Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Balanceamento Dinâmico de Carga Orientado à Qualidade de Serviço em Funções Virtuais Sobre Comutadores OpenFlow Heterogêneos

Pedro Clemente Pereira Bellotti

JUIZ DE FORA
FEVEREIRO, 2022

Balanceamento Dinâmico de Carga Orientado à Qualidade de Serviço em Funções Virtuais Sobre Comutadores OpenFlow Heterogêneos

Pedro Clemente Pereira Bellotti

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Sistemas de Informação

Orientador: Luciano Jerez Chaves

JUIZ DE FORA

FEVEREIRO, 2022

Balanceamento Dinâmico de Carga Orientado à Qualidade de Serviço em Funções Virtuais Sobre Comutadores OpenFlow Heterogêneos

Pedro Clemente Pereira Bellotti

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 SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Luciano Jerez Chaves
Doutor em Ciência da Computação

Alex Borges Vieira
Doutor em Ciência da Computação

Eduardo Pagani Julio
Doutor em Ciência da Computação

JUIZ DE FORA
21 DE FEVEREIRO, 2022

Resumo

O paradigma de virtualização de funções de rede substitui dispositivos de *hardware* dedicado por *software*, executando em máquinas virtuais ou em servidores comuns. No entanto, para superar os problemas de desempenho e escalabilidade das funções virtuais de rede, especialmente daquelas que demandam um intenso processamento de pacotes, é possível implementar as funções virtuais como regras distribuídas sobre comutadores programáveis de uma rede definida por *software*. Essa abordagem permite que os provedores de infraestrutura combinem comutadores programáveis tanto em *hardware* quanto em *software*, explorando o equilíbrio entre a velocidade no processamento de pacotes e a capacidade das tabelas de fluxo programáveis. Este trabalho busca alcançar o equilíbrio mencionado através de um mecanismo de escalabilidade do plano de dados de uma função virtual de rede, que decide o tipo e a quantidade de comutadores necessários para atender a demanda da rede, além de um mecanismo de balanceamento de carga para distribuição de fluxos entre os comutadores ativos. Os mecanismos propostos são avaliados através de simulações em diferentes cenários, observando aspectos relativos à escalabilidade, confiabilidade e conformidade com os indicadores de qualidade de serviço dos tráfegos individuais.

Palavras-chave: Virtualização das funções de rede, redes definidas por *software*, protocolo OpenFlow, comutadores programáveis heterogêneos, balanceamento de carga, escalabilidade das funções de rede.

Abstract

The network function virtualization paradigm replaces dedicated hardware devices with software running on virtual machines or regular servers. However, to overcome the performance and scalability problems of virtual network functions, especially those that demand intensive packet processing, it is possible to implement the virtual functions as distributed rules over programmable switches within a software-defined network. This approach allows infrastructure providers to combine programmable switches in both hardware and software, exploiting the tradeoff between packet processing speed and flow table capacity. This work aims to exploit these features through a data plane scaling mechanism that decides the type and number of heterogeneous switches required to meet the virtual network function demand, in addition to a load balancing mechanism for distributing flows between active switches. The proposed mechanisms are evaluated through simulations in different scenarios, observing aspects related to scalability, reliability and compliance with individual quality of service indicators.

Keywords: Network function virtualization, software-defined networking, OpenFlow protocol, heterogeneous programmable switches, load balancing, scalability of network functions.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviações

DPDK	*Data Plane Development Kit*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
MANO	*Management and Orchestration*
NFV	*Network Function Virtualization*
P-GW	*Packet Gateway*
QoS	*Quality of Service*
SDN	*Software Defined Networking*
TCAM	*Ternary Content-Addressable Memory*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
VNF	*Virtual Network Function*
VoIP	*Voice over IP*

1 Introdução

A demanda por conectividade à Internet cresce a cada ano. De acordo com o relatório da Cisco (CISCO, Inc., 2020), quase dois terços da população global terá acesso à Internet até 2023. Além disso, mais de 29 bilhões de dispositivos estarão conectados, o que representa mais de três vezes a população global em 2023. Essa demanda crescente coloca mais pressão sobre a evolução das arquiteturas de rede de computadores em direção a um modelo sustentável para oferecer acesso ubíquo sem aumentar as despesas operacionais.

As arquiteturas de rede tradicionais são limitadas e não podem oferecer o desempenho necessário para uma demanda cada vez maior, principalmente porque várias funções de rede ainda dependem de *hardware* dedicado. Além disso, a maioria dos provedores de infraestrutura dependem do provisionamento em excesso de recursos para acomodar o tráfego nos horários de pico, incorrendo em custos mais altos, grande consumo de energia e desperdício de recursos fora de tais horários (YI et al., 2018; MONTAZEROLGHAEM; YAGHMAEE; LEON-GARCIA, 2020).

Nesse contexto, o paradigma de virtualização de funções de rede (NFV, do Inglês *Network Function Virtualization*) surge como uma alternativa atraente para redução de custos e aumento da flexibilidade de arquiteturas de rede. A NFV fornece o desacoplamento de equipamentos físicos de rede (*hardware*) das funções que rodam neles (*software*). Este paradigma aproveita os benefícios da computação em nuvem, substituindo dispositivos de rede tradicionalmente implementados em *hardware* dedicado por *software* executado em máquinas virtuais ou contêineres, permitindo uma nova maneira mais ágil de projetar, implantar e gerenciar serviços de rede.

A flexibilidade do NFV vem acompanhada de novos desafios, principalmente aqueles relacionados ao desempenho e à escalabilidade (WANG et al., 2016). Algumas funções virtuais de rede (VNFs, do Inglês *Virtual Network Functions*) 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 genéricos

ainda apresenta desempenho moderado quando comparado às plataformas de *hardware* dedicado (NGUYEN et al., 2016).

Para atender às expectativas de desempenho e flexibilidade podemos utilizar tecnologias recentes para acelerar a manipulação de pacotes em ambientes virtuais, como é o caso do Open vSwitch (PFAFF et al., 2015) que foi construído sobre o conjunto de desenvolvimento de plano de dados (DPDK, do Inglês *Data Plane Development Kit*) da Intel, além de replicar as VNFs e balancear a carga entre as instâncias disponíveis (LAGHRISSI; TALEB, 2019; HAMDAN et al., 2021). Neste sentido, o paradigma de redes definidas por *software* (SDN, do Inglês *Software Defined Networking*), juntamente com sua implementação padrão OpenFlow, representam uma abordagem moderna para gerenciamento de tráfego em uma rede baseada em comutadores. O SDN centraliza a inteligência da rede em *software* e distribui o plano de dados sobre *hardware* programável otimizado para o encaminhamento de pacotes. Podemos também contar com o desacoplamento entre o estado da VNF e o processamento (KABLAN et al., 2017; SZALAY et al., 2019), concentrando na implementação do plano de dados como uma coleção de regras de fluxo distribuídas entre comutadores programáveis. É importante enfatizar que os paradigmas de SDN e NFV são totalmente independentes, mas podem ser implantados juntos como soluções complementares.

Este trabalho busca explorar a integração entre os paradigmas de SDN e NFV para implementar uma VNF como um conjunto de regras de fluxo distribuídas sobre de comutadores programáveis heterogêneos. Para isso, são propostos um mecanismo de balanceamento de carga e um mecanismo de escalabilidade do plano de dados. O cenário proposto foi instanciado no simulador de redes ns-3 e os resultados experimentais mostram que a combinação desses mecanismos permite que a demanda da VNF seja atendida com qualidade ao mesmo tempo que economiza recursos de infraestrutura sob diferentes cargas de tráfego na rede. De maneira resumida, as contribuições deste trabalho são:

- Incorporação de um plano de dados heterogêneo sem estado de uma função de rede de processamento de pacotes como uma coleção de regras de fluxo distribuídas entre comutadores OpenFlow programáveis em *hardware* e *software*;

- Criação de um mecanismo de escalabilidade para ajustar o tipo e número de comu-

tadores OpenFlow ativos no plano de dados heterogêneo da VNF;

- Uma política de balanceamento de carga dinâmica para operação conjunta com o mecanismo de escalabilidade do plano de dados, sustentando a demanda da VNF com uso econômico de recursos.

O restante deste trabalho está organizado como segue: o Capítulo 2 apresenta os conceitos básicos das tecnologias utilizadas. Os trabalhos relacionados são apresentados no Capítulo 3. O Capítulo 4 descreve a topologia da VNF com os comutadores heterogêneos usados para o processamento de pacotes, bem como o mecanismo de balanceamento de carga e o mecanismo de escalabilidade do plano de dados. No Capítulo 5 são apresentados a metodologia de avaliação e os resultados dos experimentos com um modelo de tráfego sintético. O Capítulo 6 mostra resultados para o mesmo cenário, porém utilizando um modelo de tráfego com características realistas. Finalmente, o Capítulo 7 conclui este trabalho e apresenta as perspectivas para outros trabalhos futuros. O Apêndice A apresenta uma visualização alternativa dos algoritmos apresentados no Capítulo 4.

2 Fundamentação teórica

Este capítulo apresenta os dois paradigmas que sustentam este trabalho: o SDN e o NFV. A Seção 2.1 descreve as redes definidas por *software*, além de detalhar o funcionamento de seu protocolo de referência: o OpenFlow. Na sequência, a Seção 2.2 apresenta os fundamentos relacionados à virtualização das funções de rede. Por fim, a Seção 2.3 conclui o capítulo discutindo sobre os benefícios da integração entre esses dois paradigmas.

2.1 Redes definidas por software

As redes definidas por *software* (SDN) representam um paradigma que centraliza a inteligência da rede em *software* e distribui o plano de dados sobre *hardware* programável, eliminando os tradicionais algoritmos distribuídos executados individualmente em cada elemento da rede. Conforme ilustra a Figura 2.1 de (ALAM; KATIB; ALZAHRANI, 2013), a arquitetura de referência do SDN é dividida em três camadas: infraestrutura, que contém os elementos de rede responsáveis pelo encaminhamento dos pacotes; controle, que é responsável pela tomada de decisão e administração da infraestrutura; e aplicação, que contém os serviços que operam na rede.

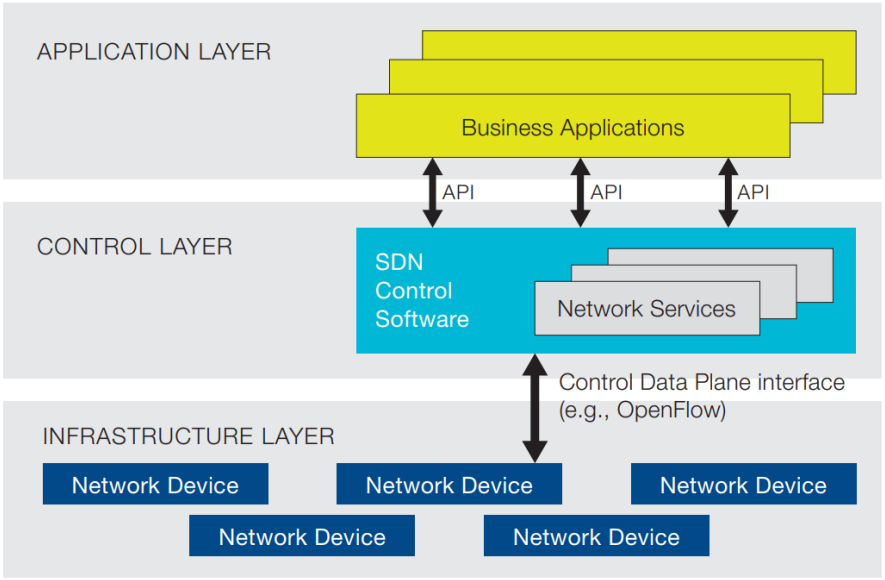


Figura 2.1: Arquitetura de referência do SDN.

Para aumentar a flexibilidade do gerenciamento e a facilidade de configuração da rede, um *software* centralizado chamado de controlador é utilizado para monitorar e (re)configurar a rede, analisando em tempo real os fluxos de dados e o estado da rede para otimizar as rotas de encaminhamento dos pacotes.

