distribuição cumulativa dos atrasos para os pacotes de sondagem em ambos os comutadores, apenas no cenário de maior demanda (i.e., carga alta). Por essa figura, a política estática apresenta uma leve vantagem sobre a política dinâmica, apenas para os fluxos direcionados ao comutador de função em *hardware*. De fato, para 50% dos pacotes de sondagem, a política estática apresenta até 0,6 ms de atraso neste comutador. Já na política dinâmica, para esse mesmo percentual, o atraso é de até 0,75 ms.

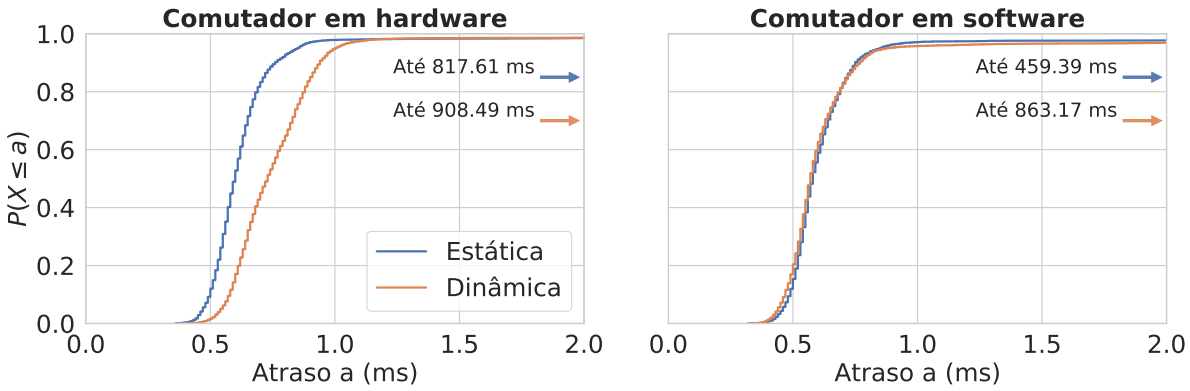


Figura 5.6: Atraso fim-a-fim no intervalo estável com carga alta (pior caso).

A diferença entre esses atrasos é consequência da natureza das políticas. Pela política estática, o comutador de função em *hardware* bloqueia alguns fluxos. Logo, a vazão agregada é menor neste comutador, as filas ficam pequenas e o atraso diminui. Por outro lado, pela política dinâmica, apenas os maiores tráfegos são atribuídos ao comutador em *hardware*. Por conta disso, a vazão agregada aumenta e as filas ficam maiores, impactando no atraso. Além disso, existe um atraso decorrente do processamento de pacotes pelo *Open vSwitch* no comutador de função em *hardware* que não existiria em um comutador físico. Portanto, os atrasos indicados para este comutador na Figura 5.6 representam o pior caso. É esperado que, em uma topologia com um comutador em *hardware* dedicado, os atrasos sejam menores do que os atrasos do comutador em *software*.

Tabela 5.2: Percentual de pacotes fora de ordem por fluxo de dados.

	Política estática		
	Carga baixa	**Carga média**	**Carga alta**
Mínimo	0	0	0
Mediana	0	0	0
Média	0	4.88×10^{-8}	4.62×10^{-7}
Máximo	0	3.95×10^{-4}	1.01×10^{-3}

	Política dinâmica		
	Carga baixa	**Carga média**	**Carga alta**
Mínimo	0	0	0
Mediana	0	0	0
Média	0	0	1.28×10^{-5}
Máximo	0	0	2.88×10^{-2}

Por fim, foi observado um número muito pequeno de pacotes sendo recebidos fora de ordem nas duas políticas, como apresentado na Tabela 5.2. Isso demonstra que a qualidade da rede, neste quesito, é semelhante para ambas as políticas. Novamente, a pequena diferença existe por conta da natureza das políticas. Mesmo com a remoção das regras antigas sendo feita após a instalação de novas regras para evitar perda de pacotes na política dinâmica, pacotes antigos já enviados pelo comutador em *software* podem chegar depois dos pacotes novos enviados pelo comutador em *hardware*, devido a diferença no tempo de processamento dos comutadores. A consequência natural é um pequeno aumento no número de pacotes recebidos fora de ordem. Porém, mesmo no cenário de carga alta, as perdas de pacote são desprezíveis para todos os fluxos em ambas as políticas.

6 Balanceamento de carga com plano de dados escalável

Neste capítulo é avaliado como o mecanismo de escalabilidade do plano de dados pode ajustar o número de comutadores em *software* de acordo com a demanda da rede, otimizando o uso de recursos enquanto mantém o desempenho da VNF.

6.1 Metodologia de avaliação

Foi considerada aqui uma instância da nossa topologia VNF escalável com até n comutadores de função em *software* (sw_1 a sw_n) e um único comutador de função de hardware (hw). Essa topologia foi apresentada na Figura 4.1. Para avaliar o mecanismo de escalabilidade do plano de dados junto com o mecanismo de balanceamento de carga dinâmico, foi implementada a topologia escolhida no simulador de rede ns-3[7], utilizando o módulo OFSwitch13 para suporte ao OpenFlow 1.3 (CHAVES; GARCIA; MADEIRA, 2016).

