UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Leonardo da Cruz Marcuzzo

UMA PLATAFORMA PARA EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE

Santa Maria, RS
2016

Leonardo da Cruz Marcuzzo

UMA PLATAFORMA PARA EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Carlos Raniery Paula dos Santos

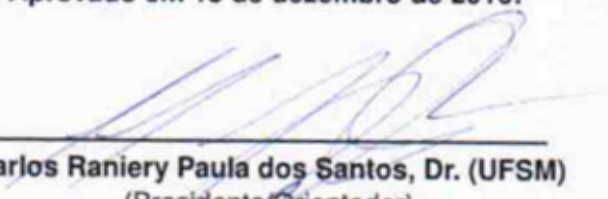
420
Santa Maria, RS
2016

Leonardo da Cruz Marcuzzo

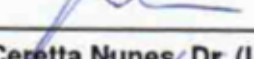
UMA PLATAFORMA PARA EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

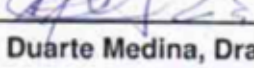
Aprovado em 15 de dezembro de 2016:



Carlos Raniery Paula dos Santos, Dr. (UFSM)
(Presidente/Orientador)



Raul Ceretta Nunes, Dr. (UFSM)



Roseclea Duarte Medina, Dra. (UFSM)

Santa Maria, RS
2016

AGRADECIMENTOS

Primeiramente agradeço aos meus pais pelo suporte durante todo o tempo do curso, principalmente nos momentos mais difíceis. Agradeço a minha irmã, que morou comigo neste último ano de curso, por me apoiar e ser uma grande amiga para conversar.

Agradeço também ao meu orientador, professor Carlos, por tirar minhas dúvidas e apresentar novas ideias, ajudando a solucionar os problemas encontrados durante o trabalho, e por revisar (várias vezes) o texto.

Agradeço também aos meus amigos de Alegrete, colegas de trabalho do INPE e do CPD, e meus colegas durante o curso.

Por fim agradeço a UFSM e aos docentes do curso de Ciência da Computação por me proporcionarem um ensino de qualidade e a banca por aceitar o convite e apresentar correções e sugestões.

RESUMO

UMA PLATAFORMA PARA EXECUÇÃO DE FUNÇÕES VIRTUALIZADAS DE REDE

AUTOR: Leonardo da Cruz Marcuzzo
ORIENTADOR: Carlos Raniery Paula dos Santos

A Virtualização de Funções de Rede (*Network Functions Virtualization* - NFV) é um conceito novo que busca utilizar técnicas de virtualização para substituir funções de rede implementadas em *hardware* especializado. NFV procura reduzir os custos e a complexidade com a implantação e gerência de funções em grandes redes de computadores. Atualmente não existe uma plataforma para a execução destas funções virtualizadas de rede que seja amplamente aceita pela comunidade científica ou que atenda todos os requisitos necessários para uma implantação efetiva de NFV. Desta forma, este trabalho têm como objetivo propor uma plataforma para a execução e gerência de funções virtualizadas de rede. Para isto, foi realizado um levantamento dos requisitos necessários para a implantação desta plataforma, seguido pela escolha de ferramentas que atendessem estes requisitos de forma efetiva. Para o desenvolvimento do trabalho, as ferramentas foram integradas tomando como base a arquitetura genérica de uma plataforma. Por fim, esta plataforma foi avaliada e comparada com as plataformas disponíveis atualmente.

Palavras-chave: Virtualização de Funções de Rede. Redes de Computadores. Virtualização.

ABSTRACT

A PLATFORM FOR RUNNING VIRTUALIZED NETWORK FUNCTIONS

AUTHOR: Leonardo da Cruz Marcuzzo
ADVISOR: Carlos Raniery Paula dos Santos

Network Functions Virtualization (NFV) is a new concept which aims to use virtualization techniques to replace network functions deployed in specialized hardware. NFV seeks to reduce Capital and Operational Expenditure (CAPEX and OPEX) and the complexity for the deployment and management of functions in large computer networks. However, there is not a widely accepted platform for the execution of these network functions that meets the required criteria defined in NFV specification. Thus, the objective of this work is to propose a plataform for executing and managing virtualized network functions. Initially, a survey was made to find the necessary requirements for the implementation of the platform, followed by choosing tools that effectively met these requirements. In the development of the work, the tools were integrated based on the generic architecture of a NFV platform. Finally, the platform has been evaluated and compared with other platforms currently available.

Keywords: Network Functions Virtualization. Computer Networks. Virtualization.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a Internet tornou-se uma ferramenta fundamental tanto para grandes empresas quanto para usuários de todo o mundo. O surgimento de novas tecnologias e serviços (*e.g.*, *smartphones*, *homebanking* e redes sociais) contribuiu para o aumento da quantidade de usuários de 400 milhões em 2000 para 3,2 bilhões em 2015 (ITU, 2016). Apesar de seu enorme sucesso, sua arquitetura apresenta algumas limitações, já que exige a interconexão entre vários *Autonomous Systems* (*i.e.*, redes sobre o controle de um único operador), administrados por provedores que muitas vezes competem entre si. Para que possa haver uma modificação nesta arquitetura é necessário uma cooperação e consenso entre estes provedores, o que ocasiona um investimento de recursos e pessoal sem um retorno significativo. O crescimento da Internet também mostra as deficiências na sua arquitetura, como vulnerabilidades e falta de suporte a QoS (TAYLOR; TURNER, 2004). Embora existam propostas para a solução destes problemas, estas não são completamente efetivas e por sua vez também introduzem alguns problemas, o que torna a Internet cada vez mais complexa. A utilização de *middleboxes* (*i.e.*, hardware especializado para execução de funções de rede) para o processamento de pacotes também ocasiona problemas, pois apesar de o protocolo IP ter sido desenvolvido para ser extensível, pacotes utilizando estas extensões possuem um custo maior de processamento ou até mesmo não são suportados, fazendo com que seu uso na Internet não seja recomendado (FONSECA et al., 2005). Devido a estes motivos, é possível dizer que a Internet está ossificada, já que o núcleo da rede permanece sem modificações significativas desde 1993 (HANDLEY, 2006). Esta ossificação pode ser vista mais claramente na gradativa implantação do novo protoco IP, o IPv6. Desde que foi definido como um padrão, em 1998, o IPv6 (DEERING; HINDEN, 1998) tem encontrado dificuldades para ser implantado, pois os *middleboxes* e equipamentos dos usuários que compõe a rede devem suportá-lo e serem eficientes em seu processamento.

Para tentar resolver o problema da ossificação, uma das alternativas é a virtualização de redes, onde a idéia geral é utilizar a infraestrutura física para a criação de múltiplas redes virtuais, que podem ter protocolos e arquiteturas próprias, de forma independente (ANDERSON et al., 2005). Dentro da área de virtualização de redes, o paradigma de Redes Definidas por Software (*Software Defined Networking* - SDN) têm recebido bastante atenção de pesquisadores por possibilitar uma arquitetura dinâmica, gerenciável e adaptável. Em SDN, o plano de dados é separado do plano de controle, e a configuração do encaminhamento de pacotes pode ser feita de forma centralizada, através de um controlador, transformando estas redes em plataformas flexíveis e programáveis (FOUNDATION, 2016). Apesar disso, o problema da ossificação não é inteiramente resolvido, já que a implementação de funções e serviços de rede mais avançados utilizando SDN (*e.g.*, *statefull*

firewalls e DPI) é difícil ou até impossível, ainda se fazendo necessário o uso de *middleboxes*. Desta forma, o conceito de Virtualização de Funções de Rede (*Network Functions Virtualization* - NFV) busca utilizar tecnologias de virtualização já existentes para a implantação destas funções e serviços.

Em NFV, a idéia principal é desacoplar as funções de rede do *hardware* dedicado, de modo que os serviços sejam desenvolvidos inteiramente em *software* e executados através da utilização de plataformas de virtualização (*i.e.*, máquinas virtuais), que por sua vez podem ser executadas em servidores de uso geral (ETSI, 2012). Isto traz uma série de vantagens tanto para provedores quanto para desenvolvedores, pois o gasto em equipamentos e manutenção é reduzido, a infraestrutura física pode ser compartilhada entre diferentes clientes, novos serviços podem ser implantados e atualizados de maneira mais ágil, além de poderem ser escalados conforme a necessidade do operador, otimizando a utilização de recursos (RAO, 2014). Contudo, existem alguns desafios que devem ser resolvidos antes que NFV possa ser utilizada efetivamente. Por exemplo, tais sistemas virtualizados devem apresentar um desempenho semelhante ao de *middleboxes* tradicionais, sua implantação deve considerar ambientes onde as soluções convencionais continuem existindo e, finalmente, devem apresentar métodos facilitados de orquestração e gerenciamento. Embora SDN e NFV não sejam dependentes entre si, sua utilização em conjunto cria vantagens para ambas tecnologias. Em suma, é possível dizer que, enquanto SDN trata a complexidade crescente do gerenciamento de redes através de centralização e programabilidade para atingir os resultados desejados, NFV transfere os serviços de rede normalmente associados com hardware especializado para uma plataforma virtualizada (SYSTEMS, 2014).

O paradigma de NFV também define que a plataforma virtualizada precisa atender a alguns requisitos específicos (*e.g.*, portabilidade, alto desempenho e simplicidade) para ser usada de maneira efetiva. Diferentemente de SDN, em NFV não existe uma plataforma amplamente adotada pela comunidade científica e, embora existam alternativas, estas ainda são limitadas. Das plataformas disponíveis atualmente, destacam-se o ClickOS (MARTINS et al., 2014) e o OPNFV (*Open Platform for NFV*) (PRICE; RIVERA, 2012) por possibilitarem a criação de múltiplas funções de rede consumindo poucos recursos computacionais. Contudo, ambas plataformas exigem modificações no hipervisor, o que afeta sua portabilidade, seu desempenho é degradado para pacotes pequenos ou quando funções devem ser utilizadas em conjunto, e sua capacidade de gerência é limitada.

Visto que as plataformas disponíveis não atendem completamente os requisitos necessários, este trabalho têm como objetivo principal criar uma plataforma onde as funções virtualizadas de rede sejam executadas e que apresente como características simplicidade, portabilidade, modularidade, alto desempenho e facilidade de gerência. Para isso, serão avaliados componentes que, integrados, formem uma solução completa que possa ser implantada, gerenciada e escalada facilmente por seus usuários.

Assim, o capítulo dois apresenta uma revisão de literatura mais aprofundada sobre o conceito de NFV, como sua arquitetura, vantagens e desvantagens e desafios que impedem sua ampla adoção. No capitulo três, serão apresentadas opções de ferramentas que constituem as partes necessárias para a criação da plataforma, com uma breve descrição de sua função e suas vantagens e desvantagens, bem como as plataformas existentes atualmente. Já no capítulo quatro, é feita inicialmente a apresentação de uma arquitetura genérica e de que forma esta arquitetura foi adaptada para a plataforma desenvolvida, além do desenvolvimento da plataforma e da interface de gerência e o seu funcionamento. No capítulo cinco, serão definidos os critérios de comparação entre a plataforma desenvolvida e outras já existentes; o cenário de testes e os resultados obtidos ao avaliar as plataformas. Por fim, no capítulo seis serão apresentadas e discutidas considerações sobre o trabalho, bem como os benefícios trazidos pela plataforma desenvolvida, conhecimento adquirido e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo geral de NFV é utilizar técnicas de virtualização já existentes no mercado para consolidar os mais variados tipos de equipamentos de rede em servidores de virtualização, comutadores e sistemas de armazenamento (ETSI, 2012). Isto se faz necessário pois atualmente as redes de grandes porte fazem ampla utilização de *middleboxes*. Um *middlebox* pode ser definido como um equipamento especializado que realiza funções mais avançadas do que o simples encaminhamento de pacotes (*e.g., firewall*, balanceamento de carga e sistemas de detecção de intrusão). Embora sua utilização seja justificada por seus benefícios com relação a segurança, desempenho e redução de uso de banda (*e.g.*, aceleradores WAN), existem desvantagens associadas com a utilização destes equipamentos. Por exemplo, os custos para a implantação de novas funcionalidades de rede (*e.g.*, espaço, energia e equipamentos), o ciclo de vida curto, sua complexidade de gerenciamento e escalabilidade são alguns dos problemas mais comuns (SHERRY et al., 2012).

Dado os problemas que surgem com a utilização de *middleboxes*, um consórcio de grandes empresas de telecomunicação estabeleceu em 2012, em conjunto com o Instituto de Padrões de Telecomunicação Europeu (*European Telecommunications Standards Institute* - ETSI) o Grupo de Padrões da Indústria para Virtualização de Funções de Rede (*Network Functions Virtualization Industry Standards Group* - ISG NFV). O ISG NFV é responsável por publicar as especificações de NFV, como a sua arquitetura, métricas para qualidade de serviço, casos de uso, *framework* de gerência e orquestração e requisitos para sua implantação. Definindo um pequeno conjunto de padrões abertos para as plataformas, ao contrário de sistemas executando em *hardware* proprietário, o grupo acredita que NFV possa ser aplicado para ambos os planos de controle e dados, e em diferentes infraestruturas (*e.g.*, redes fixa e móveis) (STALLINGS, 2015).