O protocolo OpenFlow foi criado para permitir a comunicação entre as camadas de infraestrutura e controle de uma rede SDN. Logo, ele precisa ser implementado em todos os dispositivos programáveis de encaminhamento e também no controlador. A comunicação entre um comutador e o controlador é feita através de um canal de administração da rede, onde todas as mensagens devem seguir o padrão estabelecido pelo protocolo. Para identificar os fluxos de dados e realizar ações específicas para cada pacote, o controlador instala regras no comutador, que por sua vez utiliza essas regras para inspecionar os pacotes de dados e decidir a rota de encaminhamento. Vale ressaltar que um comutador pode receber regras de diferentes controladores simultaneamente pelo canal de administração.

As regras instaladas num comutador são armazenadas em tabelas de fluxo, grupo e de medição. As regras utilizadas para encaminhamento de pacotes são armazenadas na tabela de fluxo, onde cada regra contém os campos que serão inspecionados no cabeçalho dos pacotes para a devida identificação do fluxo, as estatísticas indicando o uso da regra, além de um conjunto de instruções que serão aplicadas aos pacotes deste fluxo.

A tabela de grupo pode ser usada para armazenar conjuntos de instruções sofisticados como, por exemplo, diversas regras de fluxo que juntas fazem o balanceamento de carga entre os enlaces adjacentes ao comutador. Por fim, na tabela de medição são armazenadas as regras que monitoram a vazão dos fluxos, podendo ser usadas para descartar pacotes excedentes de modo a implementar mecanismos como os limitadores de vazão.

Conforme ilustrado pela Figura 2.2, o processamento dos pacotes em um comutador OpenFlow acontece da seguinte maneira: ao receber um pacote, o comutador inspeciona o mesmo e procura na primeira tabela de fluxo uma regra correspondente ao pacote recebido, respeitando as prioridades definidas em cada regra. Caso uma correspondência seja encontrada, o comutador realiza as instruções contidas na regra, que podem ser o encaminhamento do pacote para uma das portas de saída do comutador, direcionamento do pacote para processamento nas tabelas de medição ou de grupo, ou modificações no

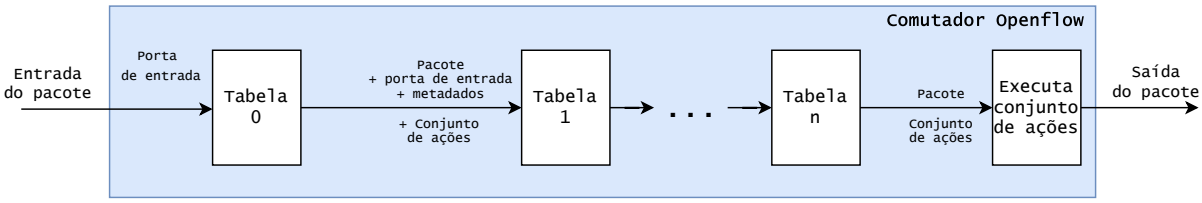


Figura 2.2: Processamento de pacotes em um comutador OpenFlow.

conteúdo do pacote. Caso contrário, o comutador continua a busca nas tabelas de fluxo seguintes (caso existam). Na ausência de correspondência ao final das tabelas de fluxo, o comutador pode descartar o pacote ou enviá-lo ao controlador para que este possa extrair as informações do pacote e decidir pelo correto encaminhamento do mesmo.

2.2 Virtualização das funções de rede

O paradigma de virtualização das funções de rede (NFV) tem como objetivo desacoplar os equipamentos físicos de rede (*hardware*) das funções que rodam neles (*software*). Através dessa separação, o NFV executa diversas funções de rede em uma infraestrutura de *hardware* compartilhado, substituindo o *hardware* dedicado por *hardware* genérico de menor custo e reduzindo custos de operação e manutenção (MIJUMBI et al., 2015a).

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. Com a virtualização é possível executar diversas aplicações simultaneamente usando um *hardware* físico compartilhado. Esse conceito já é altamente conhecido e utilizado em diversas cenários. Entretanto, para funções tradicionais de rede, como os *firewalls* e *gateways*, onde o *hardware* e suas implementações são definidas pelo fabricante, o desenvolvimento de novos recursos e serviços é muito difícil, uma vez que o funcionamento do *hardware* não pode ser modificado.

A arquitetura de referência do NFV é apresentada na Figura 2.3. As funções virtuais de rede (VNFs) estão no topo da arquitetura, onde cada VNF representa um elemento virtualizado, como roteadores ou *firewalls*. Abaixo das VNFs temos os recursos virtuais e físicos, que são os recursos de *software* e *hardware* necessários para a instanciação de uma VNF. O elemento responsável por garantir que existam recursos como processamento e

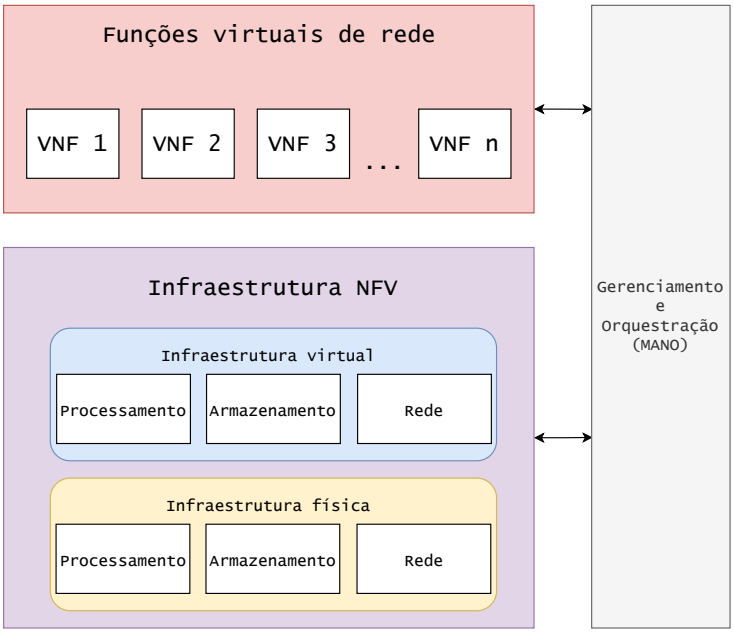


Figura 2.3: Arquitetura de referência do NFV.

armazenamento disponíveis para cada VNF é o gerenciador e orquestrador (MANO, do Inglês *Management and Orchestration*).

Diversas tecnologias para acelerar o manuseio de pacotes em ambientes virtuais estão disponíveis, em especial o FD.io, que é uma uma coleção de bibliotecas especialmente pensada para viabilizar serviços flexíveis e programáveis em uma plataforma de *hardware* genérica. Através de uma biblioteca de processamento vetorial de pacotes, a ferramenta permite aos desenvolvedores especificar os fluxos de processamento de pacotes através de diferentes grafos de encaminhamento. A ferramenta aproveita os recursos do DPDK da Intel para acelerar o plano de dados por meio de uma camada de abstração que fornece uma interface de programação padrão para aceleradores de *hardware*, bibliotecas e outros elementos do sistema operacional.

2.3 Integração entre o SDN e o NFV

O SDN e o NFV permitem que os arquitetos de rede projetem, implementem e gerenciem serviços com muito mais eficiência. É nesta linha que a Figura 2.4[1] ilustra os benefícios que podem ser alcançados com a junção destes paradigmas. Ao explorar a separação entre os planos de dados e de controle proposta pelo SDN em conjunto com a desassociação

[1] ⟨https://www.redeemsystems.com/sdnnfv.php⟩

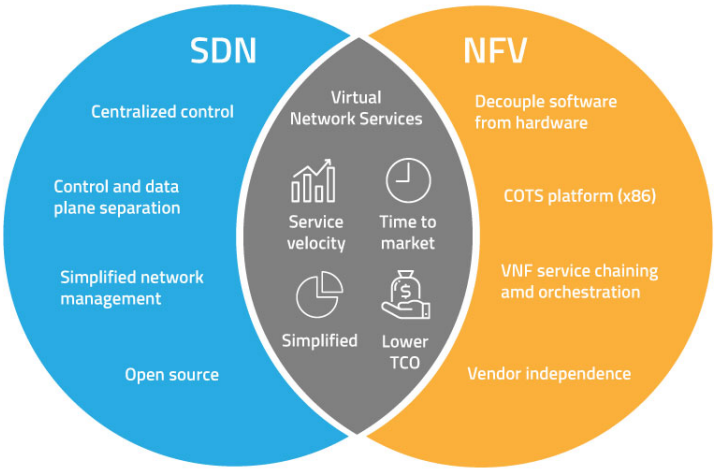


Figura 2.4: Benefícios da integração entre o SDN e o NFV.

entre o *hardware* e a função em *software* proposta pelo NFV, é possível alcançar uma velocidade de serviço mais rápida, menores custo de entrada, testes de prova de conceito feitos rapidamente e tempo até o mercado reduzido.

Um dos atrativos do uso desses dois paradigmas é o custo. Enquanto o SDN reduz as despesas operacionais por meio da automação da configuração da rede, permitindo o ajuste dos fluxos de forma rápida e com a possibilidade de antecipação de respostas, o NFV elimina a necessidade de adquirir *hardware* de rede especializado para cada função. Isso diminui a demanda por equipamentos, espaço físico e consumo de energia.

A flexibilidade também é um ponto importante, dado que as interfaces programáveis permitem o provisionamento de novos dispositivos e a reconfiguração automatizada de dispositivos existentes. Além disso, esses paradigmas permitem o reposicionamento das VNFs próximo aos usuários na borda da rede, otimizando os indicadores de qualidade do serviço (QoS, do Inglês *Quality of Service*) (LEONHARDT, 2020).

Recentemente, o SDN e o NFV também estão sendo usados em conjunto como habilitadores das funções de serviço encadeadas, muito importantes no contexto de fatiamento de rede. Estas funções encadeadas são definidas dinamicamente por um conjunto ordenado de funções individuais que devem ser aplicadas aos pacotes para oferecer um determinado serviço (HALPERN; PIGNATARO et al., 2015). Com a facilidade de instanciação de uma VNF oferecida pelo NFV e com a programabilidade do roteamento suportada pelo SDN fica mais simples e ágil gerenciar esse encadeamento, independente de onde as VNFs estejam, de fato, localizadas na rede.

3 Trabalhos relacionados

Este capítulo apresenta a revisão da literatura no contexto deste trabalho. A Seção 3.1 aborda o balanceamento de carga entre funções de rede. Por sua vez, a Seção 3.2 discute sobre a escalabilidade das funções virtuais de rede. Em ambos os casos, os paradigmas SDN e NFV apresentam-se como soluções atrativas para viabilizar estas propostas, reduzindo os custos e aumentando a flexibilidade da rede (YI et al., 2018; MIJUMBI et al., 2015b).

3.1 Balanceamento de carga

O balanceamento de carga é a técnica que visa otimizar a utilização da rede, dividindo a demanda pelos diferentes recursos como enlaces, comutadores e servidores. Lidar com cargas de trabalho pesadas pode exigir tecnologias avançadas para processamento rápido de pacotes em servidores convencionais, bem como um balanceamento de carga entre réplicas de uma VNF (WANG et al., 2016). Os paradigmas SDN e NFV são independentes um do outro, mas os provedores de infraestrutura podem implantá-los juntos, como soluções complementares. Uma estratégia comumente adotada na literatura como solução para o desafio da escalabilidade é replicar VNFs e usar a visão global oferecida pelo SDN para balancear a carga de trabalho entre os recursos disponíveis (LAGHRISSI; TALEB, 2019; HAMDAN et al., 2021; KREUTZ et al., 2015). Problemas relacionados à interconexão e posicionamento de VNFs em redes SDN foram discutidos por (CARPIO; DHAHRI; JUKAN, 2017) e (LAGHRISSI; TALEB, 2019). Esses autores consideraram soluções de posicionamento orientadas ao balanceamento de carga entre os enlaces de rede. Eles também examinaram aspectos relacionados à replicação de instâncias de VNFs para acomodar a demanda da rede. Neste mesmo contexto, a virtualização do plano de dados de um *gateway* de pacotes (P-GW, do Inglês *Packet Gateway*), utilizado na arquitetura de redes móveis 4G, é proposta por (AN; KIESS; PEREZ-CAPARROS, 2014). Os autores adotaram um conjunto local de réplicas da VNF e um par de comutadores OpenFlow para distribuir o tráfego entre as instâncias ativas, proporcionando a escalabilidade da VNF.