A Tabela 6.1 descreve os parâmetros que foram usados para configurar os comutadores OpenFlow no simulador. Com relação aos comutadores de função, o *hardware* apresenta quase três vezes mais capacidade de processamento da CPU do que o *software*. Por outro lado, os comutadores em *software* têm tabelas de fluxo oito vezes maiores em comparação com os comutadores em *hardware*. Esses valores foram escolhidos em proporção ao tamanho do cenário de simulação que será descrito na sequência. Comutadores de redirecionamento com alta capacidade de CPU foram adotados, pois pretendemos criar o gargalo apenas nos comutadores de função. Da mesma forma, o controlador OpenFlow e os nós cliente/servidor também apresentam recursos suficientemente grandes em nossas simulações. Os links de conexão entre todos os dispositivos da rede é de 10 Gbps, evitando congestionamentos que poderiam ocultar o desempenho real da VNF.

Os pares cliente/servidor utilizam uma versão modificada de uma aplicação UDP,

[7] ⟨https://www.nsnam.org⟩

Tabela 6.1: Parâmetros para os comutadores OpenFlow no ambiente simulado.

Parâmetros	**Comutador de redirecionamento**	**Comutador de função**	
		hardware	***software***
Tabela de fluxos	65535	1024	8192
Processamento da CPU	100 Gbps	2 Gbps	750 Mbps
Atraso TCAM	20 us	20 us	100 us

que envia pacotes em uma única direção com taxa constante. Eles foram configurados de forma balanceada, sendo que existem clientes e servidores em ambos os lados da VNF, criando tráfego em ambas as direções. A Tabela 6.2 descreve os parâmetros do tráfego simulado. Cada cliente inicia novos tráfegos seguindo um processo de *Poisson* com $\lambda = 0,5$ aplicações por segundo. Assim, ajustamos o número total de pares cliente/servidor para gerar diferentes cargas de trabalho. Precisamente, foi utilizado 50, 100 e 150 pares para cargas de tráfego baixa, média e alta, respectivamente.

Tabela 6.2: Parâmetros para a geração do tráfego no ambiente simulado.

Parâmetros	**Distribuição**	**Atributos**
Taxa de chegada de fluxos	*Poisson*	$\lambda = 0.5$
Duração do fluxo	Uniforme	min = 5 s e max = 100 s
Vazão do fluxo	Exponencial	média = 1024 Kbps

O Algoritmo 3 de escalabilidade do plano de dados possui alguns atributos configuráveis, sendo eles o número inicial de comutadores de função em *software* ativos n e o número máximo de comutadores de função em *software* suportados σ, que variam dependendo da simulação executada. Além destes, há também os limiares de incremento e decremento, ajustados para $\theta_i = 95\%$ e $\theta_d = 80\%$, respectivamente. Foi adicionado um atraso de dez segundos para a instanciação de um novo comutador de função em *software*, simulando o tempo de inicialização de uma nova máquina virtual. O controlador executa o Algoritmo 2 para realocar os fluxos entre os comutadores de função a cada dez segundos e o Algoritmo 3 para a escalabilidade do plano de dados a cada quinze segundos.

Foram realizadas 10 simulações independentes, e os resultados apresentados na Seção 6.2 a seguir são a média das execuções, considerando as análises no período estável do experimento, com um nível de confiança de 99%.

6.2 Resultados e Discussões

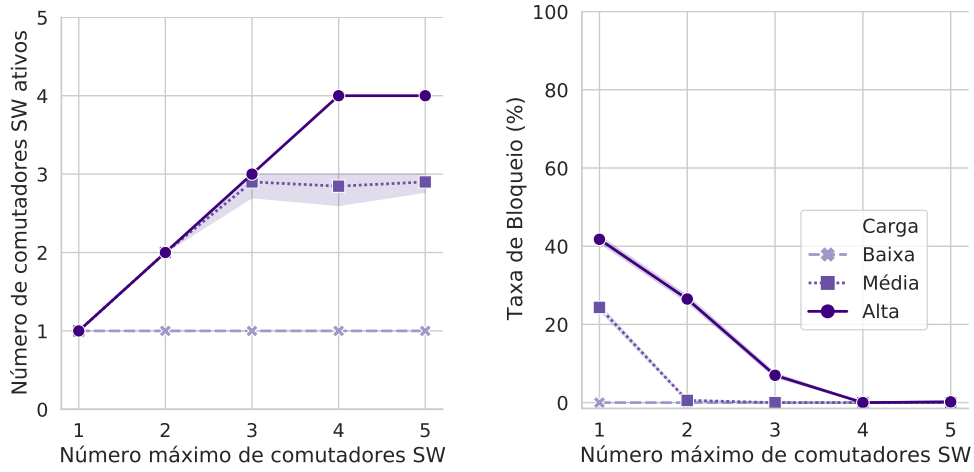
As simulações foram organizadas em dois conjuntos. O primeiro para verificar o mecanismo de escalabilidade do plano de dados, variando o número máximo de comutadores de função em *software*. O segundo considera os dois mecanismos operando em conjunto, porém com diferentes configurações no plano de dados heterogêneo da VNF.