2.1 ARQUITETURA NFV

De acordo com as especificações do NFV ISG (ETSI, 2012), a arquitetura de NFV é dividida em três blocos distintos, sendo eles a Infraestrutura NFV (*NFV Infrastructure* - NFVI), o *framework* de Gerência e Orquestração (*NFV Management and Orchestration* - NFV MANO) e, por fim, as Funções Virtualizadas de Rede (*Virtual Network Functions* - VNF). Como pode ser visto na Figura 2.1, a NFVI fornece os recursos físicos e virtuais que são utilizados pelas VNFs, que por sua vez são responsáveis por executar as funções virtualizadas, enquanto que o *framework* NFV MANO deve gerenciar tanto a infraestrutura quanto as funções virtualizadas. A descrição detalhada destes blocos é apresentada a

seguir.

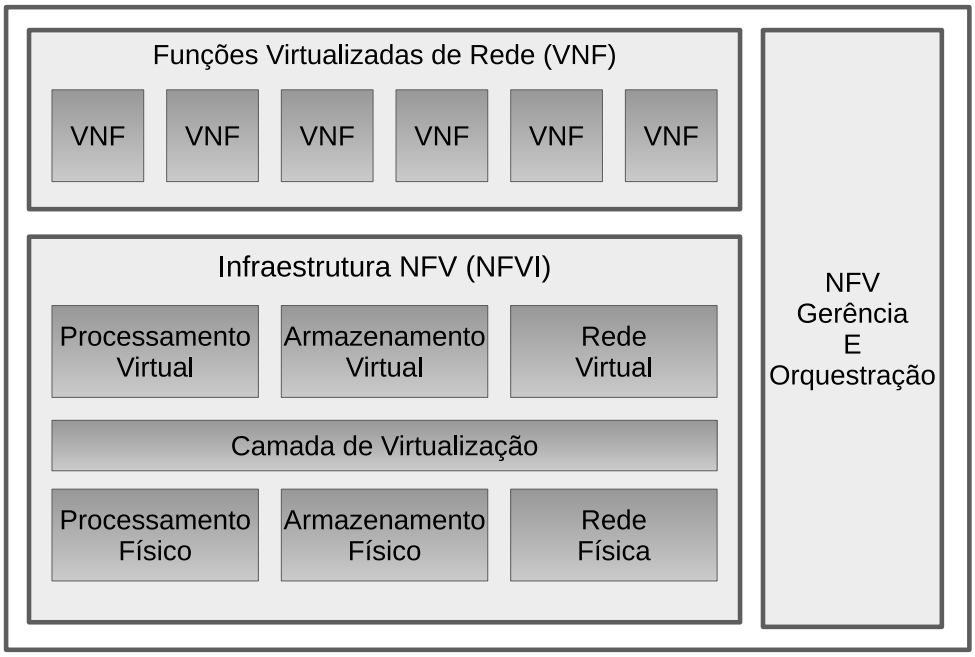


Figura 2.1 – Arquitetura de alto nível de NFV

Fonte: Adaptado de (ETSI, 2012).

- **Infraestrutura NFV (NFVI):** A NFVI pode ser definida como todos os recursos, tanto de *hardware* como de *software*, que compõem o ambiente onde as VNFs são implantadas, gerenciadas e executadas (NFV, 2014). A infraestrutura física é composta de equipamentos comerciais amplamente disponíveis (*e.g.*, servidores, *storages* e comutadores), e deve fornecer recursos de processamento, armazenamento e conectividade para as VNFs através de uma camada de virtualização. Em NFV, os recursos de rede são diferenciados em dois tipos: NFVI-PoP, que faz a interconexão dos recursos dentro do ponto de presença, e redes de transporte, que interconectam os NFVI-PoP com outras redes. A camada de virtualização deve abstrair, particionar e isolar (entre diferentes usuários) os recursos e possibilitar a utilização da infraestrutura física pelas plataformas que implementam VNFs (através de uma camada de abstração de *hardware*), de modo que a implementação das VNFs não seja específica ao *hardware* onde será executada.

- **Funções Virtualizadas de Rede (VNF):** Uma função virtualizada de rede (*Virtual Network Function* - VNF) pode ser definida como uma implementação virtual (*i.e.*, totalmente implementada em software) de funções tradicionais de rede (*e.g.*, *gateway*, *firewall*, VPNs e DPI). Uma VNF possui um propósito e interfaces de entrada e saída bem definidos na infraestrutura da rede e, internamente, são compostas de um ou mais componentes interconectados (*e.g., classificadores, contadores, filas*) que, em

conjunto, constituem uma função. Estas interconexões e componentes internos da VNF não são visíveis para a infraestrutura externa, que considera toda a VNF como um único bloco funcional (ISG, 2013). Apesar disso, estes blocos podem ser interconectados com outros, utilizando o conceito de grafos de encaminhamento, que será definido mais a frente.

- **Gerência e Orquestração de NFV (NFV MANO):** Tem por objetivo a gerência e a orquestração de todo o ambiente NFV. Por ser uma tarefa complexa, este bloco pode ser subdividido em Gerenciamento de Infraestrutura Virtualizada(*Virtualized Infrastructure Manager* - VIM), Gerenciamento de Funções Virtualizadas de Rede (*Virtual Network Function Manager* - VNFM) e, por fim, Orquestrador NFV (*NFV Orchestrator* - NFVO) (NFV, 2014). O VIM é responsável por todas as funções necessárias para o controle e gerência da interação entre as VNFs e os recursos utilizado por elas. Uma instância do VIM é responsável por gerenciar todos os recursos físicos e suas abstrações dentro do domínio operado, que pode consistir de recursos de um ou múltiplos NFVI-PoP, ou apenas uma parte de um NFVI-PoP. O VNFM, por sua vez, é responsável pela instanciação, configuração, atualização e ciclo de vida das VNFs. Pode ser implementado como uma única instância para o controle de todas VNFs dentro de um domínio ou para o controle individual de VNFs. O NFVO é responsável pela orquestração dos recursos e serviços de rede. Isto quer dizer que serviços compostos por múltiplas VNFs devem ser controlados por este módulo, assim como a instanciação de VNFMs. Também deve ter suporte ao controle de diferentes VIMs, atingindo assim uma visão global do ambiente.

2.2 GRAFOS DE ENCAMINHAMENTO

Em NFV, cada uma das VNFs possui uma funcionalidade específica e limitada. No entanto, várias funções podem ser encadeadas de modo que um nova funcionalidade seja criada. Por exemplo, um fluxo que têm como destino final um servidor Web pode passar por uma série de VNFs conectadas em cadeia (*e.g., firewall*, NAT e balanceador de carga) para que possa finalmente chegar ao seu destino. Esse encadeamento das funções virtualizadas é chamado de *service chaining* ou grafos de encaminhamento (*forwarding graphs* - FG), e pode ser utilizado para criar serviços de redes mais avançados, compostos por funções menores. Um FG permite que este encadeamento seja feito de maneira flexível, de modo que fluxos diferentes podem percorrer caminhos diferentes, como ilustrado na Figura 2.2, onde fluxos com diferentes conteúdos são processados por funções diferentes. É importante notar que estes grafos de encaminhamento podem possuir tanto funções virtualizadas quanto funções físicas, apresentando assim uma forma de facilitar na transição

de uma arquitetura de rede tradicional para NFV. Um modelo é utilizado para descrever as características do FG (*e.g.,* funções virtualizadas, interconexões, infraestrutura física) para o bloco de gerência, que por sua vez é responsável por monitorar as VNFs e conexões utilizadas para que os requisitos dos serviços sejam atendidos.

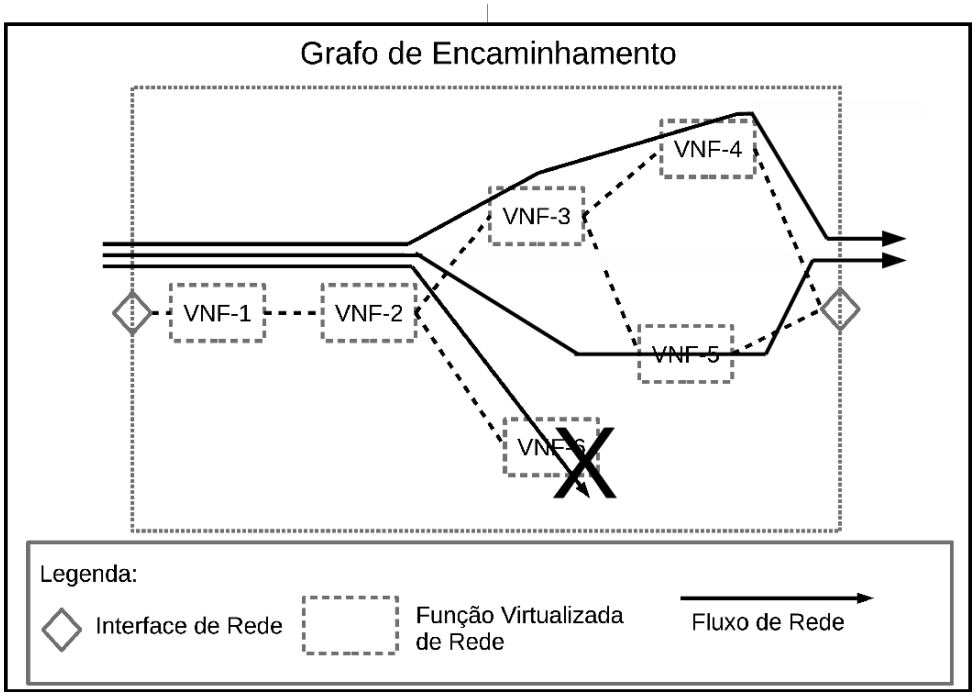


Figura 2.2 – Exemplo de um *forwarding graph* com múltiplos caminhos

Fonte: Adaptado de (NFV, 2013).

O conceito de FG é análogo ao de conexões físicas entre *middleboxes*, apesar de trazer uma série de vantagens como, por exemplo, a otimização dos recursos utilizados, a agilidade em implantar atualizações nas VNFs, a expressividade do modelo que descreve o FG e a facilidade de instanciar e conectar novas funções virtualizadas. Contudo, a utilização efetiva de grafos de encaminhamento é altamente dependente das VNFs, que devem ser compatíveis entre si, e da topologia da rede. A utilização de SDN para otimizar a topologia da rede com relação aos serviços implementados com FG ajuda a mitigar problemas como o desempenho e configuração das interconexões.

2.3 BENEFÍCIOS E REQUISITOS

A utilização de NFV traz benefícios principalmente para os provedores de telecomunicações mas também para usuários, desde que seja implantada de forma correta. Estes benefícios podem contribuir para uma mudança dramática nas operações de rede atuais (ETSI, 2012). Alguns dos benefícios mais importantes proporcionados por NFV são:

- **Custos:** Um dos principais benefícios da utilização de NFV é a redução de custos. A consolidação de funções virtualizadas de rede em servidores de virtualização contribui para uma dramática redução de custos operacionais e de capital, como por exemplo custos relacionados ao consumo de energia, espaço e equipamentos;

- **Inovação:** Novos serviços, por serem implantados em *software*, podem ser disponibilizados mais rapidamente, o que facilita a adaptação dos provedores a novos requisitos, aumenta o retorno do investimento e possibilita que novos serviços possam ser testados em paralelo com os que já estão implantados;

- **Compartilhamento de recursos:** A infraestrutura física pode ser compartilhada entre múltiplos usuários de forma isolada, o que permite que operadores compartilhem recursos entre diferentes serviços e para diferentes aplicações;

- **Escalabilidade:** Serviços podem ser rapidamente escalados para atender a demanda dos usuários, sem que haja necessidade de alterar a infraestrutura física, resultando em agilidade na escalação e otimização dos recursos;

- **Interoperabilidade:** A utilização de padrões abertos para a definição das interfaces faz com que VNFs desenvolvidas por empresas diferentes possam ser utilizadas em conjunto e sob uma mesma plataforma.

Contudo, para que os serviços de NFV possam ser utilizados de maneira eficaz, uma série de requisitos e desafios técnicos devem ser solucionados, de forma a facilitar a operabilidade em diferentes ambientes e a evolução para redes completamente virtualizadas (ISG, 2013). Assim, os requisitos e desafios mais relevantes são:

- **Portabilidade:** Para que a execução de VNFs de diferentes empresas em uma variedade de plataformas padronizadas seja possível, é necessário definir uma interface unificada que claramente separa as instâncias das VNFs de sua infraestrutura física, o que pode ser atingido ao utilizar-se virtualização;

- **Desempenho:** Por NFV ser implantada em equipamentos e tecnologias padronizadas (*i.e.,* sem otimizações encontradas em hardware proprietário), é possível que haja uma perda de desempenho com relação a *middleboxes*. O desafio deve ser mitigar a degradação de desempenho o máximo possível, utilizando para isto tecnologias e hipervisores apropriados;

- **Estabilidade da Rede:** Deve se garantir que a rede irá manter-se estável mesmo quando exista um grande número de funções virtualizadas sendo gerenciadas ou instanciadas. Este requisito é particularmente importante durante a reconfiguração da infraestrutura;

- **Integração:** Operadores de rede devem ser capaz de utilizarem em conjunto servidores, hipervisores e funções virtualizadas de rede de diferentes fornecedores, de modo que isto não incorra custos de desempenho. O ecossistema deve oferecer serviços de integração e configuração, evitando a dependência de um fornecedor específico;

- **Gerência e Orquestração:** Uma arquitetura consistente para gerência e orquestração se faz necessária. A flexibilidade de NFV apresenta uma oportunidade para que interfaces padronizadas de gerência e orquestração possam ser desenvolvidas, de forma que diferentes VNFs podem ser configuradas por um *framework* comum a todas.