Uma vasta revisão da literatura sobre soluções para o balanceamento de tráfego em centros de dados é feita por (ZHANG et al., 2018), que discute sobre os mecanismos centralizados para distribuição dos fluxos entre os enlaces de uma infraestrutura de rede SDN. Já (CEROVIC et al., 2018) e (FEI et al., 2020) pesquisaram sobre as tecnologias recentes para acelerar o processamento de pacotes em ambientes virtuais. Por sua vez, (JAMALI; BADIRZADEH; SIAPOUSH, 2019) também discutem sobre o balanceamento de carga entre enlaces e comutadores numa rede SDN, utilizando técnicas de programação genética para encontrar rotas que evitem gargalos e que minimizem o custo das operações de configuração dos comutadores.

3.2 Escalabilidade de VNFs

Nos trabalhos anteriores, a existência de uma infraestrutura SDN atua como um facilitador para a implementação das técnicas de balanceamento de carga, dado a fácil programabilidade destas redes (KAUR; MANGAT; KUMAR, 2020). Entretanto, o SDN também pode ser usado para implementar funções virtuais de rede no contexto de NFV, principalmente aquelas que demandam de inspeção individual de pacotes.

A natureza implícita sem estado (*stateless*) de algumas funções de rede as tornam adequadas para instanciação em comutadores OpenFlow, demandando interação mínima com o plano de controle do SDN. No entanto, várias outras funções de rede possuem estado (*stateful*), e sua virtualização direta incorre em novos desafios de implementação. (KABLAN et al., 2017) discutiram como o forte acoplamento entre estado e processamento dificulta a escalabilidade e a confiabilidade de uma VNF. Eles também propuseram quebrar esse acoplamento e mover o estado da VNF para um armazenamento de dados de baixa latência compartilhado entre instâncias de planos de dados sem estado, obtendo maior elasticidade e resiliência da função.

Um *firewall* sem estado foi implementado de maneira distribuída no trabalho de (KAUR; KAUR; GUPTA, 2017). Os autores usaram uma rede de comutadores OpenFlow e espalharam as regras de processamento de fluxos entre os comutadores dessa rede, eliminando o ponto único de falha comumente presente nas arquiteturas tradicionais. Em (RODRIGUES et al., 2015) é apresentado um cenário onde os autores substituem

os equipamentos especializados por um comutador OpenFlow ligado à um controlador centralizado para realizar o balanceamento de carga em nível de aplicação. Usando essa arquitetura, os autores implementaram diferentes políticas para distribuir requisições entre servidores Web, oferecendo escalabilidade ao serviço.

No trabalho de (CHAVES; GARCIA; MADEIRA, 2017), os autores virtualizaram o plano de dados do *gateway* P-GW do 4G usando comutadores OpenFlow gerenciados por um controlador SDN local. Eles implementaram as tarefas do plano de dados do P-GW combinando regras de fluxo, de medição e de grupo. Pelo menos uma regra OpenFlow é instalada nos comutadores para cada fluxo de tráfego ativo na rede. Essas regras são responsáveis pela classificação, contabilidade e encaminhamento do tráfego dentro da arquitetura da rede celular. Para resolver os problemas de escalabilidade, os autores distribuíram essas regras entre vários comutadores OpenFlow homogêneos. Com isso, eles apresentaram um mecanismo adaptativo para ativar ou desativar os comutadores OpenFlow e movimentar as regras entre eles, de acordo com a demanda da rede.

Inspirado por (CHAVES; GARCIA; MADEIRA, 2017) e (KABLAN et al., 2017), o trabalho atual se concentra na implementação do plano de dados de uma VNF sem estado como um conjunto de regras de fluxo distribuídas entre comutadores OpenFlow programáveis que são gerenciados por um controlador SDN local. No entanto, diferentemente das soluções anteriores, o cenário adotado aqui consiste num plano de dados OpenFlow heterogêneo. Portanto, essa VNF combina o poder de comutadores com *hardware* dedicados para garantir o processamento de pacotes com alto desempenho aliado à facilidade na instanciação de comutadores de *software* rodando sobre servidores de uso geral. As infraestruturas OpenFlow heterogêneas são uma realidade, e lidar com essa diversidade é essencial para otimizar o uso de recursos, reduzir os custos e fomentar a próxima geração de arquiteturas de rede (GEISSLER et al., 2020).

4 Cenário e mecanismos propostos

Este capítulo apresenta as principais contribuições deste trabalho. A Seção 4.1 descreve o cenário proposto para funções de processamento de pacotes composto por um plano de dados com comutadores OpenFlow heterogêneos. A Seção 4.2 detalha o mecanismo de balanceamento de carga dinâmico para distribuir a demanda da VNF entre os comutadores ativos. Por fim, a Seção 4.3 revela o mecanismo de escalabilidade para ajustar o tipo e o número de comutadores OpenFlow no plano de dados heterogêneo da VNF.

4.1 Cenário heterogêneo para VNFs

A implementação do plano de dados sem estado de uma função virtual de rede sobre comutadores OpenFlow programáveis para tomada de decisão seletiva de pacotes é uma abordagem atraente, pois a inspeção, encaminhamento e descarte de pacotes são ações comumente executadas pelos comutadores OpenFlow. No entanto, como o número de regras de fluxo necessárias para implementar uma VNF pode aumentar consideravelmente com a carga da rede e a complexidade da função, um único comutador OpenFlow pode não conseguir lidar com todo o tráfego de maneira eficiente. A solução imediata é combinar vários comutadores OpenFlow e dividir a carga de trabalho entre eles. Nessa linha, a Figura 4.1 retrata a topologia genérica para a VNF considerada neste trabalho.

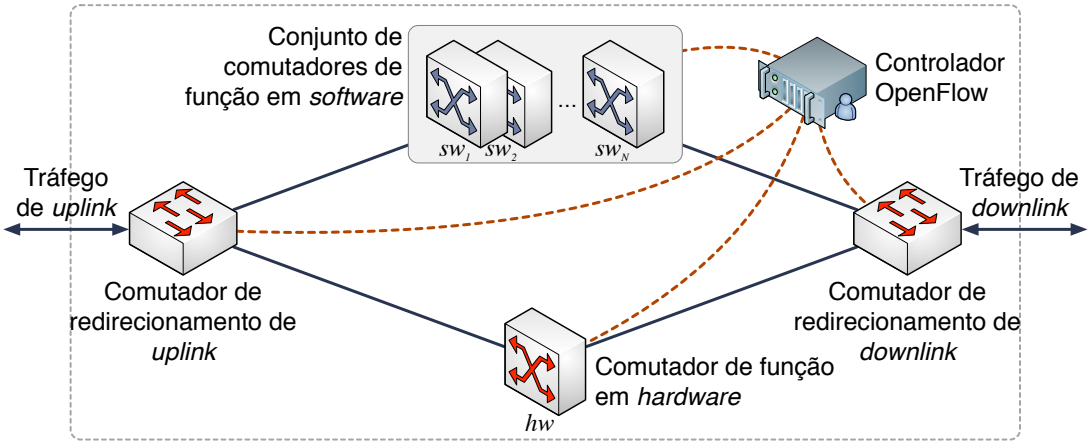


Figura 4.1: Topologia genérica para a VNF com comutadores heterogêneos.

Nessa topologia, existe um conjunto $\mathcal{S} = \{sw_1, \ldots, sw_n\}$ de comutadores de função em *software* e um comutador de função em *hardware* (hw). As regras OpenFlow distribuídas entre esses comutadores de função heterogêneos são as que permitem que as tarefas da VNF sejam executadas de maneira escalável. Além disso, os comutadores de redirecionamento de *uplink* (ul) e *downlink* (dl) à esquerda e à direita da figura, respectivamente, atuam simultaneamente como pontos de entrada e saída de tráfego na VNF. Esses dois comutadores de redirecionamento também são responsáveis por impor o balanceamento de carga entre os comutadores de função. Por fim, o controlador OpenFlow local gerencia o tráfego dentro da VNF, instalando, atualizando e removendo as regras nos comutadores de função e redirecionamento para implementar as políticas desejadas.

Sem perder a generalidade, poderíamos utilizar essa topologia para implementar um *firewall* de perímetro sem estado. Nesse cenário, os comutadores de redirecionamento garantiriam a conexão da VNF com a rede local e com a Internet. As regras de fluxo distribuídas pelos comutadores de função implementariam as tarefas do *firewall*, como identificar o endereço 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, também seria possível estender esse *firewall* para uma versão com estado, onde as informações sobre as conexões ativas seriam salvas no controlador centralizado.

Uma característica fundamental da topologia apresentada na Figura 4.1 é o uso de comutadores de função heterogêneos, representados por comutadores OpenFlow baseados em *software* e *hardware*. Conforme discutido por (COSTA et al., 2021), os comutadores OpenFlow implementados em *hardware* são otimizados para o encaminhamento de pacotes, suportam altas cargas de trabalho e apresentam atraso mínimo. No entanto, eles sofrem de uma limitação significativa: o pequeno tamanho das tabelas de fluxo. Esse problema surge pelo uso da memória endereçável por conteúdo ternária (TCAM, do Inglês *Ternary Content-Addressable Memory*), uma tecnologia de memória cara e exigente em consumo de energia que é usada nestes dispositivos com a finalidade específica para oferecer pesquisa rápida de campos com casamento curinga. Por outro lado, os comutadores OpenFlow implementados em *software* (como é o caso do Open vSwitch[2]) possuem tabelas de fluxo praticamente

[2]⟨http://www.openvswitch.org⟩

infinitas e podem ser facilmente instanciados e configurados em servidores de prateleira. Ainda assim, o atraso introduzido ao processar pacotes em um ambiente virtualizado pode ser uma ordem de grandeza maior do que o atraso imposto por um comutador em *hardware*. Este atraso limita consideravelmente a capacidade do comutador em *software*. Em suma, esta topologia com comutadores de função heterogêneos apresenta vários benefícios. O provedor de infraestrutura pode manter um único (ou poucos) comutadores OpenFlow em *hardware* para sustentar a operação contínua de alto desempenho da VNF. No entanto, ele pode contar com vários comutadores OpenFlow em *software*, instanciados temporariamente sob demanda, para lidar com picos intermitentes de tráfego.

Apresentaremos nas próximas seções dois mecanismos de gerenciamento cientes da topologia heterogênea da VNF. Primeiro, na Seção 4.2, consideramos o mecanismo de balanceamento de carga para distribuir o trabalho entre os comutadores de função. Como os comutadores de redirecionamento podem estar sujeitos à altas cargas de tráfego, eles exigem dispositivos em *hardware* e com desempenho elevado, possivelmente com restrições no tamanho da tabela. Portanto, a estratégia de balanceamento de carga aplicada nesses comutadores devem contar com um número moderado de entradas de fluxo. Na sequência, examinamos na Seção 4.3 a escalabilidade do plano de dados, onde o controlador OpenFlow monitora a carga da VNF para ajustar dinamicamente o número de comutadores de função em *software* ativos. Nesse caso, o mecanismo de balanceamento de carga deve estar ciente das alterações no plano de dados da VNF, redistribuindo a carga de trabalho entre os comutadores de função quando necessário.