6.2.1 Mecanismo de escalabilidade do plano de dados

Esta primeira análise avalia o mecanismo de escalabilidade do plano de dados. Neste cenário, o Algoritmo 2 foi testado com diferentes valores para o número máximo de comutadores de função em *software*, sendo $\sigma \in \{1, 2, 3, 4, 5\}$. É importante notar que o comutador de função em *hardware* sempre está ativo nessas simulações. Para cada configuração, avaliamos a função virtual em carga de trabalho baixa, média e alta.

A Figura 6.1 apresenta o número médio de comutadores de função em *software* ativos e a taxa de bloqueio média para o diferentes número máximo de comutadores em *software*. São apresentados os resultados para carga de tráfego baixa, média e alta em ambas as figuras. De acordo com a Figura 6.1a, a VNF precisa de apenas um único comutador em *software*, em companhia do comutador de função em *hardware*, para lidar com a carga baixa de trabalho. Como esperado, sob baixa demanda de tráfego, a taxa de bloqueio é desprezível, como mostrado na Figura 6.1b. A função virtual necessita de três e quatro comutadores em *software* para suportar toda a demanda da rede em cargas média e alta de trabalho. Em outras palavras, aumentar o número máximo de comutadores acima desses valores pode não apresentar efeitos práticos, exceto pelo desperdício de recursos de virtualização. Podemos observar uma alta taxa de bloqueio sob as cargas média e alta quando o número máximo de comutadores de função em *software* é muito baixo. No entanto, essa taxa de bloqueio diminui rapidamente à medida que o número máximo de comutadores em *software* aumenta, o que evidencia a eficácia da escalabilidade do cenário.

A Figura 6.2 apresenta os indicadores de uso médio das tabelas de fluxo e de CPU dos comutadores de função em *hardware* e em *software* durante o intervalo estável das simulações. De acordo com a Figura 6.2a, o uso da tabela de fluxos do comutador em *hardware* é sempre próximo de 95%, não importando a carga de trabalho e nem o número

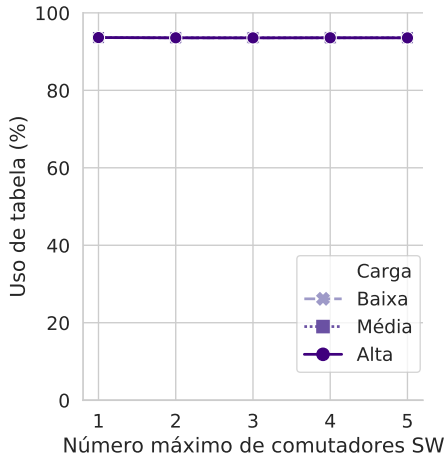


(a) Números de comutadores ativos. (b) Percentual de bloqueio.

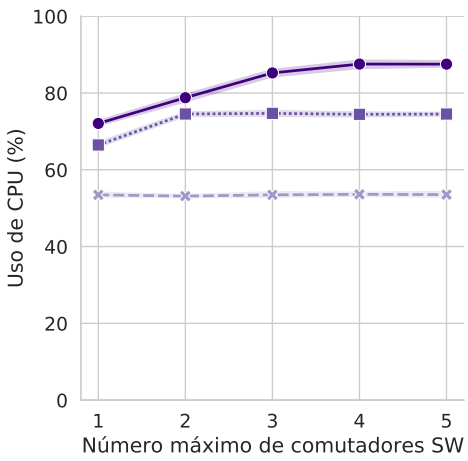
Figura 6.1: Números médio de comutadores de função e percentual de bloqueio.

máximo de comutadores em *software*. Este comportamento acontece pois o Algoritmo 2 para realocação dos fluxos move periodicamente os fluxos "elefante" para o comutador em *hardware*, mantendo cheia a sua tabela de fluxos até o limiar de bloqueio κ. O uso de CPU no comutador em *hardware* aumenta com a carga de trabalho, como mostrado na Figura 6.2b. Além disso, esse indicador permanece estável quando o mecanismo de escalabilidade do plano de dados atinge o número máximo de comutadores ativos. Conforme mostrado na Figura 6.2c, mesmo em configurações com alta carga de tráfego, o indicador de uso médio das tabelas de fluxo não ultrapassa 70%. Esse valor é esperado, devido as grandes tabelas de fluxo nesses comutadores. Por outro lado, o uso de CPU para os comutadores em *software* seguem exatamente a direção oposta. Como mostrado na Figura 6.2d, poucos comutadores em *software* ativos aumentam a carga da CPU até a capacidade máxima de processamento, agravando a taxa de bloqueio. Conforme o número de comutadores ativos aumenta, a carga de trabalho é melhor distribuída entre eles.