3 *NFV ENABLERS*

Tecnologias e ferramentas que contribuem de alguma forma para o desenvolvimento e implantação de NFV são chamados de *NFV Enablers*. Podem ser considerados *enablers* servidores de virtualização, hipervisores, comutadores virtuais, *frameworks* para aceleração de pacotes e para criação e implantação das funções virtualizadas de rede, APIs para automação de gerência e instanciação de máquinas virtuais e também para gerência do plano de dados de rede (*e.g., OpenFlow*) (ETSI, 2012).

Dado que cada umas destas tecnologias atendem uma parte diferente dos requisitos para a implantação de NFV, é possível utilizá-las em conjunto, de modo que uma gama maior de requisitos possam ser atendidos. Assim, este capítulo é dedicado a investigar estas ferramentas e tecnologias, as quais podem ser utilizadas para atingir os objetivos da plataforma proposta. Foram identificadas quatro categorias onde estas ferramentas podem ser divididas, sendo elas sistemas operacionais, *frameworks* para aceleração de pacotes, *frameworks* para criação e implantação de funções virtualizadas de rede e, finalmente, tecnologias que possibilitem a gerência das outras ferramentas.

3.1 FERRAMENTAS

Esta seção apresenta ferramentas que se encaixam em cada uma das categorias, com uma breve descrição sobre cada uma e quais os requisitos definidos na especificação de NFV são atendidos por elas.

3.1.1 Sistema Operacional

Sistemas operacionais tradicionais são projetados para serem robustos, utilizados por múltiplos usuários e devem realizar várias tarefas ao mesmo tempo com os mais variados objetivos. Além disso, possibilitam suporte a diversos dispositivos de *hardware*, que por sua vez precisam de *drivers* específicos.

Isto faz com que estes sistemas consumam uma grande quantidade de recursos (*e.g.*, disco, memória e processamento) para que possam funcionar corretamente. Quando estes sistemas tradicionais são utilizados de forma virtualizada , muitas destas funções não são necessárias como, por exemplo, a utilização por vários usuários e a implementação de drivers diversos (PAVLICEK, 2016).

Containers são um método de virtualização onde o sistema hospedeiro aloca uma quantia de recursos para um certo processo ou função e o isola do resto do sistema opera-

cional. Assim, a função ainda pode executar em conjunto com o sistema hospedeiro mas utilizando apenas os recursos que foram disponibilizados para ela. Por não existir uma camada de abstração de *hardware* entre a função e o hospedeiro, o desempenho costuma ser melhor do que sistemas virtualizados utilizando paravirtualização. No entanto, a utilização de *containers* não satisfaz o requisito de flexibilidade, pois a função virtualizada deve suportar o sistema operacional e o *hardware*. O requisito de portabilidade também não é satisfeito, pois para mover um *container* de um servidor de virtualização para outro, é necessário que possuam um *hardware* e sistema operacional compatível (EVENS, 2015).

Assim, o conceito de *Unikernels* apresenta um paradigma diferente destes sistemas, com foco na simplicidade, escalabilidade, segurança e desempenho. Em um *Unikernel*, o *kernel* é compilado em uma biblioteca conectada diretamente ao binário da aplicação, com o único propósito de prover as funções necessárias para a execução do aplicativo e sua comunicação com o hipervisor, como pode ser visto na Figura 3.1, criando assim uma máquina virtual especializada, isolada e segura (MADHAVAPEDDY et al., 2013).

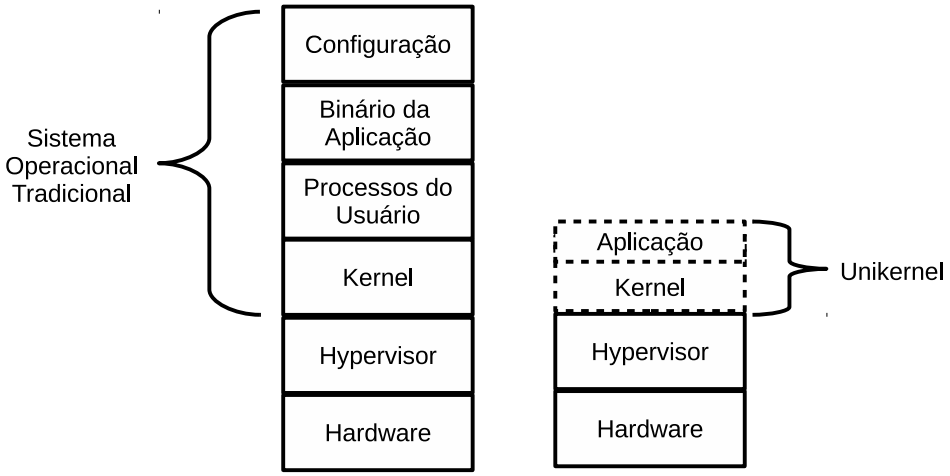


Figura 3.1 – Comparação entre sistemas operacionais tradicionais e *Unikernels*

Fonte: do Autor.

Unikernels satisfazem os requisitos de portabilidade e escalabilidade, por possuírem reduzida utilização de recursos, segurança, por serem isolados das outras máquinas virtuais, e simplicidade, já que são sistemas relativamente simples e que funcionam em arquiteturas de virtualização amplamente utilizadas.

A seguir serão apresentados alguns exemplos de *unikernels* que foram avaliadaos para serem utilizados no desenvolvimento da plataforma.

- **MiniOS:** O MiniOS (HYPERVISOR, 2015) é um *kernel* minimalista distribuído em conjunto com o código fonte do hipervisor Xen e fornece um escalonador cooperativo, gerência de interrupções e memória, *drivers* paravirtualizados para comunicação com o hipervisor e um único espaço de endereçamento de memória. Inicialmente desenvolvido como um *kernel* de exemplo para a demonstração de paravirtualização

no Xen, foi extendido para ser utilizado na criação de *stub domains*, que são máquinas virtuais executando componentes auxiliares do Xen. Através da utilização de *grant tables*, seu espaço de memória pode ser compartilhado com o hipervisor, diminuindo assim a latência e custo de recursos normalmente associados com a virtualização.

- **IncludeOS:** O IncludeOS (BRATTERUD et al., 2015) é um sistema operacional dedicado para a execução de aplicativos escritos em C++. No código fonte do programa que será executado no sistema, são adicionados uma biblioteca e funções simples que podem ser chamadas para controlar o programa. Este código, através de um compilador especial, gera uma imagem que contêm, além do programa em si, métodos para a inicialização do *kernel* e de seus subsistemas que, por sua vez, fornecem as funcionalidades necessárias para que o programa possa ser executado em diversos hipervisores (*e.g.*, Xen, KVM, Hyper-V). Apenas as partes necessárias para a execução do aplicativo são compiladas junto, de forma a otimizar ainda mais a utilização de recursos, o tempo de *boot* e o overhead das chamadas de sistema.

- **OSv:** O OSv (KIVITY et al., 2014) é um sistema operacional capaz de executar aplicações escritas em diversas linguagens com pequenas modificações (*e.g.*, JAVA, C++ e Ruby), utilizando o sistema de arquivos ZFS e com suporte a *threads*. Possui um módulo de gerência integrado ao *kernel*, com interfaces Web e REST. Por ser compatível com a ABI do Linux, aplicações desenvolvidas necessitam apenas ser compiladas como bibliotecas compartilhadas (*shared libraries*), condição esta imposta pelo fato do OSv utilizar um único espaço de endereçamento. Já para a utilização de funções mais avançadas do *kernel*, como sua pilha TCP/IP otimizadada, são necessárias modificações mais profundas nas aplicações. Seu tempo de *boot* é inferior a um segundo e é capaz de executar na maioria dos hipervisores disponíveis atualmente (*e.g.*, Xen, KVM e Hyper-V).

3.1.2 *Frameworks* para aceleração de pacotes

Apesar de *Unikernels* serem otimizados para execução em ambientes virtualizados, sua pilha de rede, assim como a de sistemas tradicionais, não apresenta o desempenho necessário para usufruir totalmente de interfaces de rede mais modernas (*e.g.*, 10Gb, 40Gb e 100Gb) ou, quando possuem este desempenho, requerem modificações extensivas na arquitetura das aplicações. Devido a isto, *frameworks* para a aceleração de pacotes têm sido apresentados como uma alternativa. A função destes *frameworks* é basicamente substituir a pilha de rede dos sistemas, com um foco na otimização dos recursos utilizados

para o encaminhamento e processamento de pacotes. Dentre as técnicas utilizadas por estes *frameworks* para atingir um alto desempenho no processamento de pacotes estão a pré-alocação de *buffers* de memória para os pacotes, o uso de *pooling* (*i.e.*, o sistema verifica a interface e pré-aloca na memória os pacotes ao invés de esperar uma interrupção) e o processamento dos pacotes em lotes. Para os propósitos deste trabalho, dois *frameworks* destacam-se por possuírem suporte a execução em *unikernels*. Estas ferramentas atendem os requisitos de alto desempenho e de estabilidade da rede, definidos pela especificação de NFV.

- **netmap:** O *framework netmap* (RIZZO, 2012) implementa *drivers* modificados de forma a eliminar custos no processamento de pacotes, sendo os principais custos a alocação dinâmica de memória para cada pacote (eliminado pela pré-alocação de recursos), o *overhead* causado por chamadas de sistema (reduzido ao utilizar-se processamento em lotes) e as cópias de memória (eliminado ao compartilhar *buffers* e metadados com a aplicação). Como os *buffers* são pré-alocados e o *driver* está sempre fazendo *pooling* na interface de rede, os pacotes que chegam nela são imediatamente disponibilizados para a aplicação utilizando uma quantidade reduzida de recursos. Para a sua utilização em plataformas virtualizadas, o *netmap* precisa que tanto o hipervisor quanto a máquina virtualizada possuam *drivers* compatíveis, de forma a criar um canal de comunicação próprio. Também existe uma implementação de um comutador virtual, chamado VALE (RIZZO; LETTIERI, 2012) que, quando utilizado em conjunto, são capazes de fornecer um desempenho superior a 40Gbits/s na comunicação convidado-hóspede. É importante notar que o *netmap* também é capaz de executar sem esta arquitetura, em um modo reserva, de forma que seu desempenho fica limitado a implementação da pilha de rede do hipervisor.

- **Data Plane Development Kit (DPDK):** O DPDK (INTEL, 2014) é um *framework* desenvolvido pela Intel com suporte de grandes empresas de telecomunicações, possuindo *drivers* para diversas interfaces de rede físicas e paravirtualizadas. Utiliza técnicas semelhantes a do *netmap* para a redução na utilização de recursos. No entanto, sua arquitetura apresenta uma divisão em bibliotecas, cada uma com funcionalidades diferentes (*e.g.*, *framework multicore*, *ring buffers* e *pool-mode drivers*), que por sua vez podem ser utilizadas separadas ou em conjunto, tanto para o simples encaminhamento de pacotes quanto para o desenvolvimento de funções mais avançadas, embora não tão complexas quanto as que podem ser desenvolvidas pelos *frameworks* para criação de funções virtualizadas de rede. As otimizações implementadas pelo DPDK fazem com que o encaminhamento de pacotes possa ser realizado em menos de 80 ciclos de CPU, e é capaz de ser utilizado também com difererentes hipervisores.

3.1.3 *Frameworks* para implementação de VNFs

Nesta categoria estão as ferramentas que efetivamente são responsáveis por criar e executar as funções virtualizadas de rede. Embora os *frameworks* citados anteriormente sejam efetivos para realizar o encaminhamento de pacotes e outras funções simples, eles não são adequados para a implementação de funções e serviços de rede mais complexos (*e.g.*, IDS, *statefull firewalls* e outras funções de roteamento avançado). Assim, é necessário a utilização de frameworks desenvolvidos especialmente para a criação e execução destas funções mais avançadas.

Algumas das ferramentas que podem ser utilizadas para a implantação das VNFs são:

- **Berkeley Extended Software Switch (BESS):** Anteriormente conhecido como SoftNIC, o BESS (HAN et al., 2015) consiste de uma arquitetura híbrida de *hardware-software*, disponibilizando um *framework* para a programação de funcionalidades das interfaces físicas de rede. Funcionalidades que não são possíveis de serem implementadas diretamente em *hardware*, podem ser implementadas em *software* e, através de *frameworks* de aceleração de pacotes, como os citados anteriormente, podem ser utilizadas sem que haja muita perda de desempenho. Esta interface programável fornecida pelo BESS possui um *pipeline* modular que pode ser utilizado para a criação de funcionalidades que utilizam a interface de rede de modo otimizado.