4.2 Mecanismo de balanceamento de carga

Sempre que um pacote não corresponder a nenhuma regra existente nas tabelas de fluxo dos comutadores de função, significa que um novo fluxo f está chegando na VNF (identificado pela quíntupla IPs de origem/destino, portas de origem/destino e protocolo de transporte). Nesse caso, o comutador envia o pacote ao controlador local que irá executar o Algoritmo 4.1 para admissão do novo fluxo. O algoritmo verifica inicialmente os recursos disponíveis. Se o uso médio das tabelas de fluxo $T(\mathcal{S})$ e o uso médio das CPUs $C(\mathcal{S})$ dos comutadores de função em *software* em $\mathcal{S}$ não excederem o limite de bloqueio κ, o controlador aceita o novo

Algoritmo 4.1: ADMISSÃO DE NOVO FLUXO

entradas: O conjunto $\mathcal{R}_f$ de regras de função do novo fluxo f
O indicador $T(sw_i)$ de uso da tabela de fluxo, $\forall sw_i \in \mathcal{S}$
O indicador $C(sw_i)$ de uso da CPU, $\forall sw_i \in \mathcal{S}$
O limiar κ de bloqueio
saída: Verdadeiro se o novo fluxo for aceito, falso caso contrário

```
1  início
       /* Calcula os indicadores médios para os comutadores em software */
2      T(S) ← (1/n) Σ_{i=1}^{n} T(sw_i)
3      C(S) ← (1/n) Σ_{i=1}^{n} C(sw_i)
       /* Verifica a disponibilidade de recursos */
4      se T(S) < κ e C(S) < κ então
           /* Instala o fluxo nos comutadores em software */
5          para cada regra r ∈ R_f faça
6              para cada comutador sw_i ∈ S faça
7                  Instala uma cópia da regra r no comutador sw_i
8          retorna verdadeiro
9      senão
10         retorna falso
```

fluxo e instala seu conjunto individual de regras de função $\mathcal{R}_f$ em todos os comutadores de função em *software*. Caso contrário, o controlador bloqueia a requisição, dada a falta de recursos na VNF. A Figura A.1 apresenta o fluxograma para este algoritmo.

Sempre que o controlador admite um novo fluxo, ele deve instalar uma cópia completa do conjunto de regras $\mathcal{R}_f$ em todos os comutadores de função em *software* ativos. Assim, podemos usar uma única regra de grupo nos comutadores de redirecionamento para encaminhar pacotes de todos os fluxos para os comutadores de função em *software*. Os grupos do OpenFlow representam conjuntos de ações de encaminhamento complexas. Aqui, usamos uma regra de grupo do tipo de seleção (*select*) para distribuir uniformemente os pacotes (e consequentemente, a carga de tráfego) entre os comutadores de função em *software* ativos. É importante enfatizar que esta abordagem exige que todos os comutadores de função em *software* possuam as regras para processar todos os fluxos, evitando perda de pacotes devido à falta de regras nas tabelas de fluxo. Instalar cópias de todas as regras de fluxo em todos os comutadores de função *software* não é um problema, pois as tabelas

de fluxo nesses comutadores são consideravelmente grandes.

A abordagem adotada pelo Algoritmo 4.1 tem como objetivo aumentar a capacidade geral da VNF explorando as grandes tabelas de fluxo dos comutadores de função em *software*. No entanto, esta estratégia possui um ponto negativo. Conforme discutido na Seção 4.1, o processamento intensivo de pacotes em ambientes virtuais apresenta um atraso considerável, o que reduz a capacidade final de processamento de um comutador em *software* (COSTA et al., 2017; COSTA et al., 2021). Como consequência, os comutadores de função em *software* ficarão rapidamente sobrecarregados, possivelmente resultando em alta perda de pacotes e elevadas taxas de bloqueio. Para lidar com esse problema, o Algoritmo 4.2 realiza periodicamente a realocação dos fluxos ativos para balanceamento de carga entre os comutadores de função.

O controlador da VNF monitora continuamente as estatísticas de tráfego dos fluxos através das mensagens de requisição disponíveis no protocolo OpenFlow, que são enviadas periodicamente para todos os comutadores de função. O controlador inspeciona os

Algoritmo 4.2: Realocação dos fluxos ativos

entradas: O conjunto $\mathcal{F}_{sw}$ de todos os fluxos instalados em $\mathcal{S}$
O conjunto $\mathcal{R}_f$ de regras de função do fluxo, $\forall f \in \mathcal{F}_{sw}$
O indicador $V(f)$ de vazão individual do fluxo, $\forall f \in \mathcal{F}_{sw}$
O indicador $T(hw)$ de uso de tabela de fluxo
O limiar κ de bloqueio

```
1  início
2      Ordena os fluxos de F_sw pelo indicador de vazão V(f)
3      enquanto T(hw) < κ faça
4          f' ← fluxo em F_sw com a maior vazão V(f)
5          Move para hardware (f')
6          Remove f' de F_sw

7  procedimento Move para hardware (fluxo f) é
8      para cada regra r ∈ R_f faça
9          Instala uma cópia da regra r no comutador hw
10         para cada comutador sw_i ∈ S faça
11             Escalona a remoção da regra r do comutador sw_i
12     Instala uma regra de maior prioridade nos comutadores ul e dl para
         encaminhar novos pacotes do fluxo f para o comutador hw
```

valores recebidos e calcula o indicador de vazão individual $V(f)$, para cada fluxo f ativo na VNF. Conforme detalhado no Algoritmo 4.2, o controlador usa este indicador para ordenar de maneira decrescente pela vazão todos os fluxos atualmente atribuídos aos comutadores de função em *software*. Então, enquanto houver espaço na tabela de fluxo do comutador de função em *hardware*, o controlador move o conjunto de regras de função do fluxo com a maior vazão dos comutadores de função em *software* para o comutador de função *hardware*. Sendo assim, ao final da execução deste algoritmo, os poucos fluxos que possuem alta taxa de vazão individual (os chamados fluxos "elefantes") estarão alocados no comutador de função em *hardware*. Em contrapartida, os inúmeros outros fluxos com pequena taxa de vazão individual (os chamados fluxos "ratos") permanecerão alocados nos comutadores de função em *software*. Em suma, o mecanismo proposto explora a alta capacidade de processamento do comutador de função em *hardware* sem extrapolar sua tabela de fluxo limitada. Além disso, os fluxos deixados nos comutadores de função em *software* possuem demanda agregada de processamento reduzida, melhorando os indicadores de perda de pacotes e taxa de bloqueio. A Figura A.2 apresenta o fluxograma para este algoritmo.

O procedimento para mover dinamicamente o conjunto de regras de função de um fluxo do comutador de *software* para o comutador de função de *hardware* funciona da seguinte maneira: para cada regra, o controlador instala uma cópia da mesma no comutador de função em *hardware* e (opcionalmente) programa a remoção da regra original de cada um dos comutadores de função em *software*. Em seguida, o controlador instala novas regras de fluxo com maior prioridade nos comutadores de redirecionamento para encaminhar os pacotes subsequentes desse fluxo específico para o comutador de função em *hardware*. Respeitando esta ordem de execução, o controlador minimiza eventuais perdas de pacotes durante este procedimento de realocação.

Observe que o Algoritmo 4.2 assume que os conjuntos de regras de função de todos os fluxos ativos na VNF são disjuntos. Essa suposição é natural aqui, pois estamos dentro de uma função de rede que processa os fluxos em alta granularidade. No entanto, para casos específicos envolvendo dependências entre regras de fluxos distintas, seria necessário monitorar essas dependências e possivelmente replicar algumas regras durante o procedimento de realocação (KATTA et al., 2016).

4.3 Mecanismo de escalabilidade do plano de dados

O tamanho da tabela de fluxo nos comutadores de função em *software* não é uma preocupação para a VNF. No entanto, mesmo com os fluxos "elefantes" sendo processados pelo comutador de função em *hardware*, ainda é possível que a vazão agregada dos fluxos "ratos" atinja a capacidade máxima do conjunto de comutadores de função em *software*. Para lidar com esse problema, o mecanismo de escalabilidade do plano de dados pode aumentar ou diminuir dinamicamente o número de comutadores de função em *software* para atender à demanda agregada da rede. O Algoritmo 4.3 detalha este mecanismo e a Figura A.3 apresenta o fluxograma simplificado para este algoritmo.

O controlador monitora continuamente os comutadores de função em *software* para medir seus indicadores individuais $C(sw_i)$ de uso da CPU. Esta informação permite que o

Algoritmo 4.3: Escalabilidade do plano de dados

entradas: O número atual n de comutadores de função em $\mathcal{S}$
O número máximo m de comutadores de função para $\mathcal{S}$
O indicador $C(sw_i)$ de uso da CPU, $\forall sw_i \in \mathcal{S}$
Os limiares de incremento θ_i e decremento θ_d

1 **início**
  /* *Calcula o uso médio de CPU para os comutadores em software* */
2 $C(\mathcal{S}) \leftarrow \frac{1}{n} \sum_{i=1}^{n} C(sw_i)$
  /* *Ativa um novo comutador em software* */
3 **se** $C(\mathcal{S}) > \theta_i$ **e** $n < m$ **então**
4 $n \leftarrow n + 1$
5 Ativa o novo comutador sw_n
6 **para cada** regra r instalada em sw_1 **faça**
7 Instala uma cópia da regra r no comutador sw_n
8 Atualiza as regras de grupos nos comutadores *ul* e *dl*
  /* *Desativa um (ou mais) comutador(es) em software* */
9 **senão se** $n > 1$ **então**
10 $C'(\mathcal{S}) \leftarrow \frac{1}{n-1} \sum_{i=1}^{n-1} C(sw_i)$
11 **enquanto** $C'(\mathcal{S}) < \theta_d$ **e** $n > 1$ **faça**
12 Desativa o comutador sw_n
13 $n \leftarrow n - 1$
14 $C'(\mathcal{S}) \leftarrow \frac{1}{n-1} \sum_{i=1}^{n-1} C(sw_i)$
15 Atualiza as regras de grupos nos comutadores *ul* e *dl*

Algoritmo 4.3 calcule o indicador $C(\mathcal{S})$ de uso médio da CPU para todos os comutadores de *software* ativos. Se $C(\mathcal{S})$ exceder o limite de incremento θ_i e o número máximo m de comutadores de função em *software* suportados pela infraestrutura ainda não foi atingido, o controlador ativa um novo comutador de função em *software*. Nesse caso, ele copia todas as regras atualmente instaladas nos outros comutadores de função em *software* para este novo. Finalmente, ele atualiza as regras de grupo nos comutadores de redirecionamento com o novo número n de comutadores de função em *software* ativos. Caso contrário, o Algoritmo 4.2 verifica se é possível diminuir o número atual n de comutadores de função em *software* ativos. Assim, ele calcula o indicador $C'(\mathcal{S})$ de uso médio da CPU para uma possível configuração com o número de comutadores de função em *software* igual a $n - 1$. Se $C'(\mathcal{S})$ for inferior ao limite de decremento θ_d, o controlador desativa um comutador de função em *software*. Esse processo é repetido até que $C'(\mathcal{S})$ seja maior que θ_d ou quando restar apenas um comutador de função em *software* ativo na VNF. Se ocorrer algum ajuste na quantidade n, o controlador também precisará atualizar as regras de grupo nos comutadores de redirecionamento.

5 Avaliação com modelo de tráfego sintético

Este capítulo apresenta o primeiro conjunto de testes que foi utilizado para avaliar os mecanismos propostos neste trabalho. Para isso, definimos um modelo de tráfego sintético, desenvolvido com o propósito específico de alcançar a capacidade máxima da VNF e verificar o comportamento dos mecanismos de balanceamento e escalabilidade em situações de uso extremo. A Seção 5.1 descreve a metodologia de avaliação adotada enquanto a Seção 5.2 apresenta os resultados obtidos e as discussões subsequentes.

5.1 Metodologia de avaliação

Para avaliar os mecanismos propostos neste trabalho, a topologia da VNF com comutadores OpenFlow heterogêneos apresentada na Figura 4.1 foi implementada no simulador de rede ns-3[3] estendido com o módulo OFSwitch13 para suporte ao protocolo OpenFlow versão 1.3 (CHAVES; GARCIA; MADEIRA, 2016).

Uma versão modificada de um aplicativo que utiliza o protocolo de transporte UDP (do Inglês *User Datagram Protocol*) foi instalado em pares de nós cliente/servidor para gerar tráfego com vazão unidirecional constante. A Tabela 5.1 descreve os parâmetros usados na geração do tráfego sintético. Cada cliente inicia novos fluxos seguindo um processo de *Poisson* com $\lambda = 0{,}5$ fluxos por segundo. A quantidade de pares cliente/servidor foi definida em 50 para gerar uma carga de trabalho baixa, 100 para uma carga média e 150 para uma carga alta. Existem clientes e servidores em ambos os lados da VNF, gerando fluxos em ambas as direções.

Tabela 5.1: Parâmetros para a geração de tráfego sintético.