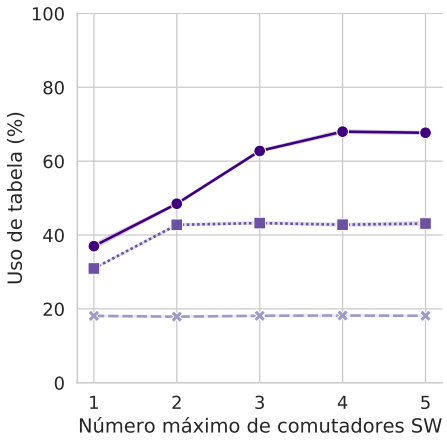
A Figura 6.3 apresenta a taxa média de pacotes perdidos e a vazão agregada média da VNF. Como mostrado na Figura 6.3a, um pequeno valor de σ implica em uma alta taxa de perda de pacotes, como esperado. Em particular, para uma rede sobrecarregada, a VNF pode sofrer quase 20% de perda de pacotes. Essa taxa de perda diminui rapidamente quando o número máximo de comutadores em *software* aumenta. Por exemplo, para um máximo de dois comutadores, a taxa de perda geral torna-se insignificante em todos os cenários de carga. Esses resultados podem ser confirmados comparando-os com



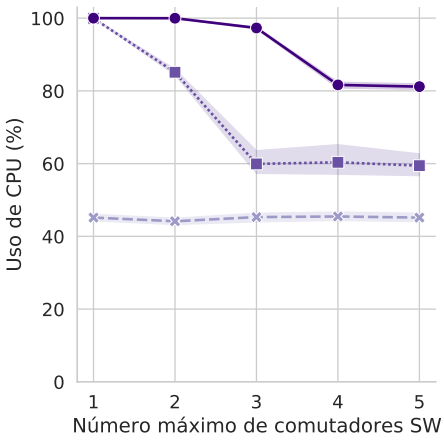
(a) Tabela do comutador em *hardware*.



(b) CPU do comutador em *hardware*.

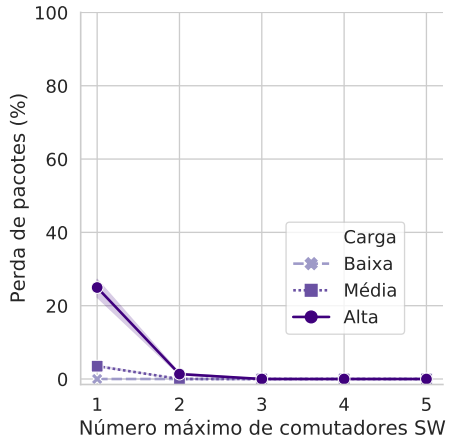


(c) Tabela dos comutadores em *software*.

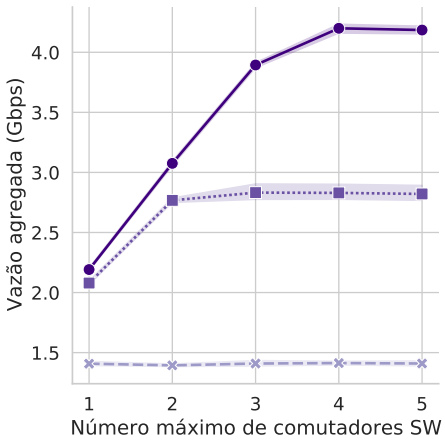


(d) CPU dos comutadores em *software*.

Figura 6.2: Uso de tabela e de CPU dos comutadores em *hardware* e em *software*.



(a) Perda de pacotes.



(b) Vazão agregada média.

Figura 6.3: Perda de pacotes e vazão agregada média.

a Figura 6.2d. Quando os comutadores em *software* estão funcionando em suas capacidades máximas, existem perdas significativas de pacotes. Finalmente, como mostrado na

Figura 6.3b, a VNF atinge sua vazão agregada máxima ao operar com vários comutadores em *software*. Nessas simulações, o pico foi de aproximadamente 4,2 Gbps com quatro e cinco comutadores em *software* com alta carga de trabalho na rede.

6.2.2 Mecanismos propostos sob diferentes configurações

Esta segunda análise de simulações avalia o funcionamento conjunto dos mecanismos propostos considerando as seguintes configurações da VNF:

- Sem comutador em *hardware* (HW = 0) e com exatamente 5 comutadores em *software* ativos ao longo da simulação (SW = 5);
- Sem comutador em *hardware* (HW = 0) e com escalabilidade do plano de dados, variando os comutadores em *software* de 1 a 5 ao longo da simulação (SW = 1–5);
- Com 1 comutador em *hardware* (HW = 1) e com exatamente 5 comutadores em *software* ativos ao longo da simulação (SW = 5);
- Com 1 comutador em *hardware* (HW = 1) e com escalabilidade do plano de dados, variando os comutadores em *software* de 1 a 5 ao longo da simulação (SW = 1–5);