- **Click Modular Router (CMR):** O *Click Modular Router* (KOHLER et al., 2000) é outro *framework* desenvolvido para a criação de roteadores flexíveis e configuráveis. Um roteador criado com o CMR é composto por pequenos módulos com funções específicas, denominados “elementos”. Cada elemento implementa uma função simples (*e.g.*, classificação, escalonamento, enfileiramento e comunicação com dispositivos de rede), possui um ponto de entrada e saída e pode ser configurado individualmente. Uma configuração no CMR é análoga a um grafo de encaminhamento, com elementos que, quando interconectados, são capazes de fornecer uma função virtualizada de rede avançada. Como o CMR já possui vários elementos implementados, sua configuração resume-se a montar um grafo e configurar os elementos, inclusive utilizando ferramentas (*e.g.*, Clicky) para realizar esta montagem de forma gráfica, de modo que configurações genéricas podem ser criadas e utilizadas em ambientes diferentes com pequenas modificações. Inicialmente desenvolvido como um módulo para o *kernel* do Linux, também possui suporte para ambos *frameworks* de aceleração de pacotes citados anteriormente.

3.1.4 Gerenciamento

Por fim, é necessária uma forma de gerenciar os sistemas utilizados para a implantação da plataforma remotamente. Alguns requisitos necessários para a gerência de plataformas de virtualização de funções de rede não fazem parte do escopo deste trabalho e não serão considerados. No *framework* MANO, o componente de gerenciamento desta plataforma se enquadra em parte no VNFM, ou seja, é capaz de atualizar, modificar e finalizar o funcionamento de uma VNF, apesar que não pode ser utilizado para instanciar novas VNFs ou controlar sua escalabilidade, já que estas são responsabilidades da infrastrutura. Também é capaz de disponibilizar métricas para monitoramento e reconfiguração, necessários para alguns cenários (BONDAN; SANTOS; GRANVILLE, 2014). Dois protocolos apresentam características desejáveis para utilização em conjunto com a plataforma, sendo eles:

- ***Simple Network Management Protocol (SNMP)*:** É um protocolo para o gerenciamento de dispositivos de rede e o mais utilizado para a gerência e configuração de *middleboxes*, por ser simples e flexível (PRESUHN, 2002). Através de uma *Management Information Base* (MIB), o protocolo SNMP utiliza UDP para comunicação entre o agente e a plataforma de gerência, e define funções (*e.g., GetRequest e SetRequest*) para recuperar e alterar as informações que são expostas pela MIB de uma maneira hierárquica, para assim poder gerar métricas sobre o uso do dispositivo. O SNMP também suporta *Traps*, que são notificações assíncronas enviadas pelo agente para a plataforma, seguindo regras pré-definidas na sua configuração. Embora utilizado em sua maior parte por *middleboxes* e comutadores, servidores e sistemas operacionais também possuem suporte para envio de métricas através de SNMP.

- ***Web Services* (WS):** WS podem ser descritos como um conjunto de componentes independentes disponibilizados utilizando um protocolo Web. Podem sem utilizados como uma ferramenta de integração nas mais variadas áreas, incluindo o gerenciamento de redes. Entre os exemplos que podem ser citados de WS para o gerenciamento de redes estão o *Simple Object Access Protocol* (SOAP) (MENDELSOHN et al., 2007), e mais recentemente, Representational State Transfer (REST) (RICHARDSON; RUBY, 2008). O protocolo REST segue a arquitetura cliente-servidor e de protocolo sem estado, de forma que um cliente pode acessar as informações disponibilizadas pelo servidor através de protocolos amplamente utilizados na Internet (*e.g.*, HTTP), e as requisições são independentes entre si. Novas funcionalidades podem ser adicionadas ao WS de forma incremental, conforme for apresentada a necessidade de serem utilizadas.

3.2 PLATAFORMAS EXISTENTES

Embora algumas plataformas para a virtualização de funções de redes tenham sido desenvolvidas, estas atendem apenas parcialmente os requisitos definidos para a implantação de NFV, limitando sua ampla adoção. Assim, esta seção apresenta duas plataformas que podem ser relacionadas com trabalho desenvolvida e suas características e deficiências, de modo a identificar quais requisitos são atendidos apenas em parte ou não são atendidos.

3.2.1 ClickOS

O ClickOS (MARTINS et al., 2014) é uma plataforma otimizada para a execução de VNFs, utilizando o Mini-OS com *drivers* paravirtualizados modificados para suportar o *netmap*. Para a criação e execução das VNFs, é utilizada uma versão modificada do Click Modular Router. Sua arquitetura têm como foco flexibilidade, isolação de ambiente, escalabilidade e alta vazão com baixo atraso. Suas máquinas virtuais são pequenas (5Mb em disco), possuem um tempo de *boot* rápido e adicionam pouca latência. Em conjunto com o comutador VALE e modificações no sistema de entrada/saída do Xen, é capaz de saturar *links* de 10Gb utilizando um único núcleo de processamento e 12Mb de memória.

3.2.2 *Open Platform for NFV* (OPNFV)

O OPNFV (PRICE; RIVERA, 2012) é uma plataforma completa que abrange todos os blocos da arquitetura de NFV. Para isso, utiliza-se de diversas tecnologias existententes, como OpenStack para o papel de VIM, Ceph para armazenamento e uma versão modificada do OpenvSwitch (Openvswitch for NFV) para a aceleração de pacotes. Embora não seja focado especificamente na implantação e execução das VNFs em si, são oferecidas algumas funções virtualizadas já implementadas, como VyOS. No contexto deste trabalho, por estas funções oferecidas ofertarem apenas uma funcionalidade, elas diferem do conceito de uma plataforma para as VNFs, que por sua vez pode ser utilizada para implementar diferentes funcionalidades utilizando uma plataforma em comum.

3.2.3 Soluções Comerciais

Grandes empresas do ramo de telecomunicações estão iniciando o desenvolvimento de plataformas para a implantação de NFV. Estas soluções ainda estão em testes

e como exemplo podem ser citados *Cisco Integrated Services Virtual Router* (ISRv) (SYSTEMS, 2016), um roteador virtual projetado pela Cisco para ser implantado em ambientes NFV, e o Juniper vSRX (NETWORKS, 2016), um *firewall* virtual que pode ser executado na plataforma OPNFV. O fato de serem soluções proprietárias e a impossibilidade de obte-lás para testes faz com que este trabalho foque-se apenas em plataformas e componentes de código livre. Além disso, a utilização de soluções proprietárias, dependendo de sua implementação, pode quebrar o requisito de integração, discutido no capítulo anterior.

3.3 DISCUSSÃO

Este capítulo apresentou o conceito de *NFV enablers*, as tecnologias e ferramentas que podem ser utilizadas para a criação de uma plataforma e quais os requisitos que cada uma delas atende. Também foi apresentada plataformas já desenvolvidas com o objetivo semelhante ao deste trabalho. Assim como o ClickOS, a idéia deste trabalho é escolher e integrar as ferramentas apresentadas nas categorias anteriores, de modo que a plataforma desenvolvida atenda a maior quantidade de requisitos possível. As justificativas para a escolha da ferramenta em cada uma das categorias será apresentada no próximo capítulo, junto com a arquitetura e desenvolvimento da plataforma.

Com relação ao ClickOS é possível dizer que, no momento, é a plataforma mais próxima de atender os requisitos definidos pelo ETSI para a implantação de funções virtualizadas de rede. No entanto, algumas das especificidades utilizadas para a sua implementação não satisfazem completamente os requisitos de NFV. Por exemplo, o requisito de alto desempenho é completamente atendido pelo fato do ClickOS ter sido desenvolvido com este foco, embora para isto sejam necessárias modificações no hipervisor e o uso de um comutador específico, o que ocasiona problemas nos requisitos de portabilidade e integração. O requisito de portabilidade não é atendido pois a plataforma pode ser executada em apenas um hipervisor (Xen). Já o requisito de integração, que determina que uma plataforma deve ser capaz de executar em redes com infraestrutura fornecida por diferente provedores também não é atendido, dado que sua necessidade de alterações no hipervisor e comutador o impossibilita de ser utilizado em infraestruturas heterogêneas. Por fim, o requisito de gerenciamento é atendido apenas parcialmente pois, embora o ClickOS possa ser controlado localmente por linha de comando, não existe uma interface de gerência que pode ser conectada a um *framework* de gerência e orquestração.

4 DESENVOLVIMENTO

Este capítulo apresenta inicialmente a arquitetura interna genérica de uma VNF. Com base nesta arquitetura, componentes individuais foram escolhidos para compor a plataforma proposta neste trabalho. Por fim, este capítulo apresenta uma visão mais aprofundada da implementação em si e do funcionamento da plataforma desenvolvida.

4.1 ARQUITETURA GENÉRICA DE UMA VNF

De acordo com a especificação NFV-SWA 001 (NFV-SWA, 2014), a arquitetura de uma VNF pode ser composta de um ou mais componentes, chamados de *Virtual Network Function Components* (VNFC). Cada um destes componentes é responsável por uma funcionalidade específica dentro da VNF, como o gerenciamento ou a comunicação com as interfaces de rede da VM. Conforme a Figura 4.1, estes componentes podem ser interligados entre si internamente, e devem ser conectados também ao NFVI, de modo que a função virtualizada possa se comunicar com a rede externa.

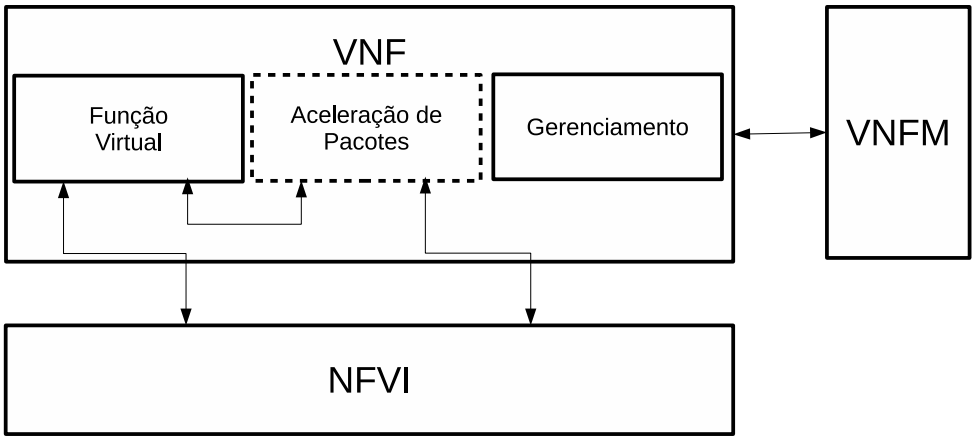


Figura 4.1 – Arquitetura interna de uma VNF Genérica

Fonte: do Autor.

Também é necessário que exista uma interface para ser conectada ao VNFM, de modo que o ciclo de vida e as funcionalidades da VNF possam ser controladas pelo *framework* de gerência e orquestração. Todos os componentes internos da VNF e a sua interface de gerência devem estar contidos em uma imagem que será executada pelo hipervisor. Para isso, é necessário que um sistema operacional forneça um ambiente onde estes componentes possam ser executados. Vale notar que estes componentes internos podem ser implementados como uma única aplicação, desde que esta possua pelo menos

as funcionalidades de execução de funções virtuais e gerenciamento, fazendo com que a plataforma deixe de ser modular mas também possua um maior desempenho. O acelerador de pacotes, por sua vez, é opcional na arquitetura de uma VNF, e pode ser utilizado em casos onde a pilha de rede do sistema operacional não atende os requisitos de NFV.

Esta arquitetura genérica foi adaptada pelos desenvolvedores do ClickOS, apresentando diferenças com relação ao acelerador de pacotes, que é implementado diretamente nos *drivers* paravirtualizados do Mini-OS ao invés de ser um módulo separado, e também não existe uma interface dedicada ao gerenciamento, com a configuração da VNF sendo realizada através do XenStore (*i.e.*, espaço de configuração compartilhado entre o hipervisor e a máquina virtual).

4.2 ARQUITETURA DA PLATAFORMA DESENVOLVIDA

Baseando-se na arquitetura genérica definida anteriormente, a arquitetura da plataforma proposta neste trabalho pode ser descrita como uma máquina virtual composta por três componentes distintos executados no sistema operacional OSv, sendo eles o DPDK, *Click Modular Router* (CMR) e um *Web Service* (WS) utilizando o protocolo REST para comunicação. Conforme a seção anterior, cada um dos componentes utilizados atende um requisito específico, de modo que a utilização em conjunto abrange uma gama maior de requisitos. Assim, a Figura 4.2 apresenta a organização destes componentes dentro da plataforma desenvolvida.

A utilização de máquinas virtuais para a implantação de *middleboxes* ou outros serviços com um alto custo computacional implica em problemas de desempenho, já que existe a adição de uma camada de virtualização entre o ambiente onde a função virtualizada está executando e o *hardware* em si. Com o objetivo de melhorar o desempenho, hipervisores implementam a técnica de paravirtualização, onde tarefas com alto custo de processamento podem ser realizadas com ajuda do sistema hospedeiro. Tal solução, contudo, exige que tanto o hipervisor quanto o sistema operacional virtualizado possuam suporte a paravirtualização, por exemplo, através da utilização de *drivers* de disco e rede especificamente desenvolvidos para este propósito (AGESEN et al., 2012).