Parâmetro	**Distribuição**	**Atributo(s)**
Taxa de chegada	*Poisson*	$\lambda = 0{,}5$
Duração do tráfego	Uniforme	min = 5 s e max = 100 s
Vazão do tráfego	Exponencial	média = 1024 Kbps

[3] ⟨https://www.nsnam.org⟩

Tabela 5.2: Parâmetros para a configuração dos comutadores OpenFlow.

Parâmetro	**Comutador de redirecionamento**	**Comutador de função**	
		hardware	***software***
Tamanho da tabela de fluxo	65535	1024	8192
Capacidade de processamento	100 Gbps	2 Gbps	750 Mbps
Atraso por operação TCAM	20 us	20 us	100 us

A Tabela 5.2 descreve os parâmetros que foram usados para configurar os comutadores OpenFlow no simulador ns-3. O comutador de função de *hardware* apresenta quase três vezes mais capacidade de CPU do que os comutadores de função em *software*. Por outro lado, os comutadores de função de *software* possuem tabelas de fluxo oito vezes maiores do que os comutadores de função em *hardware*. Esses valores refletem as proporções encontradas na literatura, mas foram ajustados para a dimensão do cenário simulado. Os comutadores de redirecionamento, o controlador OpenFlow e os pares de cliente/servidor possuem recursos superdimensionados, para que não se tornem gargalos no experimento. A largura de banda de todos os enlaces da rede foi definida em 10 Gbps, evitando congestionamentos que podem ocultar o desempenho real da VNF.

Os parâmetros dos algoritmos foram definidos em $\kappa = 95\%$, $\theta_i = 95\%$, $\theta_d = 80\%$. O controlador VNF executa o Algoritmo 4.2 para realocar fluxos a cada 10 segundos e o Algoritmo 4.3 para ajustar o plano de dados a cada 15 segundos. O tempo necessário para ativar um novo comutador de função em *software* foi definido para 10 segundos, simulando o tempo de inicialização de uma nova máquina virtual.

Diferentes configurações foram simulados nos experimentos, conforme mostra a Tabela 5.3. Simulações independentes com sementes distintas foram executadas para cada configuração e os resultados a seguir mostram o valor médio obtido no intervalo estável da simulação com um nível de confiança de 99%.

Tabela 5.3: Configurações dos cenários simulados.

Configuração	**Balanceamento de carga**	**Escalabilidade do plano de dados**	**Comutador em *hardware***	**Comutadores em *software***
HW = 0; SW = 5	Não	Não	0	5
HW = 0; SW = 1–5	Não	Sim	0	1 a 5
HW = 1; SW = 5	Sim	Não	1	5
HW = 1; SW = 1–5	Sim	Sim	1	1 a 5

5.2 Resultados e discussões

A Figura 5.1 mostra o número médio de comutadores de função em *software* ativos na VNF para lidar com a carga de trabalho imposta pela rede durante o intervalo estável da simulação. Quando o mecanismo de escalabilidade do plano de dados está desabilitado (barras em cores claras), o número de comutadores de função em *software* ativos permanece fixo em cinco, conforme esperado. No entanto, quando o mecanismo de escalabilidade do plano de dados está habilitado (barras em cores escuras), podemos observar a economia de recursos em praticamente todos os casos. A exceção é para a carga alta sem o comutador de função em *hardware* (barra azul escura), onde a VNF continua usando os cinco comutadores de função em *software* para suportar a demanda elevada. Outro comportamento interessante é que um único comutador de função em *hardware* pode substituir até dois comutadores de função em *software* (considerando as barras em cores escuras no cenário de carga baixa). Isso só é possível quando os mecanismos de balanceamento de carga e escalabilidade do plano de dados estão ativos simultaneamente. Por fim, ao habilitar o mecanismo de escalabilidade do plano de dados com um comutador de função em *hardware* (barra verde escura), nenhum dos níveis de carga exige mais de quatro comutadores de função em *software*.

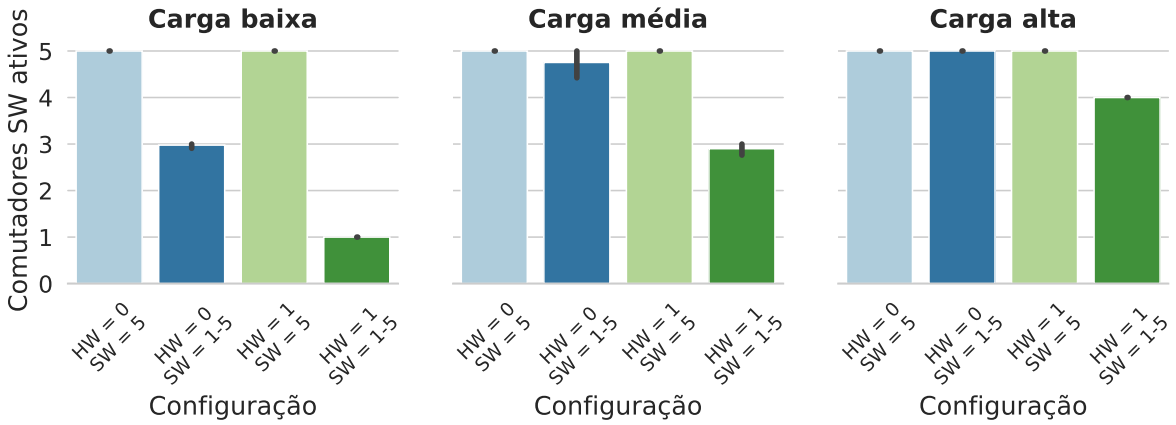


Figura 5.1: Número médio de comutadores de função em *software* ativos.

A Figura 5.2 mostra a vazão agregada média para as diferentes configurações da VNF durante o intervalo estável da simulação. Para as cargas baixa e média, os resultados são estatisticamente semelhantes entre as configurações. Em outras palavras, sob essas cargas de tráfego, a VNF pode lidar com a demanda da rede usando qualquer configuração.

Então, quando estamos usando cinco comutadores de função em *software* o tempo todo, estamos simplesmente desperdiçando recursos. No entanto, apenas as configurações usando um comutador de função em *hardware* (barras verdes) podem melhorar a vazão agregada com carga alta. Nesse caso, ambas as configurações (com ou sem escalabilidade do plano de dados) apresentam uma vazão estatisticamente semelhante. No entanto, analisando este resultado em conjunto com Figura 5.1, é possível confirmar que a VNF com o mecanismo de escalabilidade do plano de dados ativo atinge o mesmo desempenho em termos de vazão mas utiliza menos recursos.

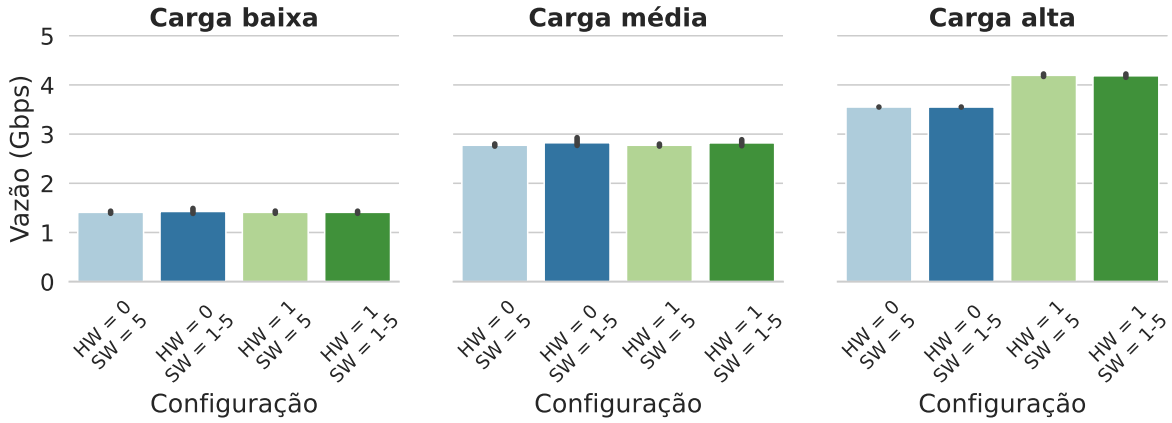
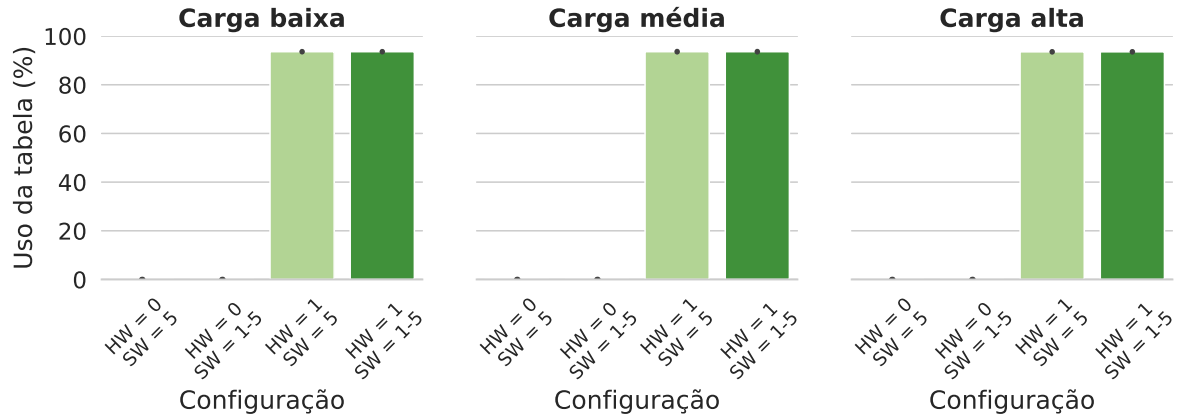
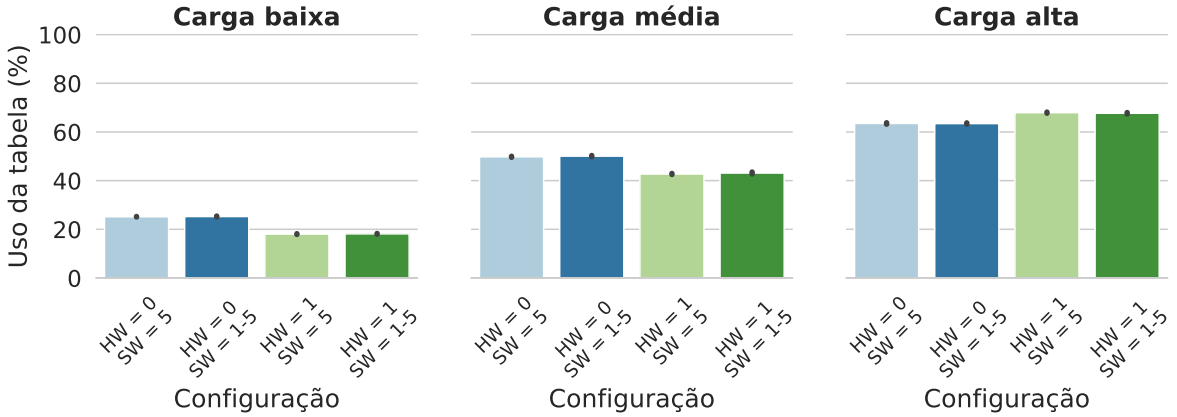


Figura 5.2: Vazão agregada média da VNF.

A Figura 5.3 mostra o uso médio das tabelas de fluxo nos comutadores de função em *hardware* e em *software*. Conforme mostra a Figura 5.3a, quando o comutador de função em *hardware* está presente (barras verdes), o mecanismo de balanceamento de carga mantém sua tabela de fluxo sempre cheia (em 95% da capacidade máxima, conforme o valor definido para o parâmetro κ). Consequentemente, como mostra a Figura 5.3b, os fluxos movidos para o comutador de função em *hardware* reduzem o uso médio das tabelas de fluxo dos comutadores de função em *software* em aproximadamente 10% nos cenários com carga baixa e média. A exceção é para o cenário com carga alta, onde os comutadores de função em *software* ficam completamente sobrecarregados nas configurações sem o comutador de função em *hardware* (barras azuis), o que aumenta a taxa de bloqueio e oculta essa diferença no indicador de uso da tabela de fluxo. A saber, nas configurações sem o comutador de função em *hardware* (barras azuis), foram observados cerca de 4% de novos fluxos bloqueados no cenário com carga média e pouco mais de 16% de novos fluxos bloqueados no cenário com carga alta.