A Figura 6.4 mostra o número médio de comutadores de função em *software* ativos que são necessários para atender toda a demanda da VNF durante o intervalo estável da simulação. Quando o mecanismo de escalabilidade do plano de dados está desabilitado (barras em cores claras, com SW = 5), o número de comutadores em *software* ativos permanece fixo em cinco, como esperado. Porém, quando o mecanismo de escalabilidade do plano de dados está habilitado (barras em cores escuras, com SW = 1–5), podemos observar uma economia de recursos em praticamente todos os casos. A única exceção é para a alta carga de trabalho sem o uso do comutador em *hardware*, onde a VNF continua a usar os cinco comutadores em *software* para atender a excessiva demanda de tráfego. Outro comportamento interessante é que um único comutador em *hardware* pode substituir até dois comutadores em *software* (considerando as barras escuras no cenário de baixa carga). Isso só é possível combinando os dois mecanismos propostos neste trabalho. Finalmente, ao habilitar o mecanismo de escalabilidade do plano de

dados com um comutador em *hardware* (barra verde escura), nenhum dos níveis de carga de trabalho exige mais do que quatro comutadores em *software*, corroborando nossos resultados anteriores, apresentados na Figura 6.1a.

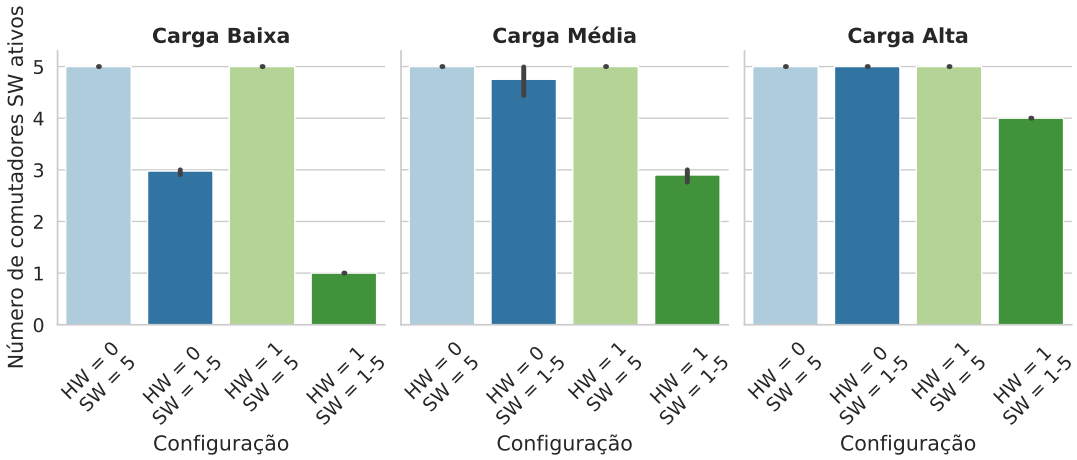


Figura 6.4: Números de comutadores em *software* ativos nas diferentes configurações.

Em seguida, a Figura 6.5 apresenta a vazão agregada média para as configurações distintas da VNF. Para cargas baixa e média, os resultados são estatisticamente iguais. Em outras palavras, sob essas cargas de trabalho, a VNF consegue atender toda a demanda da rede utilizando qualquer configuração. Portanto, ao utilizar cinco comutadores de função em *software* ao longo da simulação, estamos simplesmente desperdiçando recursos. No entanto, apenas as configurações que utilizam um comutador em *hardware* (barras verdes) podem melhorar a vazão agregada em alta carga de trabalho. Nesse caso, ambas as configurações (com ou sem escalabilidade do plano de dados) apresentam vazão estatisticamente igual. No entanto, ao analisar este resultado em conjunto com a Fi-