O OSv, diferentemente de sistemas operacionais tradicionais, disponibiliza apenas funções necessárias para que uma aplicação possa ser executada. No caso da plataforma desenvolvida, as funções necessarias são um escalonador simples para a gerência de *threads* e interrupções, drivers paravirtualizados para os recursos disponibilizados pelo hipervisor (*e.g.*, dispositivos PCI, teclado e mouse), e gerenciamento de memória e contexto (simplificado pelo uso de um espaço de enderaçamento único). Durante a inicialização da plataforma, o OSv utiliza parâmetros previamente definidos para inicializar e atribuir um endereço IP para a interface de gerência, repassar o controle das interfaces de rede que

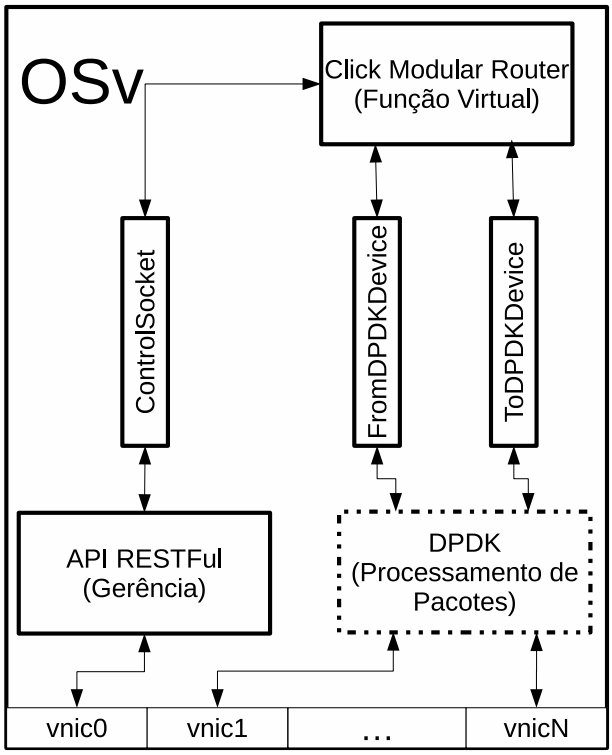


Figura 4.2 – Arquitetura interna da plataforma desenvolvida

Fonte: do Autor.

serão utilizadas pelo CMR para o DPDK e inicializar as *threads* dos componentes. Quando a API de gerência recebe um comando para alterar o arquivo de configuração da função virtualizada ou reinicializar o sistema, isto também é realizado pelo OSv.

Dentro do OSv, *threads* independentes executam cada um dos três componentes. A *thread* da API de gerência fornece fornece uma interface Web contendo estatísticas (*e.g.*, uso de memória, processamento e disco, parâmetros de inicialização e *threads* sendo executadas) sobre o sistema em geral, além de possibilitar o *upload* de novas funções virtualizadas.

Já o DPDK desempenha funções tipicamente atribuídas a pilha de rede dos sistemas operacionais tradicionais. Isto quer dizer que o DPDK instala seus próprios *drivers* para as interfaces, com funções específicas para que a técnica de *pooling* possa ser utilizada, e controla diretamente a alocação de memória e interrupções destas interfaces de forma a otimizar seu funcionamento, sem que o OSv interfira. Embora aplicações mais avançadas (*e.g.*, *firewalls*, encaminhamento de pacotes na camada 3 e *proxys*) possam ser desenvolvidos utilizando apenas o framework do DPDK, sua função nesta plataforma é apenas gerenciar as interfaces e encaminhar os pacotes(*i.e., frames Ethernet*) para que o processamento possa ser realizado pelo CMR.

A *thread* do CMR, por sua vez, é responsável pela criação e execução das funções virtualizadas. Considerando o conceito de VNFs apresentado anteriormente no capítulo

dois, o CMR possui elementos bem definidos que indicam o ponto de entrada e de saída dos pacotes. Na plataforma, estes elementos são, respectivamente, `FromDPDKDevice(DEVID)` e `ToDPDKDevice(DEVID)`, que constituem os únicos pontos de comunicação entre esta *thread* e a do DPDK e, consequentemente, a comunicação com a rede externa. A configuração das funções virtualizadas é definida através de um arquivo lido na inicialização da *thread*, que pode ser alterado através da API de gerência. Caso seja necessário modificar a função, é necessário substituir o arquivo e reinicializar a plataforma.

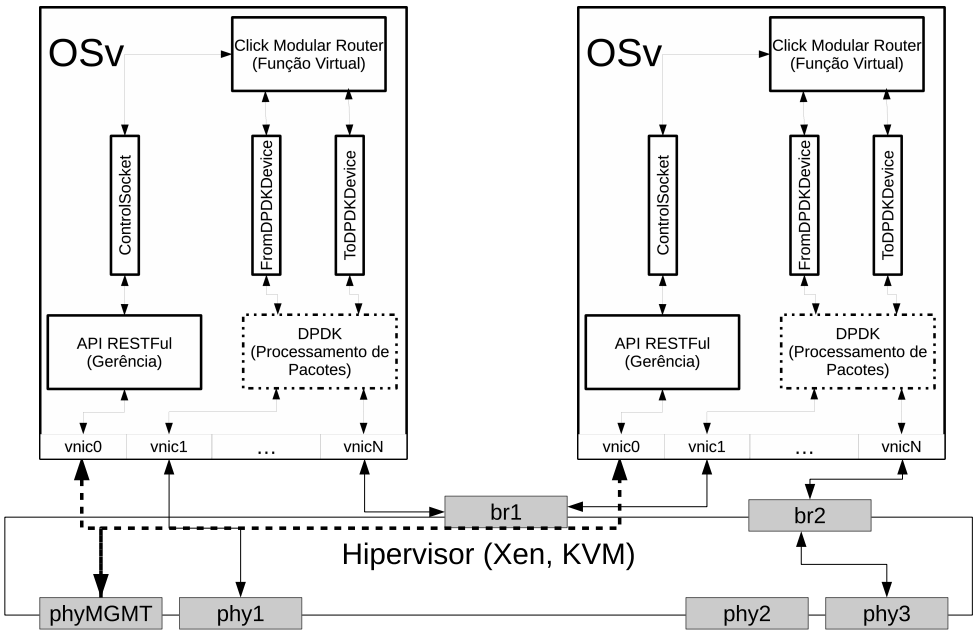


Figura 4.3 – Múltiplas instâncias da plataforma executando em um mesmo hipervisor

Fonte: do Autor.

Como o foco da plataforma é apenas a configuração e execução de VNFs, operações como a interconexão de várias instâncias de modo a criar um grafo de encaminhamento, a atribuição dos recursos virtuais para as instâncias e o provisionamento delas não fazem parte do escopo deste trabalho. Apesar disso, é possível visualizar na Figura 4.3 que, dado uma infraestrutura NFV e um *framework* de gerência e orquestração, a plataforma pode facilmente compor um grafo de encaminhamento ou ser instanciada.

Considerando que o mercado de virtualização de funções de rede está em amplo crescimento (GRAHAM, 2016), a arquitetura da plataforma foi projetada de uma forma modular. No momento, foram utilizados os componentes mais promissores e que estão em um estágio maduro de desenvolvimento, mas isto não impede que no futuro novos componentes possam ser utilizados na platafora. É importante ressaltar, porém, que estes componentes precisam ser portados para o sistema operacional e não apresentar limitações quanto ao funcionamento em conjunto com os outros componentes. Por exemplo, o DPDK poderia ser substituído pelo *netmap*, já que existem elementos para comunicação entre o CMR e o netmap, apesar de que não existe uma versão do *netmap* para o OSv.

O CMR também poderia ser substituído, apesar que, para isto, seria necessário que as configurações das funções virtualizadas fossem reescritas.

4.3 JUSTIFICATIVAS

Nesta seção são apresentados os fatores que possibilitaram a escolha das ferramentas mais adequadas para serem utilizadas na plataforma descrita na seção anterior. Esta seleção levou em consideração principalmente o número de requisitos atendidos por cada ferramenta, bem como quão bem cada requisito é suportado. Também serão apresentados os problemas encontrados com as outras ferramentas, de modo a reforçar as escolhas realizadas. Outros fatores que contribuíram para a seleção das ferramentas foi o seu estado de desenvolvimento, sua utilização por outros trabalhos relacionados e, por fim, a possibilidade de integração com as ferramentas de outras categorias.

A primeira ferramenta a ser escolhida foi o *framework* para realizar o desenvolvimento e a execução das funções virtualizadas de rede. Esta categoria apresenta-se como a mais importante por ser a ferramenta que efetivamente irá executar as funções virtualizadas, portanto deve ser robusta, simples e possuir uma boa documentação para facilitar o desenvolvimento das funções virtualizadas por usuários. Das alternativas pesquisadas, o CMR apresentou-se como a melhor, devido ao longo tempo de desenvolvimento, extensiva lista de componentes (elementos) que podem ser usados para a criação de funções virtualizadas de rede, seu suporte a ambos *frameworks* para aceleração de pacotes e, por fim, ser amplamente adotado em trabalhos relacionados. O BESS, por sua vez, apresentou-se como uma solução limitada por suportar um único *framework* de aceleração de pacotes, seu desenvolvimento se encontrar em um estágio inicial, a falta de documentação para a escrita das funções virtualizadas, e a necessidade de um controlador externo para o gerenciamento das instâncias.

Após a escolha do *framework* para a execução das VNFs, foi realizada a escolha do sistema operacional. Isto se deve ao fato de o sistema operacional compor a base onde as outras ferramentas serão executadas. Desta forma, optou-se pela escolha do OSv, já que este atende os requisitos de portabilidade (*i.e.*, é capaz de executar em diferentes hipervisores) e simplicidade, pois sua implementação consiste apenas de um único espaço de endereçamento de memória, diminuindo o *overhead* imposto por trocas de contexto. Também possui suporte ao modelo de paravirtualização *virtio*, além de ser isolado das outras instâncias por executar em uma máquina virtual e possuir uma interface de gerência REST. O OSv também é capaz de executar aplicações desenvolvidas originalmente para ambientes Linux, pois mantém sua compatibilidade com a ABI (*i.e.*, interface de baixo nível) do Linux. Tal característica do OSv é importante pois as outras ferramentas foram desenvolvidas originalmente para Linux. O IncludeOS, por sua vez, apresenta funcionalidades

semelhantes ao OSv porém não possui uma pilha de rede completa, o que o impossibilita de ser utilizado para a execução de VNFs, além de não possuir uma interface de gerência externa (*i.e.*, pode ser gerenciado apenas localmente) e, por estar em um estágio de desenvolvimento inicial. Finalmente, o Mini-OS não se mostra adequado para NFV por não possuir um sistema de arquivos, sua pilha de rede necessitar de modificações para que alto desempenho seja possível e por executar apenas no hipervisor Xen. Além disso, seu gerenciamento se faz pelo XenStore, novamente uma funcionalidade exclusiva do hipervisor Xen.

Já para a escolha da ferramenta de aceleração de pacotes, existe uma especificação divulgada pelo ETSI (NFV-IFA, 2016a) que apresenta os requisitos necessários. Esta especificação apresenta duas técnicas utilizadas para a aceleração de pacotes, *PCI-passtrough* e paravirtualização. No *PCI-passtrough* os descritores PCI da interface física de rede são passados diretamente para a plataforma virtualizada pelo hipervisor, de forma que a plataforma possui um controle total da interface física, o que elimina a perda de desempenho ocasionada pela camada de virtualização. No entanto, isto também implica que a interface física pode ser utilizada apenas por uma única função virtualizada, o que não satisfaz o requisito de compartilhamento de recursos, pois outras funções virtualizadas não possuem mais acesso a esta interface. Além disso, esta mesma função virtualizada torna-se dependente do *hardware* da interface, quebrando também o requisito de portabilidade, pois sua migração depende que a nova infraestrutura onde ela será implantada possua o mesmo *hardware*. Desta forma, este método é recomendado para ser utilizado durante a fase de implantação de NFV, até que o modelo paravirtualizado possua um desempenho desejável.

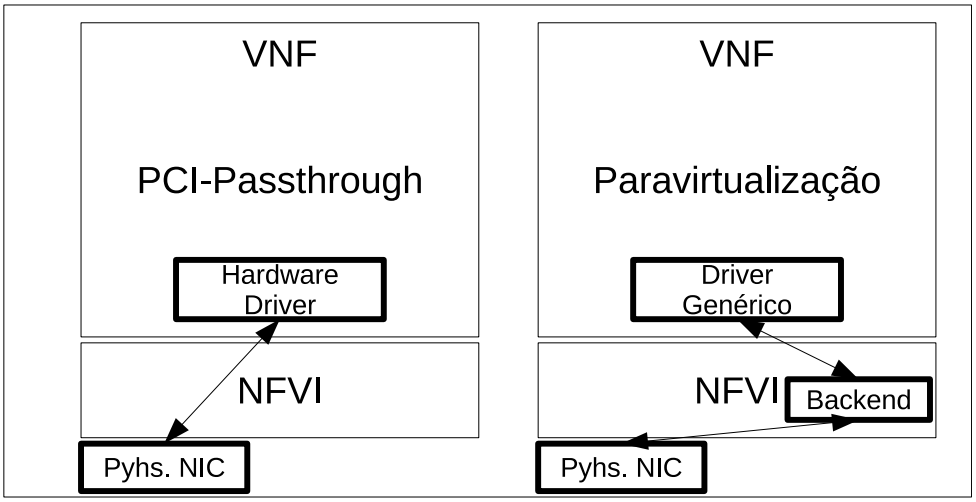


Figura 4.4 – Arquitetura de VNFs utilizando PCI-passthrough e paravirtualização

Fonte: do Autor.