(a) Uso da tabela de fluxo no comutador de função em *hardware*.

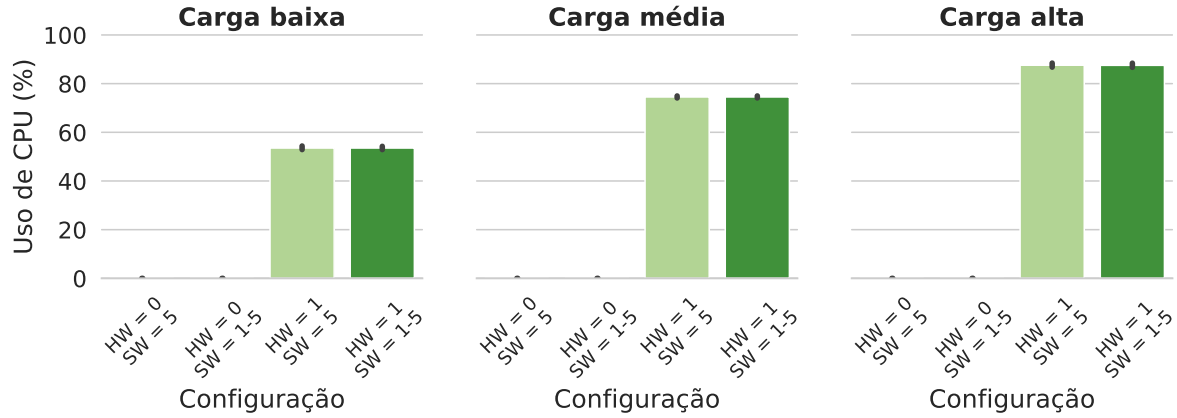


(b) Uso médio das tabelas de fluxo nos comutadores de função em *software*.

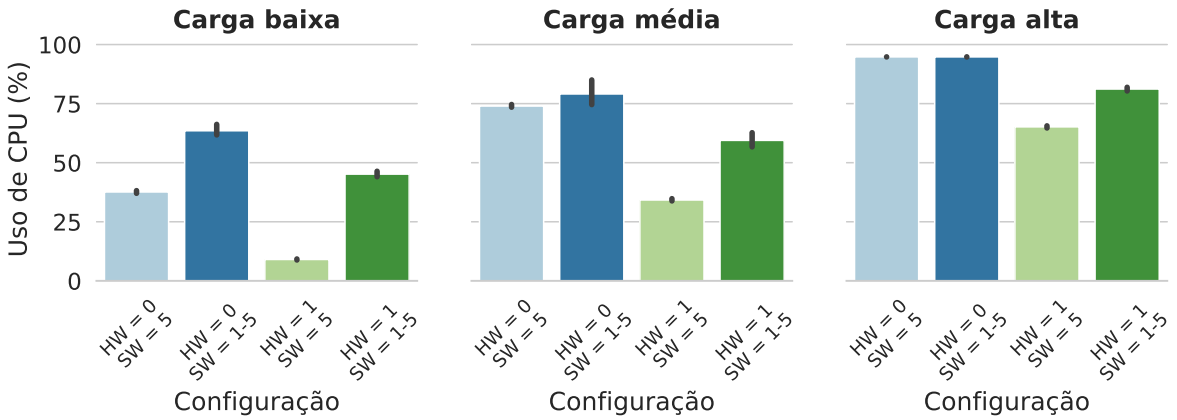
Figura 5.3: Uso médio das tabelas de fluxo nos comutadores de função.

Por fim, a Figura 5.4 apresenta o uso médio de CPU nos comutadores de função em *hardware* e em *software*. Como esperado, a utilização de CPU em todos os comutadores de função aumenta junto com a carga de rede. Claramente, quanto mais tráfego na rede, mais fluxos "elefantes" farão bom uso do alto desempenho do comutador de função em *hardware*, conforme mostra a Figura 5.4a. Como os comutadores de função em *software* possuem capacidade de processamento reduzida, o crescimento da utilização média de CPU nestes comutadores fica mais evidente na Figura 5.4b. Considerando a configuração com o comutador de função em *hardware* presente (barras verdes), o uso médio de CPU nos comutadores de função em *software* é consideravelmente menor do que na configuração sem o comutador de função em *hardware* (barras azuis). Além disso, o mecanismo de escalabilidade do plano de dados (barras em cores escuras) aumenta o uso médio de CPU dos comutadores de função em *software* quando comparado à configuração fixa de cinco comutadores de função em *software* ativos durante toda a simulação (barras em cores

claras). Com estes resultados é possível confirmar que o mecanismo de escalabilidade do plano de dados operando em conjunto com o mecanismo de balanceamento de carga fazem com que menos recursos sejam desperdiçados na VNF.



(a) Uso de CPU no comutador de função em *hardware*.



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

Figura 5.4: Uso médio de CPU nos comutadores de função.

6 Avaliação com modelo de tráfego realista

Este capítulo apresenta o segundo conjunto de testes que foi utilizado para avaliar os mecanismos propostos neste trabalho. Neste conjunto foram utilizadas aplicações com comportamento realista para a geração de tráfego. Por natureza, essas aplicações geram um tráfego agregado menor quando comparadas ao modelo sintético adotado no capítulo anterior. Mesmo assim, elas são relevantes para verificar o comportamento dos mecanismos de balanceamento e escalabilidade em situações de uso próximas do real. A Seção 6.1 descreve a metodologia de avaliação adotada enquanto a Seção 5.2 apresenta os resultados obtidos e as discussões subsequentes.

6.1 Metodologia de avaliação

A mesma topologia descrita na Seção 5.1 foi adotada neste conjunto de experimentos. Entretanto, ao invés de utilizar os geradores de tráfego sintético descritos na Tabela 5.1, foram utilizadas três aplicações realistas para avaliar os mecanismos propostos:

- **Navegação Web**. Esta aplicação modela o tráfego em rajadas de acesso à Web, um dos serviços mais utilizado da Internet. Nesta aplicação, a comunicação entre os clientes e o servidor acontece via protocolo de transferência de hipertexto (HTTP, do Inglês *Hypertext Transfer Protocol*), que roda sobre o protocolo de transporte TCP (do Inglês *Transmission Control Protocol*). Nesta aplicação, o cliente envia uma requisição ao servidor solicitando o conteúdo de uma página qualquer. Por sua vez, o servidor envia de volta ao cliente a página solicitada e também todo seu conteúdo embutido. Na sequência, o cliente aguarda um tempo aleatório (denominado tempo de leitura da página) para então proceder com uma nova requisição ao servidor. O tráfego desta aplicação foi modelado de acordo com o trabalho de (PRIES; MAGYARI; TRAN-GIA, 2012). A duração de cada sessão completa de navegação pela Web segue uma distribuição normal com média $\mu = 150$ e desvio padrão $\sigma = 30$ segundos, definida com base nas informações apresentadas em (RITWIK B, 2018).

- **Voz sobre IP**: A voz sobre o protocolo da Internet (VoIP, do Inglês *Voice over IP*) é utilizada para realizar ligações telefônicas através da Internet. A aplicação VoIP transforma os sinais de áudio analógicos em sinais digitais que são encapsulados em pacotes e posteriormente transmitidos pela rede sobre o protocolo de transporte UDP. O modelo de tráfego adotado nesta implementação é baseado na codificação G.711, que gera pacotes de 160 bytes a cada 20 milissegundos de áudio. Desta forma, a aplicação produz um tráfego constante de 64 Kbps em cada direção. A duração de cada chamada de voz segue uma distribuição de Pareto com forma $\alpha = 1.15$ e escala $\beta = 0.537$, definida com base nas informações apresentadas em (HE, 2007).

- **Vídeo em tempo real**: A transmissão de vídeo pela Internet cresce a cada dia, principalmente com a popularização de serviços como o Netflix e o YouTube. Esta aplicação implementa a transmissão de vídeo em tempo real. O modelo de tráfego utilizado é descrito em arquivos de traços que foram gerados a partir de filmes, programas de televisão e câmeras de segurança (FITZEK; REISSLEIN, 2001). O áudio e o vídeo foram codificados de acordo com o padrão MPEG-4 e as informações sobre o tamanho e frequência dos pacotes a serem transmitidos foram salvas nos arquivos. A duração de cada transmissão de vídeo segue uma distribuição normal com média $\mu = 264$ e desvio padrão $\sigma = 132$ segundos, definida com base nas informações apresentadas em (LELLA, 2014).

Neste conjunto de experimentos, a quantidade de pares cliente/servidor foi definida em 500 para gerar uma carga de trabalho baixa, 1000 para uma carga média e 1500 para uma carga alta. Este número é uma ordem de grandeza maior do que os definidos no conjunto anterior de experimentos. Isso foi necessário para aumentar a demanda agregada na rede, visto que as aplicações realistas adotadas aqui possuem vazão individual menor do que as aplicação sintéticas.

A proporção entre as aplicações instalada nos clientes e servidores foi definida em 50%, 30% e 20% para a navegação web, VoIP e transmissão de vídeo, respectivamente. Existem clientes e servidores em ambos os os lados da VNF, gerando fluxos em ambas as direções. Os demais parâmetros da simulação são os mesmos que já foram apresentados na Seção 5.1, assim como as configurações dos cenários simulados da Tabela 5.3.

6.2 Resultados e discussões

A Figura 6.1 mostra o número médio de comutadores de função em *software* ativos na VNF para lidar com a carga de trabalho imposta pela rede durante o intervalo estável da simulação. Assim como nos resultados apresentados na Figura 5.1, o comportamento está dentro do esperado quando o mecanismo de escalabilidade do plano de dados está desabilitado (barras em cores claras), já que o número de comutadores de função em *software* permanece fixo em cinco. Com o algoritmo de escalabilidade do plano de dados ativo (barras em cores escuras), podemos observar economia de recursos em todos os casos. O comportamento mais interessante é observado na carga alta, onde toda a demanda foi atendida com apenas um comutador de função em *software* e um comutador de função em *hardware* (barra verde escura). Este fato reforça a conclusão de que um único comutador de função em *hardware* pode substituir o trabalho realizado por dois ou mais comutadores de função em *software*.

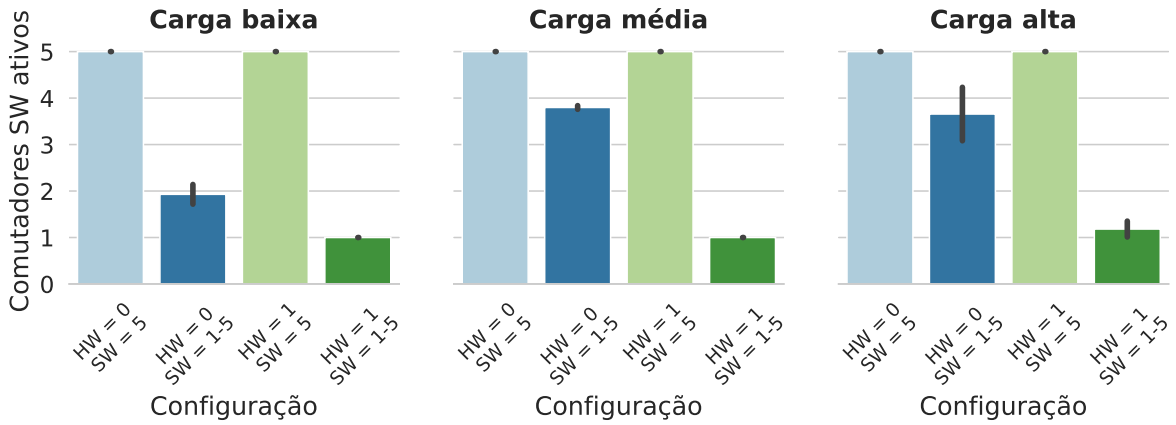


Figura 6.1: Numero médio de comutadores de função em *software* ativos.