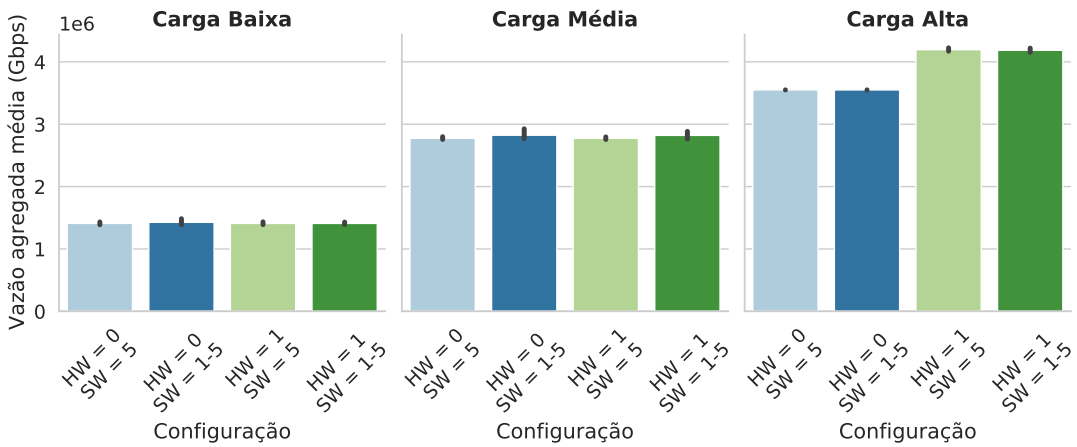
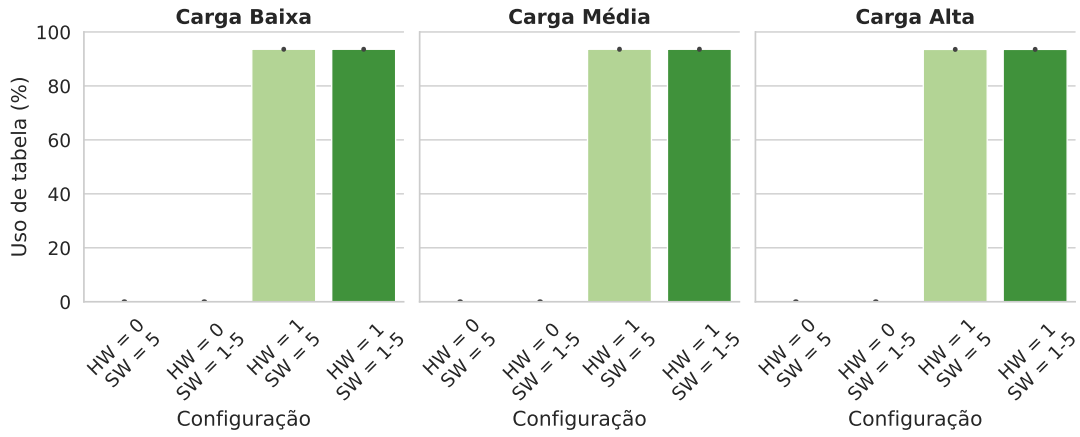


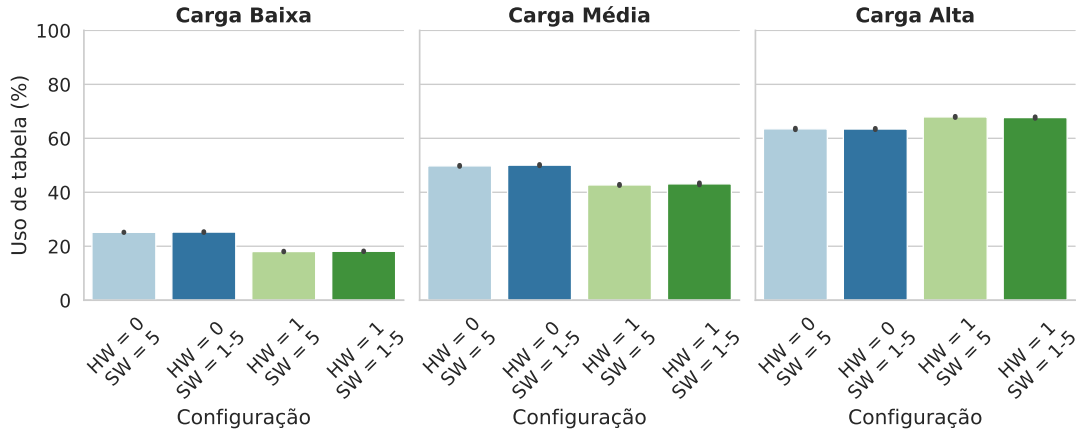
Figura 6.5: Vazão agregada média da VNF nas diferentes configurações.

gura 6.4, confirmamos que o mecanismo de escalabilidade do plano de dados (barra verde escura) atinge o mesmo desempenho e com um comutador a menos.

A Figura 6.6 exibe os indicadores de uso da tabela de fluxos para os comutadores de função em *hardware* e em *software*. Como mostrado na Figura 6.6a, quando o comutador em *hardware* está ativo (barras verdes), o mecanismo de balanceamento dinâmico mantém sua tabela de fluxo sempre cheia (em 95%) para explorar sua capacidade de processamento. Consequentemente, como a Figura 6.6b mostra, isso reduz o uso das tabelas de fluxos em cerca de 10% nos comutadores em *software*. A exceção é para a alta carga de trabalho. Nesse caso, os comutadores em *software* sobrecarregados nas configurações sem comutador em *hardware* (barras azuis) aumentam a taxa de bloqueio, ocultando essa diferença no indicador de uso das tabelas de fluxos.



(a) Uso médio de tabela do comutador de função em *hardware*.