Assim, a outra técnica recomendada pela especificação é a paravirtualização, que utiliza um modelo de abstração para a comunicação entre o sistema virtualizado e a infra-

estrutura. Como mostrado na Figura 4.4, esta técnica utiliza um driver genérico divido em dois componentes que comunicam-se entre si através de um canal específico para este propósito (*backend* e *frontend*), de modo que o hipervisor é responsável por implementar o *backend* enquanto que o sistema virtualizado implementa o *frontend*. A utilização deste modelo atende os requisitos de compartilhamento de recursos (pois vários *frontends* podem se comunicar com o mesmo *backend*) e também o requisito de portabilidade (o modelo possui uma especificação bem definida já implementada por diferentes hipervisores). Algumas outras otimizações que devem ser suportadas pelo *framework* incluem aceleração para pacotes TCP (*e.g.*, TCP e *Checksum offloading*) e criptografia (*e.g.*, SSL/TLS, IPSec e SRTP). Desta forma, o DPDK foi escolhido como o framework para aceleração de pacotes por apresentar uma implementação do modelo de paravirtualização com suporte a *pooling* e pré-alocação de *buffers*. Embora o *netmap* apresente soluções semelhantes, sua implementação efetiva depende de um comutador virtual VALE implementado no *backend*, o que impede sua portabilidade.

Por fim, a escolha do protocolo de gerência foi passível com a análise da especificação do ETSI, que define um modelo com funções necessária para a configuração e gerência de VNFs. A especificação GS NFV-IFA 010 (NFV-IFA, 2016b) define os requisitos funcionais para os três blocos do *framework* MANO (*i.e.*, NFVO, VNFM e VIM). Como a interface de gerência utilizada nesta plataforma deve disponibilizar apenas o controle do ciclo de vida da sua própria instância (sendo a visão global das instâncias uma tarefa de um agente externo), esta interface é capaz de disponibilizar métodos para serem utilizados pelo VNFM, como mudanças no ciclo de vida da instância (*e.g.*, inicializar, parar e pausar VNFs), e gerenciamento da VNF (*e.g.*, modificar funções virtualizadas e coletar estatísticas sobre utilização de recursos em geral da instância). A especificação DGS/NFV-SOL002 (SOL, 2016) apresenta uma API REST como um modelo de referência que pode ser utilizado para a comunicação entre a instância e o VNFM. Desta forma, a utilização da API REST se justifica para manter conformidade com as especificações, além de possibilitar uma implementação incremental das funções de gerência, diferentemente do SNMP que poderia ser utilizado apenas depois que uma MIB estática fosse definida.

4.4 IMPLEMENTAÇÃO

Esta seção apresenta as modificações que foram realizadas nas ferramentas que foram selecionadas na seção 4.3 para que elas possam ser executadas em conjunto no OSv. As principais modificações foram realizadas no CMR, para possibilitar seu funcionamento em conjunto com o OSv e com o DPDK. Apesar de o OSv ser compatível com a ABI do Linux e com os padrões POSIX, sua arquitetura requer modificações específicas, como a relação dos vínculos necessários pelas bibliotecas para a sua execução.

Devido a arquitetura do OSv, a aplicação a ser executada deve ser compilada como uma biblioteca compartilhada. Em sistemas tradicionais, bibliotecas compartilhadas são executáveis contendo métodos pré-compilados que são utilizados por outras aplicações. Estas bibliotecas devem funcionar independente da posição de memória em que elas são copiadas, através de uma técnica chamada *position-independent code*. Por exemplo, em sistemas tradicionais, quando é alocado um endereçamento de memória para um processo, este é isolado dos outros processos e, devido a isto, para o processo a posição inicial da memória é zero. Já quando o processo precisa de uma biblioteca compartilhada, esta será alocada no primeiro espaço disponível da área de memória do processo, que não será necessariamente zero. Devido a isso, estas bibliotecas precisam executar da mesma maneira independente de qual for a sua posição inicial da memória.

No OSv, um único espaço de endereçamento de memória é compartilhado pelo *kernel* e aplicações. Vale notar que, embora não exista o conceito de processos, aplicações diferentes podem ser executadas através de *threads* diferentes. Isto implica que uma aplicação pode acessar áreas de memórias de outras aplicações, de forma que se estas aplicações não forem confiáveis, cria-se um risco de segurança.

Na Figura 4.5 pode ser visto um exemplo do espaço virtual de memória de uma instância da plataforma, com o *kernel* utilizando as posições iniciais seguido do *Web Service*, o CMR e, finalmente, o DPDK.

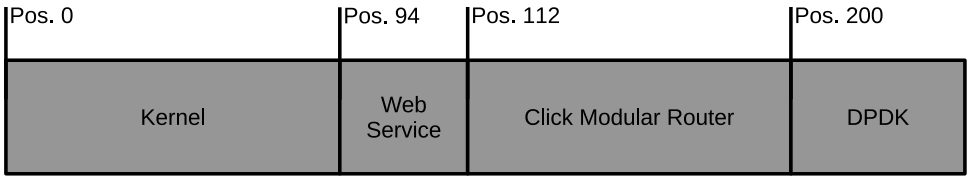


Figura 4.5 – Exemplo do espaço de endereçamento virtual de uma instância da plataforma.

Fonte: do Autor.

Inicialmente foram feitas alterações no *Makefile* do CMR, de modo a tentar executá-lo no OSv. Assim, foram adicionadas as *flags* `-fPIC` e `-shared`, sendo usadas para compilar o CMR com *position-independent code* e como uma biblioteca compartilhada. Após isso, verificou-se através de testes internos do CMR que a versão compilada funciona no OSv. No entanto, ao tentar utilizar os elementos padrão de entrada e saída de pacotes (`FromDevice` e `ToDevice`, respectivamente), a aplicação apresentou um erro ao tentar abrir um *socket*. Isto se deve a implementação de *sockets* no OSv não ser compatível com a biblioteca *pcap*, utilizada pelos componentes do CMR para a captura de pacotes. Como os elementos de entrada e saída do DPDK possuem um desempenho melhor em relação aos elementos padrão do CMR, foi preferível realizar modificações para o funcionamento dos elementos do DPDK.

Em seguida, a versão modificada do CMR foi compilada juntamente com o DPDK.

Novamente foram encontrados problemas devido as modificações realizadas no DPDK para seu funcionamento com o OSv. Estes problemas foram identificados como sendo relacionados ao tamanho do *buffer* requisitado pelo CMR, que era originalmente de 64Mb, enquanto que o DPDK suportava *buffers* de até 8Mb. , e também com relação ao tamanho do cache utilizado pelo CMR, que também foi reduzido. Após as alterações mostradas na Figura 4.6, foi possível a compilação do CMR.

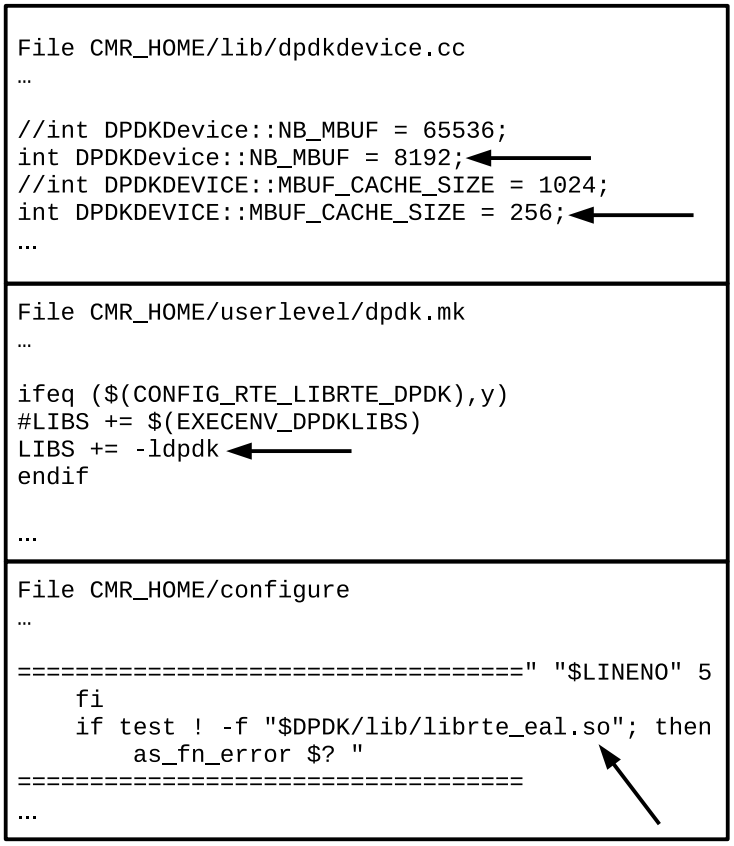


Figura 4.6 – Alterações realizadas na interface do CMR com o DPDK.

Fonte: do Autor.

Mesmo assim, ocasionalmente ocorriam erros durante a inicialização do CMR na plataforma, que foram identificados como causados pelo carregamento incorreto das bibliotecas do DPDK. A ordem de carregamento das bibliotecas no OSv é feita de forma alfabética, sem que uma verificação de dependências entre elas seja realizada, o que resultava na chamada de métodos de bibliotecas que ainda não haviam sido carregadas. A utilização de uma biblioteca combinada (*i.e.*, todas bibliotecas combinadas em um único arquivo) resolveu este problema, de modo que os métodos são chamados após o carregamento completo do arquivo e, consequentemente, de todas bibliotecas. A Figura 4.7 apresenta as dependências do CMR antes e depois das modificações apresentadas na Figura 4.6. Pode ser observado que originalmente cada uma das bibliotecas do DPDK eram chamadas separadamente e, após a modificação, apenas uma biblioteca `libdpdk.so` é

chamada.

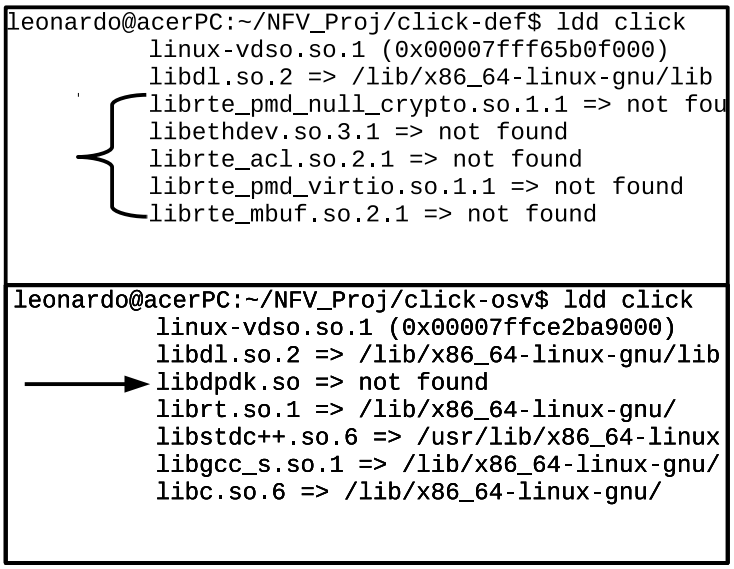
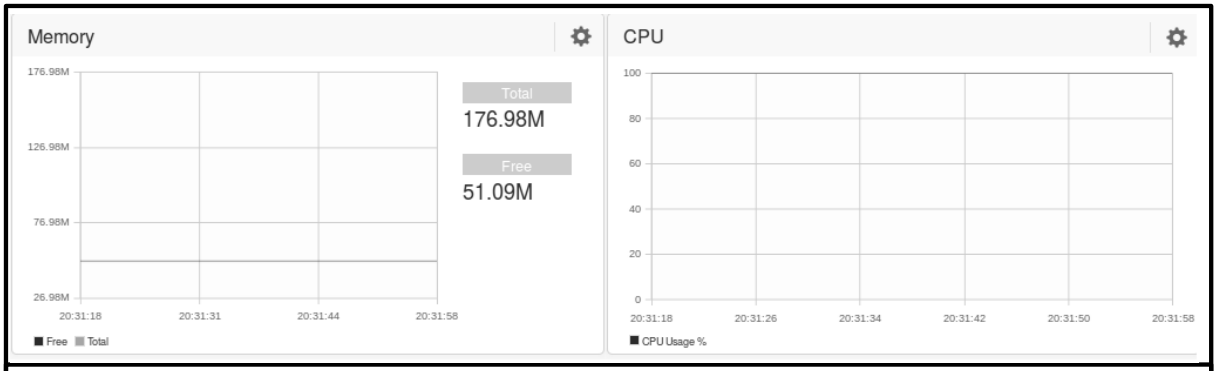


Figura 4.7 – Bibliotecas requeridas pelo CMR antes e depois da modificação (nota-se que após a modificação, apenas `libdpdk.so` é chamada).

Fonte: do Autor.

Após estas modificações terem sido realizadas no CMR, seu funcionamento em conjunto com o DPDK e a interface de gerência não apresentou mais problemas.