A Figura 6.2 mostra a vazão agregada média para as diferentes configurações da VNF durante o intervalo estável da simulação. Na carga baixa, vemos um leve aumento da vazão nas configurações em que o comutador de função em *hardware* está ativo (barras verdes). Já na carga média, o resultado se inverte. Apesar de próximos, os resultados sem o comutador de função em *hardware* ativo (barras azuis) apresentam uma vazão levemente maior. Já na carga alta, temos uma vazão estatisticamente igual de três configurações, enquanto que a configuração sem o comutador de função em *hardware* e com o número de comutadores de função em *software* fixo em cinco (barra azul claro) apresentou uma

vazão maior. Esse comportamento é explicado pela presença marcante de tráfego web na VNF. Esses inúmeros fluxos utilizam o protocolo TCP, que realiza retransmissões e reordenamento de pacotes que possivelmente chegaram ao destino fora de ordem em consequência dos eventos de rebalanceamento de fluxo ativos. É importante lembrar que não é possível obter uma imagem completa do comportamento da rede analisando apenas a vazão mostrada na Figura 6.2. Devemos analisar este resultado em conjunto com a Figura 6.1, onde fica evidente que a configuração com o comutador de função em *hardware* presente e o mecanismo de escalabilidade do plano de dados ativo (barra verde escura) alcança um resultado satisfatório sem desperdiçar recursos.

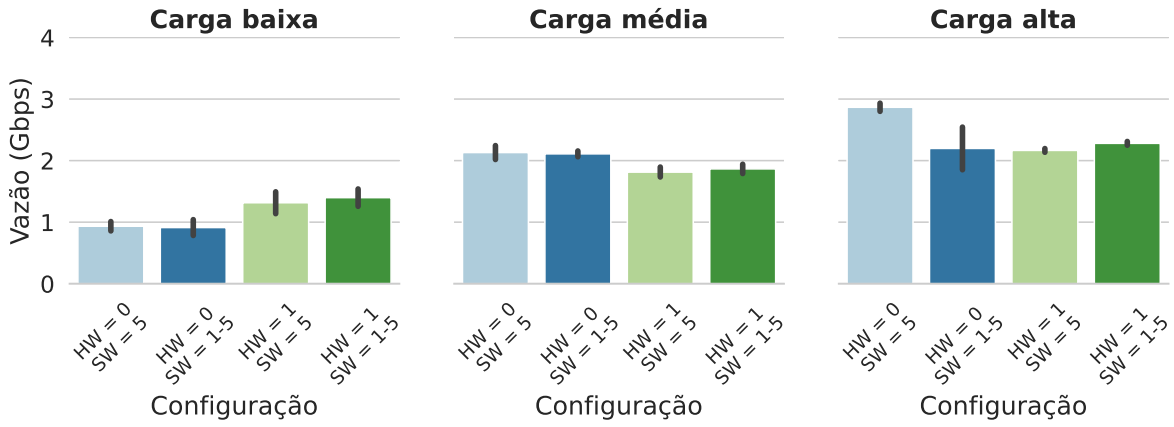
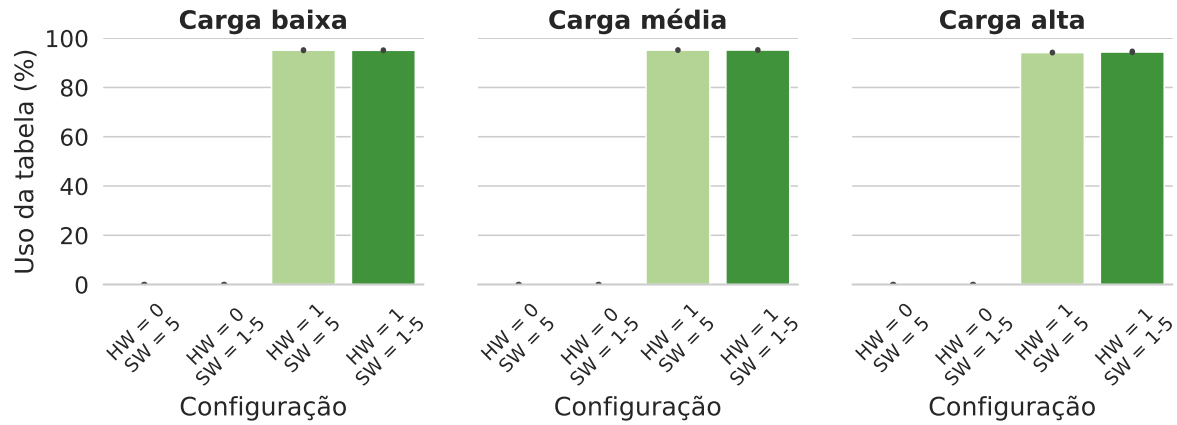
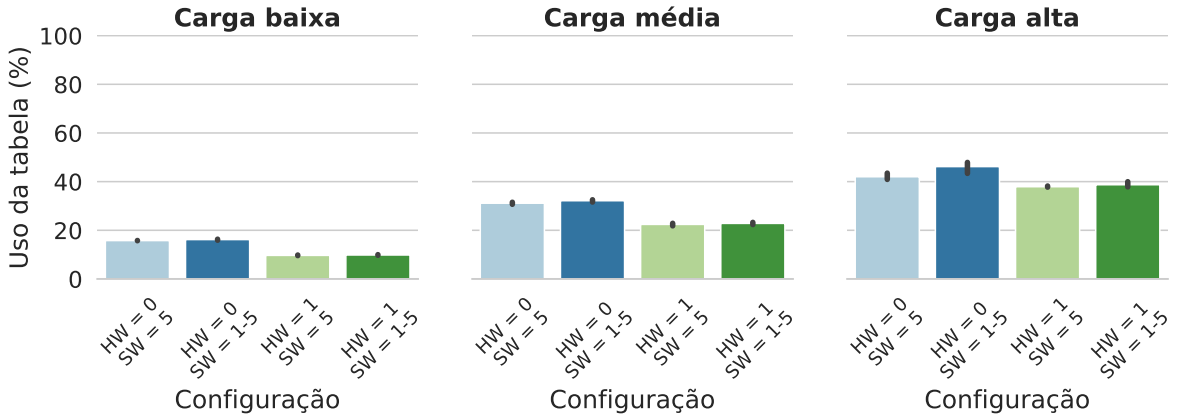


Figura 6.2: Vazão agregada média da VNF.

A Figura 6.3 mostra o uso médio das tabelas de fluxo nos comutadores de função em *hardware* e em *software*. Conforme mostra a Figura 6.3a, quando o comutador de função em *hardware* está presente (barras verdes), o mecanismo de balanceamento de carga mantém sua tabela de fluxo sempre cheia (em 95% da capacidade máxima, conforme o valor definido para o parâmetro κ). Apesar de muitos fluxos estarem ativos simultaneamente neste experimento com tráfego realista, estes fluxos também possuem uma duração média menor e, consequentemente, mais regras são movidas frequentemente dos comutadores de função em *software* para o comutador de função em *hardware*. Isso faz com que o uso médio das tabelas de fluxo dos comutadores de função em *software* seja menor do que no cenário sintético. Podemos observar também que, conforme evidenciado na Figura 6.3b, o uso médio das tabelas de fluxo nos comutadores de função em *software* é sempre menor nas configurações em que o comutador de função em *hardware* está ativo (barras verdes), conforme esperado.



(a) Uso da tabela de fluxo no comutador de função em *hardware*.

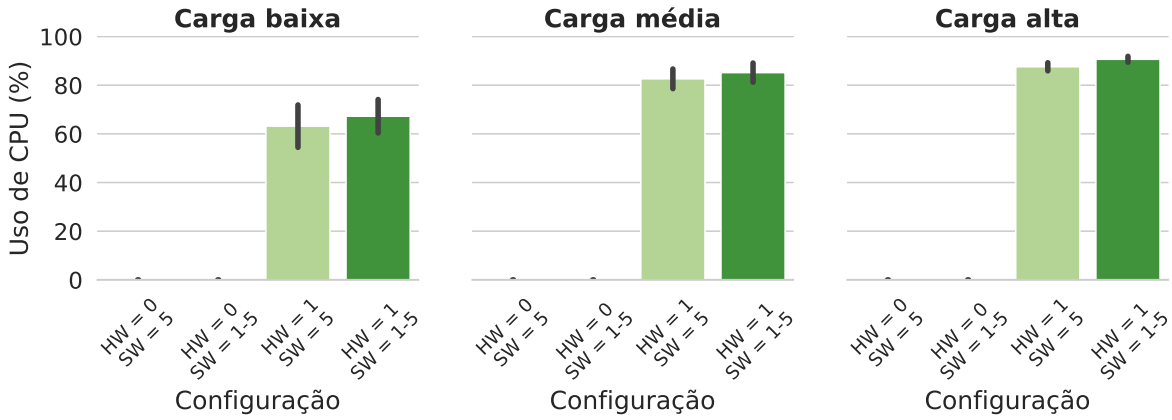


(b) Uso médio das tabelas de fluxo nos comutadores de função em *software*.

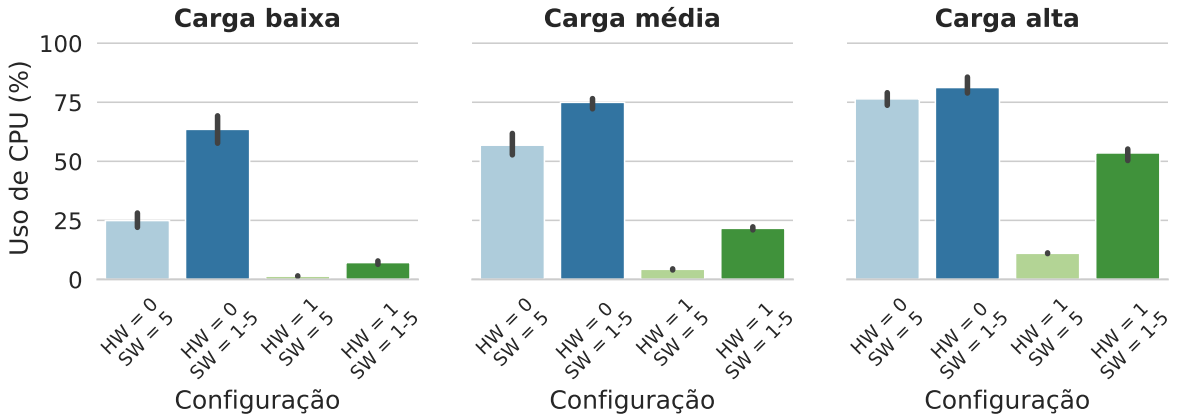
Figura 6.3: Uso médio da tabela de fluxos nos comutadores de função.

Por fim, a Figura 6.4 apresenta o uso médio de CPU nos comutadores de função em *hardware* e em *software*. Naturalmente, o uso de CPU aumenta em todos os comutadores à medida que a carga de trabalho agregada aumenta, já que mais tráfego na rede implica em maior demanda no processamento de pacotes pela VNF. O uso médio de CPU nos comutadores de função em *software* é significativamente menor nas configurações em que o comutador de função em *hardware* está ativo, conforme mostrado na Figura 6.4b. Já o comutador de função em *hardware* apresenta o uso de CPU elevado em todas as configurações, conforme exibido na Figura 6.4a. Esse é um resultado esperado, dado que o mecanismo de balanceamento de carga tende a mover regularmente os tráfegos de maior vazão individual para o comutador de função em *hardware*, explorando ao máximo sua elevada capacidade de processamento. Também vemos que o mecanismo de escalabilidade do plano de dados (barras em cores escuras) faz melhor proveito dos recursos disponíveis, aumentando o uso médio de CPU dos comutadores de função em

software quando comparado aos cenários onde o número de comutadores de função em *software* é sempre fixo em cinco (barras em cores claras).



(a) Uso de CPU no comutador de função em *hardware*.



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

Figura 6.4: Uso médio de CPU nos comutadores de função.

7 Conclusão

Neste trabalho exploramos a integração direta entre os paradigmas SDN e NFV ao propor a virtualização do plano de dados de funções de rede sem estado que realizam intenso processamento de pacotes. A solução apresentada utiliza uma coleção de regras de fluxo distribuídas sobre comutadores OpenFlow heterogêneos, de modo a aumentar a programabilidade e a escalabilidade da função virtual.