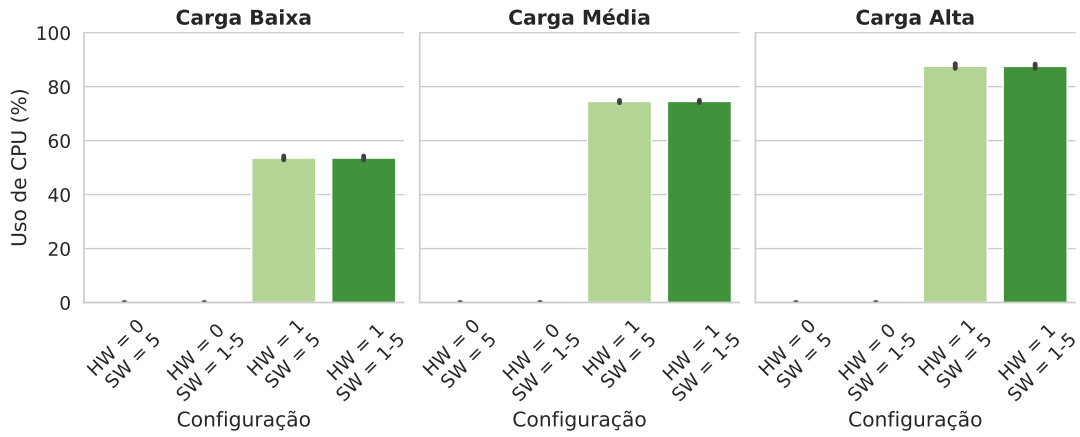


(b) Uso médio de tabela dos comutadores de função em *software*.

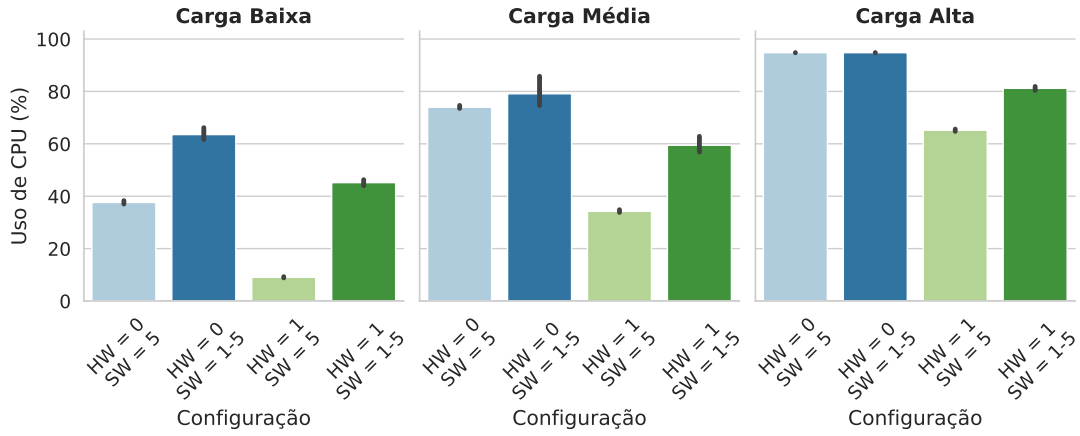
Figura 6.6: Uso médio de tabela dos comutadores de função nas diferentes configurações.

Por fim, a Figura 6.7 apresenta o uso da CPU para os comutadores em *hardware* e *software*. A carga de CPU em todos os comutadores de função aumenta de acordo com a

carga de trabalho da rede. Claramente, quanto mais tráfego na rede, mais fluxos "elefante" irão se beneficiar do alto desempenho do comutador em *hardware*, como mostrado na Figura 6.7a. No entanto, com a capacidade de processamento reduzida dos comutadores em *software*, o crescimento da sua carga de CPU se torna mais evidente na Figura 6.7b. Quando a VNF apresenta um comutador em *hardware* (barras verdes), o indicador de uso médio de CPU para os comutadores em *software* é consideravelmente menor do que as configurações sem comutador em *hardware* (barras azuis). Além disso, o mecanismo de escalabilidade do plano de dados (barras em cores escuras) aumenta o uso da CPU dos comutadores em *software* quando comparados com o número fixo de comutadores em *software* (barras em cores claras). Isso significa que o mecanismo obtém um menor desperdício dos recursos, comprovando a maior eficiência.



(a) Uso médio de CPU do comutador de função em *hardware*.



(b) Uso médio de CPU dos comutadores de função em *software*.

Figura 6.7: Uso médio de CPU dos comutadores de função nas diferentes configurações.

7 Conclusões

Neste trabalho, Foi proposta a virtualização de funções de rede para processamento de pacotes como um conjunto de regras de fluxo distribuídas por computadores OpenFlow heterogêneos programáveis. Precisamente, foram considerados comutadores em *software* com grandes tabelas de fluxo e comutadores de *hardware* com tabelas de fluxo pequenas, mas com alta capacidade de processamento. Explorar os recursos distintos, possivelmente conflitantes, no plano de dados heterogêneo é essencial para melhorar o desempenho geral da VNF, aumentando a capacidade agregada e economizando recursos.

Para melhorar o desempenho da VNF, este trabalho propõe mecanismos dinâmicos para balanceamento de carga e escalabilidade do plano de dados. Para validar a importância do balanceamento de carga no cenário proposto, foi comparado uma política simples (estática) com uma política que está ciente dos recursos disponíveis nos comutadores (dinâmica). Os resultados experimentais em um pequeno ambiente de testes confirmaram que a política dinâmica de balanceamento de carga aumenta a vazão agregada da VNF em até 27%, considerando uma alta carga de trabalho. Posteriormente, foi avaliada a operação conjunta do mecanismo de balanceamento de carga e uma nova solução de escalabilidade que ajusta o número de instâncias de comutadores de função em *software* no plano de dados de acordo com a demanda da rede. Os resultados da simulação confirmam que esses dois mecanismos são capazes de manter a alta vazão da função virtual, ao mesmo tempo em que permitem um uso inteligente e racional dos recursos de virtualização.