Através da interface REST é possível modificar os parâmetros de inicialização do CMR, recuperar estatísticas sobre o uso de memória, CPU e disco, além de poder fazer o *upload* de novas configurações e controlar o ciclo de vida da plataforma (*i.e.*, inicializar e terminar a execução de funções e reiniciar a plataforma). Vale notar que embora tenha sido utilizada uma interface Web, esta apenas repassa a chamada do método para a API REST, sendo assim apenas uma interface gráfica para a API. Na eventual utilização de um *framework* de gerência externo ao invés da interface Web integrada, o framework pode processar as requisições REST diretamente. A Figura 4.8 apresenta a interface Web exibindo consumo de memória e CPU da plataforma, e os mesmos dados sendo recuperados diretamente da API REST.




```
leonardo@acerPC:~$
curl http://192.168.100.217:8000/os/memory/total?op=GET
185573376

leonardo@acerPC:~$
curl http://192.168.100.217:8000/os/cpu/load?op=GET
0,74
```

Figura 4.8 – Interface Web apresentando uso de memória e CPU e exemplo de uma requisição REST direta.

Fonte: do Autor.

5 AVALIAÇÃO

Este capítulo tem como objetivo apresentar e discutir os resultados obtidos com a avaliação da plataforma desenvolvida e sua comparação com outras plataformas existentes. Inicialmente são apresentadas as métricas utilizadas na avaliação e por qual motivo estas foram escolhidas. Em seguida, o cenário e metodologia utilizados para a avaliação são descritos. Finalmente, os resultado obtidos são apresentados e discutidos de forma a evidenciar os benefícios obtidos com a plataforma proposta.

5.1 MÉTRICAS

A definição das métricas a serem utilizadas para a avaliação e a comparação da plataforma desenvolvida com outras plataformas foram realizadas de acordo com a RFC 2544 (BRADNER; MCQUAID, 1999), que define a metodologia para avaliação de dispositivos de rede, e a especificação NFV-IFA 003 (NFV-IFA, 2016a), que especifica a metodologia para avaliação de comutadores virtuais utilizados em arquiteturas NFV. Vale ressaltar que alguns parâmetros de testes definidos pela RFC não se enquadram diretamente no cenários de NFV como, por exemplo, tamanho de frames para FDDI e *Token Ring*.

Assim, as três métricas mais utilizadas para a avaliação de dispositivos de rede costumam ser vazão, atraso e *jitter*. A seguir serão apresentadas, de acordo com as definições da RFC 2544, uma breve descrição de cada uma destas métricas e de que forma elas contribuem para a avaliação do desempenho.

- **Vazão:** A vazão pode ser definida pela maior taxa de *frames* que pode ser enviada através do dispositivo testado sem que haja perdas. Para que esta taxa possa ser calculada, uma quantidade crescente de *frames* deve ser enviada pelo transmissor até que o receptor receba um número menor do que o enviado, o que indica uma perda de pacotes pelo dispositivo. Também devem ser utilizados diferentes tamanhos de *frames*, sendo definidos os valores de 64, 128, 256, 512, 1024, 1280 e 1500 *Bytes*, o que configura o valor máximo e mínimo de *frames Ethernet* e valores intermediários. O resultado final deve ser mostrado em forma de gráfico, com a coordenada **x** sendo o tamanho do pacote e a coordenada *y* sendo a vazão máxima. A relevância desta métrica se dá pelo aumento do tráfego em redes de computadores, devido a uma maior quantidade de usuários e informações. Desta forma, uma vazão maior significa que menos recursos serão necessários para encaminhar este tŕafego. Por exemplo, a utilização de *links* com velocidade de 10Gbits/s ao invés de 10 *links* com velocidade de 1Gbits/s ocasiona uma redução de equipamentos, consumo de

energia e complexidade de rede.

- **Atraso:** O atraso, por sua vez, é definido como o tempo que um *frame* necessita para ser enviado do transmissor até o receptor, conhecido como *One Way Delay* (OWD) ou como o tempo necessário para um frame ser enviado e a resposta ser recebida pelo transmissor, neste caso chamado de *Round Trip Time* (RTT). Um atraso alto pode prejudicar aplicações em tempo real, como simulações distribuídas e transmissão de vídeos, e também prejudicar a vazão. Embora o método OWD seja o mais preciso, sua necessidade de sincronização entre o transmissor e o receptor faz com que não seja possível a sua utilização no cenário apresentado na próxima seção.
- ***Jitter*:** O *jitter* pode ser definido como a variação entre o maior e o menor atraso. Embora não seja definido diretamente nas especificações, esta métrica é muito importante para a qualidade de aplicações em tempo real. A transmissão de voz por IP (VoIP), por exemplo,é prejudicada se o *jitter* for alto, com cortes e falhas na transmissão.

5.2 CENÁRIO E METODOLOGIA

O cenário utilizado para a execução dos testes também foi definido de acordo com a RFC 2544 (BRADNER; MCQUAID, 1999), sendo composto por 2 máquinas virtuais (VM), que funcionam como transmissor e receptor, e uma máquina virtual que executa as plataformas de teste, como pode ser visto na Figura 5.1. O sistema operacional onde as VMs estão sendo executadas é um Debian com hipervisores Xen 4.4.2 e KVM 1.2. O servidor onde o hipervisor é executado possui um processador Intel Xeon E5620@2.40Ghz com 8 núcleos e 12GB de memória DDR3-ECC@1066MHz. Para o receptor e o transmissor foram dedicados 1GB de ram e um núcleo físico, através de *CPU Pinning*.

As plataformas testadas, por sua vez, foram o ClickOS, Linux, a plataforma desenvolvida executando no Xen e no KVM, consistindo de 4 ambientes a serem testados, sendo dedicados 192MB de ram e 1 núcleo físico para o ClickOS e a plataforma desenvolvida, e 1GB de ram e um núcleo físico para o Linux, por este não inicializar com 192Mb.

Para a interconexão das VMs, o comutador virtual OpenVSwitch foi utilizado tanto no Xen como no KVM, por este ser o comutador padrão no Xen, através da utilização de duas pontes (*bridges*, sendo uma para a comunicação do transmissor com a plataforma e a outra para comunicação da plataforma com o receptor). Todas as VMs foram configuradas para utilizarem a paravirtualização padrão do seu hipervisor, no caso *netfront* para o Xen e VirtIO para o KVM, exceto no teste onde a plataforma desenvolvida foi testada no Xen,

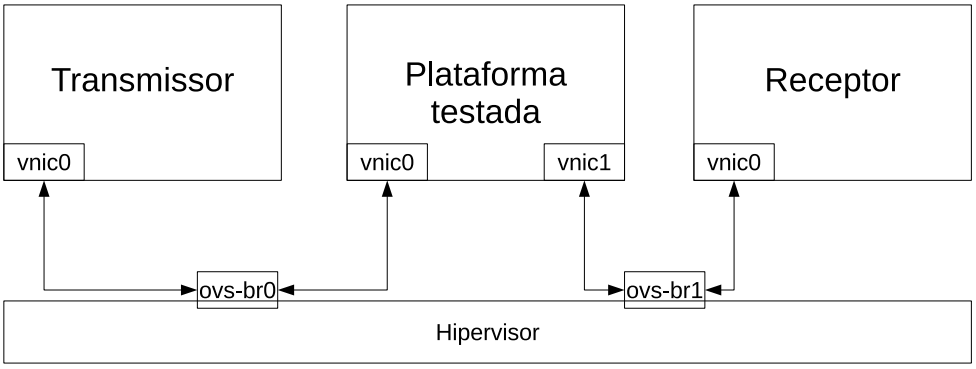


Figura 5.1 – Cenário utilizado para a avaliação das plataformas

Fonte: do Autor.

onde se fez necessário a utilização da paravirtualização VirtIO por questões de compatibilidade. Vale ressaltar que os testes com o ClickOS foram feitos com as funcionalidades das interfaces de rede *Generic Segmentation Offload* (GSO), *TCP Segmentation Offload* (TSO) e *Generic Receive Offload* (GRO) desativados, pois estas funcionalidades não são suportadas pelo ClickOS.

A função de rede utilizada para avaliação é responsável por encaminhar pacotes entre duas interfaces de rede, sem realizar nenhum processamento mais complexo. Tal função foi escolhida por possibilitar que o desempenho máximo teórico em ambas as plataformas fosse observado. Além disso, para tentar definir as melhores métricas possíveis de serem atigindas no servidor com o cenário utilizado, a máquina virtual executando Linux cria uma ponte interna, utilizando linux-bridges entre as duas interfaces.

Por fim, as ferramentas utilizadas para as medições foram o *netperf* (HP, 2015), para a medição da vazão, com um fluxo TCP contínuo para pacotes de variando de 64B a 1500B, sendo realizado no final de cada bateria de testes uma média da vazão dos pacotes, dado que um intervalo de confiança de 95 % tenha sido atingido. O D-ITG (BOTTA; DAINOTTI; PESCAPÈ, 2012), por sua vez, foi utilizado para a medição de atraso e *jitter*, com um fluxo UDP simulando uma conexão VoIP, um fluxo TCP simulando uma conexão Telnet, e um fluxo UDP com uma variação aleatória no tamanho dos datagramas entre 64 e 1500 Bytes. Estes três fluxos foram escolhidos por apresentarem aplicações normalmente utilizadas em uma rede de produção. O atraso foi calculado através do RTT, já que não foi possível atingir uma sincronização precisa entre o receptor e o transmissor utilizando NTP (MILLS et al., 2010), e protocolos mais precisos, como PTP (IEEE. . . , 2008), necessitam de funcionalidades não existentes nas interfaces de rede virtualizadas.

5.3 RESULTADOS

Esta seção tem como objetivo apresentar e discutir os resultados obtidos com os testes realizados anteriormente.

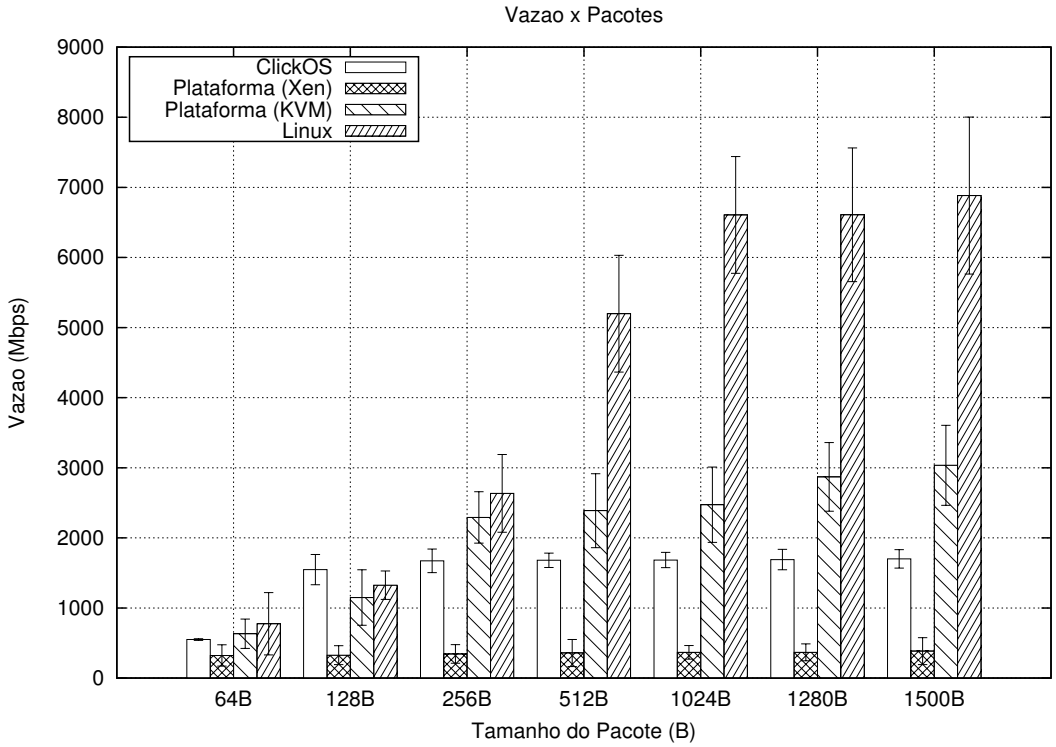


Figura 5.2 – Avaliação da vazão das plataformas

Fonte: do Autor.

A Figura 5.2 apresenta o gráfico de vazão das plataformas testadas. Tomando o Linux como o valor máximo que é possível atingir neste cenário, a plataforma apresenta bons resultados quando utilizada no KVM, e consistentemente possui um desempenho melhor em relação ao ClickOS. Nota-se que a vazão para pacotes de 128B do ClickOS é superior ao da plataforma desenvolvida, mas como a diferença ainda está dentro da variação, pode-se afirmar que as duas possuem um desempenho semelhante.

A Figura 5.3 apresenta o atraso das plataformas testadas. É possível observar que o ClickOS possui um atraso semelhante ao do Linux para os três casos, enquanto que a plataforma desenvolvida apresenta um atraso maior. Apesar disso, todas as plataforma, exceto a plataforma desenvolvida executando no Xen, apresentam valores inferiores a 1ms.

Finalmente, a Figura 5.4 apresenta o *jitter*, observando-se um resultado inferior entre as três métricas da plataforma desenvolvida. No entanto, mesmo com um *jitter* consideralvemente alto em relação ao ClickOS e Linux, a plataforma ainda sim apresenta um jitter inferior a 300 microsegundos.