Explorar recursos distintos, muitas vezes conflitantes, em um plano de dados heterogêneo é essencial para melhorar a capacidade da VNF, elevando a taxa de transferência agregada e, principalmente, economizando no uso dos recursos da infraestrutura. Para tal, apresentamos duas contribuições: um mecanismo para o balanceamento dinâmico de carga, que move periodicamente os tráfegos de maior vazão individual para comutadores em *hardware*, de modo a explorar sua elevada capacidade de processamento; e um mecanismo de escalabilidade do plano de dados, que controla dinamicamente o número de comutadores em *software* ativos na VNF para atender às variações temporais na demanda da rede.

A avaliação conjunta desses mecanismos foi realizada através de simulações, com diferentes configurações de carga de trabalho e modelos de tráfego. Primeiramente, os mecanismos foram avaliados em um cenário de uso extremo da rede, com tráfego sintético. Os resultados iniciais confirmaram que os mecanismos podem sustentar a vazão da VNF enquanto permitem o uso racional dos recursos de virtualização. Para validar a relevância dos mecanismos em um cenário mais próximo da realidade, novas simulações foram realizadas utilizando modelos de tráfego realistas. Os resultados novamente confirmaram a economia de recursos decorrente do uso conjunto dos mecanismo. Assim, podemos concluir que o uso de comutadores em *hardware* e em *software* no cenário proposto, quando aliado aos mecanismos cientes das características do plano de dado e dos tráfegos, é essencial para permitir o uso consciente dos recursos da infraestrutura e viabilizar a implementação de serviços com qualidade.

Como trabalhos futuros, pretendemos explorar o uso destes mecanismos em outros tipos de funções de rede, com destaque para aquelas que necessitam de armazenamento de

estado no plano de dados. Além disso, pretendemos incorporar técnicas de inteligência computacional para auxiliar na tomada de decisão, possivelmente com predição de demanda da rede de modo que o mecanismo de escalabilidade de plano de dados seja mais pró-ativo.

A Fluxogramas dos mecanismos para gerenciamento da VNF

Os fluxogramas presentes neste apêndice apresentam o funcionamento dos mecanismos de gerenciamento da VNF de uma maneira complementar aos algoritmos do Capítulo 4.

A Figura A.1 apresenta o fluxograma do algoritmo de admissão de fluxos. Ele é executado de forma assíncrona, somente quando o controlador recebe do comutador uma mensagem indicando que o comutador não possui uma correspondência na sua tabela de fluxos. Então, o controlador verifica os recursos disponíveis e, caso os recursos estejam esgotados, bloqueia o fluxo. Caso contrário, o fluxo é aceito e as regras são instaladas nos comutadores de função em *software*.

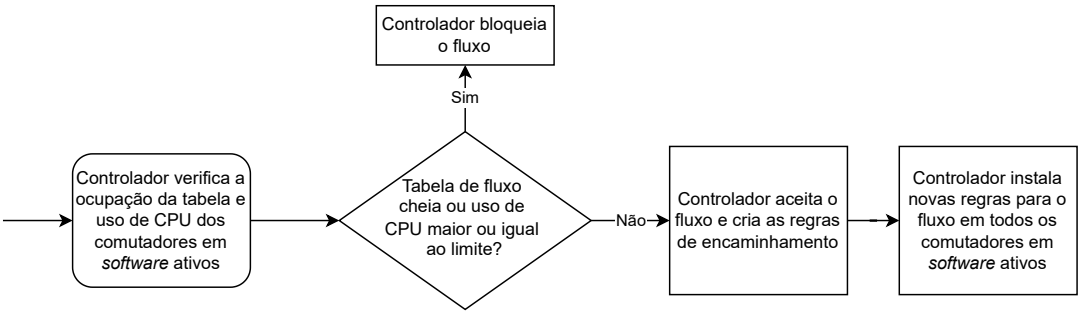


Figura A.1: Fluxograma do algoritmo de admissão de novos fluxos.

O algoritmo de balanceamento de carga da Figura A.2, executa de forma síncrona em um tempo configurável. Este algoritmo move os fluxos de maior vazão para o comutador de função em *hardware*, até que sua tabela de fluxos alcance a ocupação máxima.

Por fim, a Figura A.3 mostra o fluxograma para o algoritmo de escalabilidade, que ajusta o número de comutadores em *software* ativos para que o uso médio de CPU fique dentro dos limites definidos.

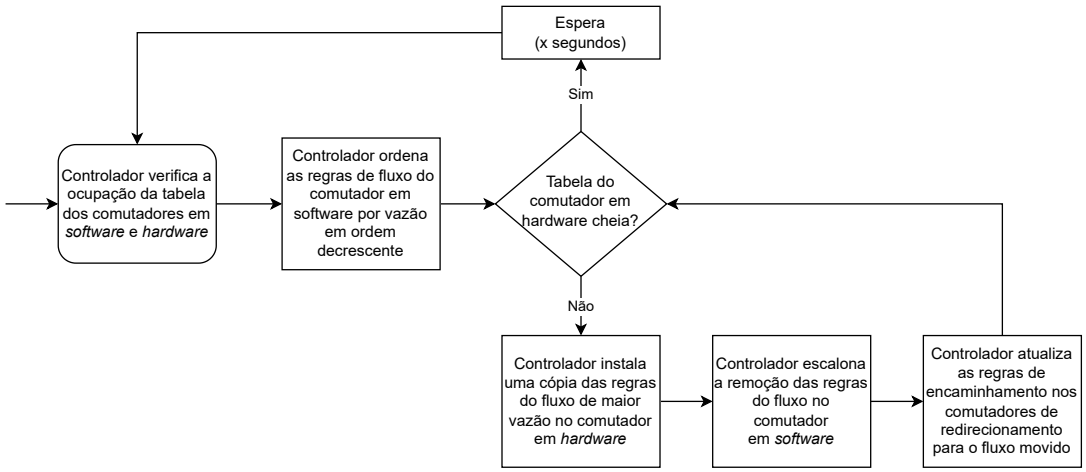


Figura A.2: Fluxograma do algoritmo de balanceamento de carga.

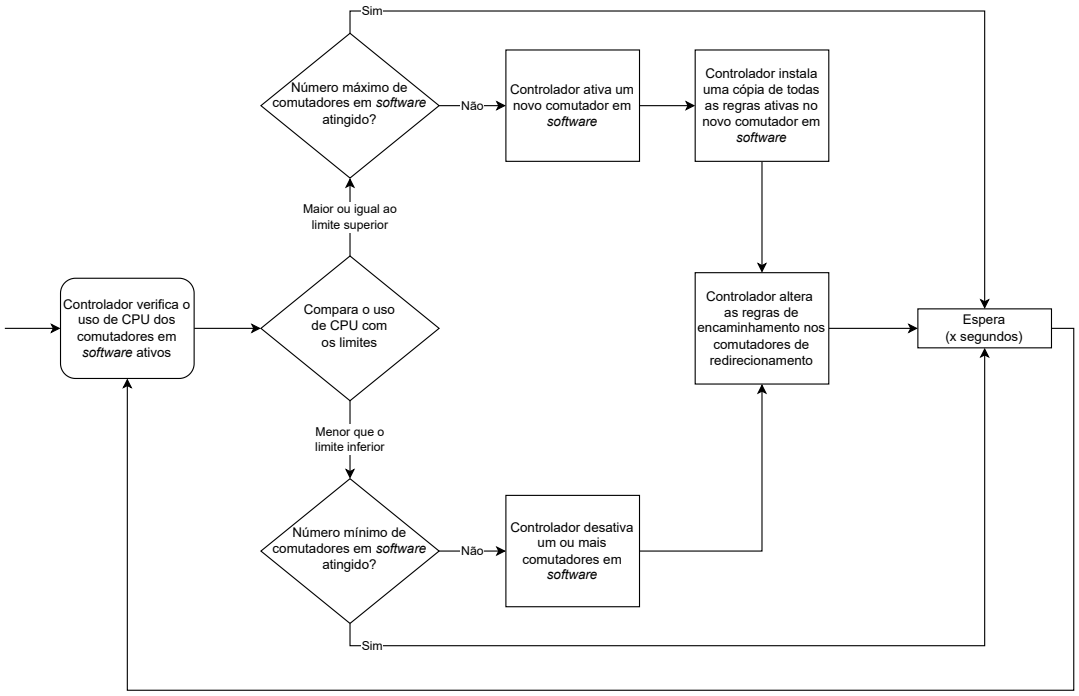


Figura A.3: Fluxograma do algoritmo de escalabilidade do plano de dados.

Bibliografia

ALAM, F.; KATIB, I.; ALZAHRANI, A. New networking era: Software defined networking. *International Journal of Advanced Research in Computer Science and Software Engineering*, v. 4, 11 2013.

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. *International Conference on Communications (ICC)*. [S.l.], 2017. p. 1–6.

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

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 CNSM*. [S.l.: s.n.], 2017.

CISCO, Inc. *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, 2021.

COSTA, L. C. et al. Performance evaluation of OpenFlow data planes. In: IFIP/IEEE. *Symposium on Integrated Network Management (IM)*. [S.l.], 2017. p. 470–475.

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.

FITZEK, F.; REISSLEIN, M. Mpeg-4 and h.263 video traces for network performance evaluation. *IEEE Network*, v. 15, n. 6, p. 40–54, 2001.

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, 2020.

HALPERN, J.; PIGNATARO, C. et al. Service function chaining (sfc) architecture. In: *RFC 7665*. [S.l.: s.n.], 2015. p. 1–32.

HAMDAN, M. et al. A comprehensive survey of load balancing techniques in software-defined network. *Journal of Network and Computer Applications*, Elsevier, v. 174, p. 1–30, 2021.

HE, Q. *Analysing the characteristics of VoIP traffic*. Tese (Doutorado) — Citeseer, 2007.

JAMALI, S.; BADIRZADEH, A.; SIAPOUSH, M. S. On the use of the genetic programming for balanced load distribution in software-defined networks. *Digital Communications and Networks*, 2019. DOI: 10.1016/j.dcan.2019.10.002.

KABLAN, M. et al. Stateless network functions: Breaking the tight coupling of state and processing. In: USENIX. *Symposium on Networked Systems Design and Implementation (NSDI)*. [S.l.], 2017. p. 97–112.

KATTA, N. et al. Cacheflow: Dependency-aware rule-caching for software-defined networks. In: ACM. *Symposium on SDN Research (SOSR)*. [S.l.], 2016. p. 1–12.

KAUR, K.; MANGAT, V.; KUMAR, K. A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture. *Computer Science Review*, v. 38, p. 1–27, 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, 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.

LELLA, A. *Comscore Releases January 2014 U.S. Online Video Rankings*. 2014. Disponível em: ⟨https://www.comscore.com/Insights/Press-Releases/2014/2/comScore-Releases-January-2014-US-Online-Video-Rankings⟩.

LEONHARDT, A. *SDN vs NFV: Understanding Their Differences, Similarities and Benefits*. 2020. Disponível em: ⟨https://blog.equinix.com/blog/2020/03/10/sdn-vs-nfv-understanding-their-differences-similarities-and-benefits/⟩.

MIJUMBI, R. et al. 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.

MONTAZEROLGHAEM, A.; YAGHMAEE, M. H.; LEON-GARCIA, A. Green cloud multimedia networking: NFV/SDN based energy-efficient resource allocation. *IEEE Transactions on Green Communications and Networking*, v. 4, p. 873–889, 2020.

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.

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.

PRIES, R.; MAGYARI, Z.; TRAN-GIA, P. An http web traffic model based on the top one million visited web pages. In: *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012*. [S.l.: s.n.], 2012. p. 133–139.

RITWIK B. *2018 Google Benchmarks: Bounce Rate & Avg. Session Duration*. 2018. Disponível em: ⟨https://www.digishuffle.com/blogs/bounce-rate-session-duration-benchmarks⟩.

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

SZALAY, M. et al. Industrial-scale stateless network functions. In: IEEE. *International Conference on Cloud Computing (CLOUD)*. [S.l.], 2019. p. 383–390.

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

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

ZHANG, J. et al. Load balancing in data center networks: A survey. *IEEE Communications Surveys & Tutorials*, v. 20, n. 3, p. 2324–2352, 2018.