Como trabalhos futuros, é pretendido aprimorar os mecanismos propostos para considerar não apenas as características dos comutadores OpenFlow heterogêneos, mas também os indicadores de qualidade de serviço (QoS) para os fluxos de dados.

Bibliografia

ABDELWAHAB, S. et al. Network function virtualization in 5g. *IEEE Communications Magazine*, IEEE, v. 54, n. 4, p. 84–91, 2016.

AN, X.; KIESS, W.; PEREZ-CAPARROS, D. Virtualization of cellular network EPC gateways based on a scalable SDN architecture. In: *IEEE GLOBECOM*. [S.l.: s.n.], 2014.

CARPIO, F.; DHAHRI, S.; JUKAN, A. VNF placement with replication for load balancing in NFV networks. In: *IEEE ICC*. [S.l.: s.n.], 2017.

CEROVIC, D. et al. Fast packet processing: A survey. *IEEE Communications Surveys & Tutorials*, v. 20, n. 4, p. 3645–3676, 2018.

CHAVES, L. J. et al. Integrating OpenFlow to LTE: some issues toward Software-Defined Mobile Networks. In: IFIP. *International Conference on New Technologies, Mobility and Security (NTMS)*. [S.l.], 2015. p. 1–5.

CHAVES, L. J.; GARCIA, I. C.; MADEIRA, E. R. M. OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 support. In: ACM. *Workshop on ns-3 (WNS3)*. [S.l.], 2016. p. 33–40.

CHAVES, L. J.; GARCIA, I. C.; MADEIRA, E. R. M. An adaptive mechanism for LTE P-GW virtualization using SDN and NFV. In: IEEE. *International Conference on Network and Service Management (CNSM)*. [S.l.], 2017. p. 1–9.

CISCO SYSTEMS. *Cisco Annual Internet Report (2018—2023)*. 2020. White Paper.

COSTA, L. C. et al. OpenFlow data planes performance evaluation. *Performance Evaluation*, v. 147, p. 1–23, Maio 2021.

COSTA, L. C. et al. Performance evaluation of OpenFlow data planes. In: *IFIP/IEEE IM*. [S.l.: s.n.], 2017.

ETSI. *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action*. 2012. White Paper.

FEI, X. et al. Paving the way for NFV acceleration: A taxonomy, survey and future directions. *ACM Computing Surveys*, v. 53, n. 4, p. 1–42, 2020.

GEISSLER, S. et al. The power of composition: Abstracting a multi-device SDN data path through a single API. *IEEE Transactions on Network and Service Management*, v. 17, n. 2, p. 722–735, Junho 2020.

KAUR, S.; KAUR, K.; GUPTA, V. Implementing OpenFlow based distributed firewall. In: *IEEE InCITe*. [S.l.: s.n.], 2017.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, Janeiro 2015.

LAGHRISSI, A.; TALEB, T. A survey on the placement of virtual resources and virtual network functions. *IEEE Communications Surveys & Tutorials*, v. 21, n. 2, p. 1409–1434, 2019.

LI, Y.; CHEN, M. Software-defined network function virtualization: A survey. *IEEE Access*, IEEE, v. 3, p. 2542–2553, 2015.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, ACM New York, NY, USA, v. 38, n. 2, p. 69–74, 2008.

MIJUMBI J. SERRAT, J.-L. G. N. B. F. D. R.; BOUTABA, R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials, vol. 18, no. 1, pp.236–262*, 2015.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, v. 18, n. 1, p. 236–262, 2015.

NGUYEN, X.-N. et al. Rules placement problem in OpenFlow networks: A survey. *IEEE Communications Surveys & Tutorials*, v. 18, n. 2, p. 1273–1286, 2016.

ONF. *Software-Defined Networking: The New Norms for Networks*. 2012. White Paper.

PFAFF, B. et al. The design and implementation of Open vSwitch. In: USENIX. *Symposium on Networked Systems Design and Implementation (NSDI)*. [S.l.], 2015. p. 117–130.

RODRIGUES, C. P. et al. Avaliação de balanceamento de carga web em redes definidas por software. In: *SBRC*. [S.l.: s.n.], 2015.

WANG, C. et al. Toward high-performance and scalable network functions virtualization. *IEEE Internet Computing*, v. 20, n. 6, p. 10–20, Novembro/Dezembro 2016.

WOOD, T. et al. Toward a software-based network: Integrating software defined networking and network function virtualization. *IEEE Network*, v. 29, n. 3, p. 36–41, Maio/Junho 2015.

YI, B. et al. A comprehensive survey of network function virtualization. *Computer Networks*, v. 133, p. 212–262, Março 2018.

YOUSAF, F. Z. et al. Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 35, n. 11, p. 2468–2478, 2017.