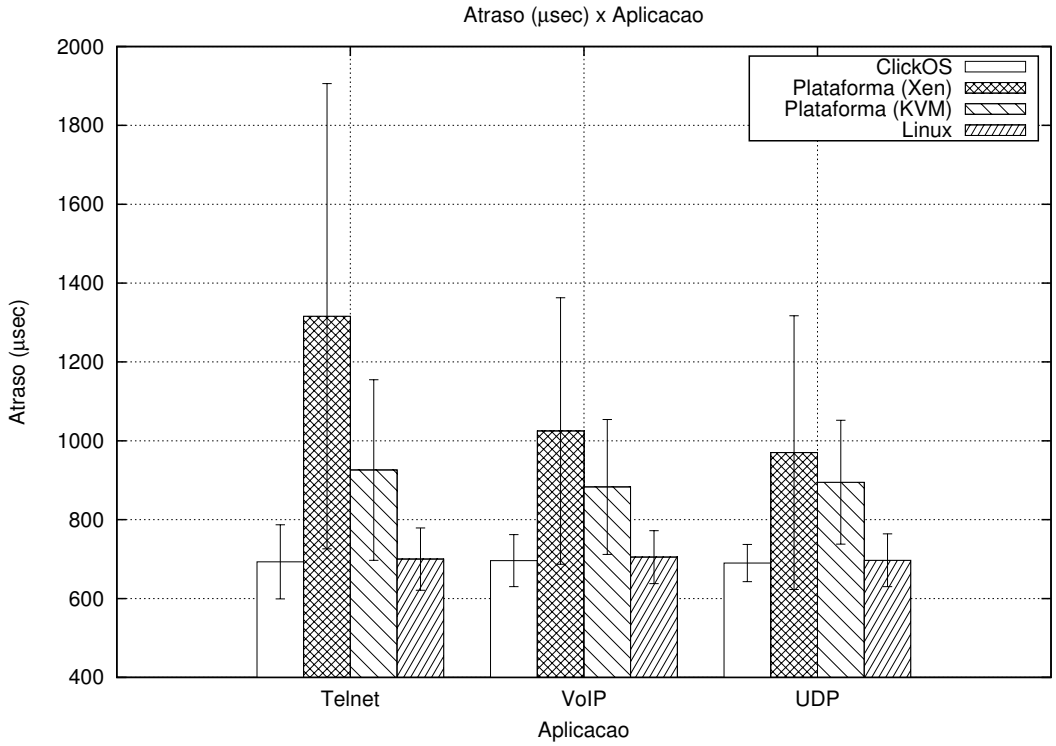


Figura 5.3 – Avaliação do atraso nas plataformas

Fonte: do Autor.

5.4 DISCUSSÃO

É possível ver que a vazão da plataforma desenvolvida é superior ao do ClickOS quando executada no KVM, o que já era esperado, devido a otimização tanto do OSv quanto do DPDK para seu funcionamento com as interfaces paravirtualizadas *virtio*. Com relação ao atraso, esperava-se que o ClickOS tivesse um resultado menor que o da plataforma desenvolvida, devido ao uso do Mini-OS, que embora não seja otimizada em termos de vazão, apresenta um atraso reduzido devido ao seu compartilhamento de uma área de memória com o hipervisor. É possível que a retirada do DPDK de dentro da plataforma, de modo que as conexões sejam feitas diretamente entre as interfaces virtualizadas e VNF, ocasione um ganho tanto de vazão como de atraso, desde que a implementação do CMR possua o suporte para a pilha otimizada do OSv.

No entanto, não esperava-se que o desempenho nas três métricas para a plataforma desenvolvida executando no servidor Xen apresentasse resultados muito inferiores quando comparado as outras plataformas. Após uma leitura da documentação do hipervisor Xen, verificou-se que a implementação de paravirtualização *virtio* (necessária para utilização da plataforma desenvolvida) foi realizada apenas para efeitos de compatibilidade, sem uma preocupação de se otimizar o seu desempenho. Assim, uma alternativa para este desempenho baixo seria implementar *drivers* no DPDK para a plataforma netfront,

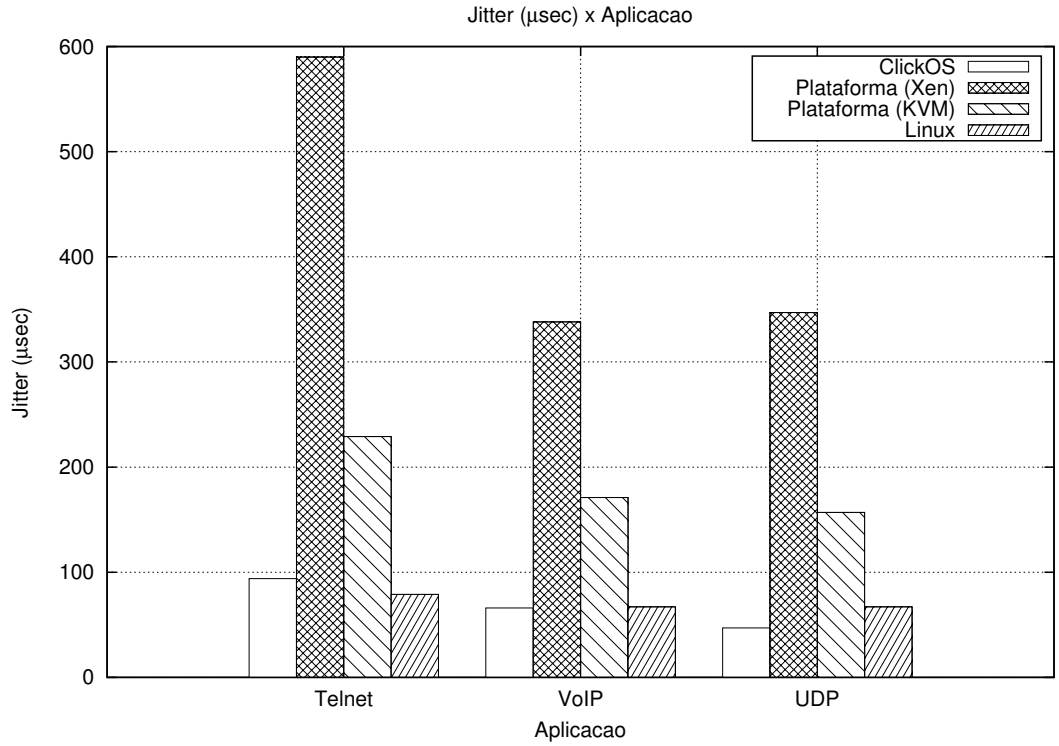


Figura 5.4 – Avaliação do jitter nas plataformas

Fonte: do Autor.

utilizada pelo Xen, ou então novamente utilizar a pilha de rede do OSv, por possuir suporte ao *netfront*, embora não tão otimizado quanto o do Mini-OS.

Por fim, não sabe-se o motivo do *jitter* apresentar uma variação superior a das outras plataformas, mesmo quando a plataforma está sendo executada no KVM. Inicialmente foi tentado desativar a interface de gerência do OSv, que poderia estar consumindo recursos, no entanto após esta ser desativada não houve diferença significativa no *jitter*.

Logo, é possível ver que a plataforma desenvolvida apresenta uma vantagem em relação ao ClickOS com relação a sua vazão, embora seu desempenho com relação ao atraso e *jitter* seja um pouco inferior (por volta de 200 microsegundos). No entanto, durante a fase de escolha das ferramentas, testes com instâncias do *netperf* executando no OSv e MiniOS sem utilizar aceleradores de pacotes em uma comunicação direta entre sistema convidado e *host* apresentaram resultados de vazão e atraso superiores no OSv, o que indica que a perda de desempenho está localizada no DPDK ou no CMR. Testes realizados utilizando um forward nativo do DPDK (*i.e.*, sem utilizar o CMR) apresentaram resultados semelhantes ao obtido pela plataforma, com um decréscimo de apenas 50 microsegundos no atraso e 30 microsegundos no *jitter*. Assumindo que sejam estes os custos impostos pelo processamento do CMR, é possível dizer que o problema de desempenho localiza-se no DPDK.

Apesar das deficiências da plataforma apresentada com relação ao atraso e ao *jit-*

ter, existem vantagens com relação aos requisitos de NFV. Por exemplo, a sua capacidade de executar em diferentes hipervisores sem modificações faz com que ela seja adequada para cenários onde existem infraestruturas de diferentes fornecedores e os requisitos com relação a atraso e *jitter* não sejam tão estritos, já que a diferença de desempenho nestes requisitos é inferior a meio milisegundo. Sua interface de gerência também é outro ponto positivo, considerando que a gerência de diversas instâncias pode ser centralizada em um único ponto de gerência através da interface REST, o que não é possível com o ClickOS.

Por fim, vale dizer que a plataforma tem possibilidade de ser compatível com a versão do *framework* MANO desenvolvida pela ETSI, embora não tenha sido testada, por possuir uma imagem pronta, com interfaces de rede bem definidas, tanto para a sua gerência como para as funções virtualizadas, diferentemente do ClickOS.

6 CONCLUSÃO

A área de Virtualização de Funções de Rede tem atraído um grande interesse de pesquisadores do mundo inteiro. Por ainda se encontrar em um estágio inicial, alguns conceitos e métodos não estão definidos claramente. A plataforma de execução das VNFs é um destes, já que embora existam modelos e especificações apresentando uma visão de alto nível de como estas devem ser implementadas, ainda não existe uma implementação capaz de atender efetivamente os requisitos especificados.

Partindo desta idéia, foi realizado uma revisão bibliográfica sobre a arquitetura e a especificação de Virtualização de Funções de Rede como um todo, de forma a identificar os requisitos, e quais os problemas existentes nas plataformas atuais. Também foi estudado o conceito de *NFV Enablers*, de forma a identificar ferramentas necessárias para a implantação e de que maneira elas poderiam ser divididas, optando-se no final pela divisão em categorias atendendo requisitos distintos. Assim, o próximo passo foi analisar as ferramentas em cada uma destas categorias de forma a selecionar as mais adequadas para que fossem integradas, formando a plataforma. Com as ferramentas escolhidas em cada uma das categorias, foi feita a sua integração. Isto envolveu pequenas modificações em todas ferramentas, para que funcionassem em conjunto e de forma efetiva. Por fim, foram realizados testes em um cenário comum a todas, para que a plataforma pudesse ser validada e seu desempenho comparado com a plataforma existente. Como apresentado no capítulo anterior, verificou-se que seu desempenho, embora inferior para atraso e *jitter*, é comparável com a outra plataforma, além de possuir algumas funcionalidades ainda não existentes, como uma interface de gerenciamento, e capacidade de executar em diferentes infraestruturas, apesar que com um desempenho reduzido.

Com relação aos objetivos do trabalho, é possível dizer que estes foram atingidos satisfatoriamente, por apresentar um protótipo funcional de uma nova plataforma que atende de forma satisfatória uma maior quantidade de requisitos do que as plataformas disponíveis atualmente, além de uma pesquisa sobre categorias de ferramentas que podem ser utilizadas para a construção destas plataforma. Acredita-se também que a relevância deste trabalho está não só no protótipo desenvolvido, mas também na pesquisa das ferramentas, que pode ser utilizada como base para trabalhos futuros que tenham como objetivo a otimização das plataformas existentes atualmente. Devido a modularidade da plataforma desenvolvida, também é possível utilizar esta pesquisa para futuramente otimizar a própria plataforma.

Como trabalhos futuros, sugere-se a substituição do módulo *framework* de aceleração de pacotes pela pilha de rede nativa do OSv, por esta possuir otimizações tanto para interfaces paravirtualizadas *virtio* quanto *netfront*, o que poderia resolver o problema de desempenho da plataforma no hipervisor Xen, além de uma menor utilização de recursos.

Também, pelo fato da API REST ser extensível, funcionalidades para funções de rede específicas podem ser adicionadas e carregadas apenas quando estas funções estiverem executando. Como um exemplo, poderia adicionar-se funções para adição e remoção de regras de um *firewall* em tempo real, dado que a função virtualizada tenho sido implementada levando em consideração *hotswapping* no CMR.

REFERÊNCIAS BIBLIOGRÁFICAS

AGESEN, O. et al. Software techniques for avoiding hardware virtualization exits. In: **Proceedings of the 2012 USENIX Conference on Annual Technical Conference**. Berkeley, CA, USA: USENIX Association, 2012. (USENIX ATC'12), p. 35–35. Disponível em: <http://dl.acm.org/citation.cfm?id=2342821.2342856>.

ANDERSON, T. et al. Overcoming the internet impasse through virtualization. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 38, n. 4, p. 34–41, abr. 2005. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.2005.136>.

BONDAN, L.; SANTOS, C. R. P. d.; GRANVILLE, L. Z. Management requirements for clickos-based network function virtualization. In: **10th International Conference on Network and Service Management (CNSM) and Workshop**. [S.l.: s.n.], 2014. p. 447–450. ISSN 2165-9605.

BOTTA, A.; DAINOTTI, A.; PESCAPÈ, A. A tool for the generation of realistic network workload for emerging networking scenarios. **Computer Networks**, v. 56, n. 15, p. 3531–3547, 2012.

BRADNER, S.; MCQUAID, J. **Benchmarking Methodology for Network Interconnect Devices**. [S.l.], March 1999.

BRATTERUD, A. et al. Includeos: A minimal, resource efficient unikernel for cloud services. In: **2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2015. p. 250–257.

DEERING, S. E.; HINDEN, R. M. **Internet Protocol, Version 6 (IPv6) Specification**. [S.l.], December 1998.

ETSI. **Network Functions Virtualisation – Introductory White Paper**. ETSI, 2012. Acesso em 04 set. 2016. Disponível em: <https://portal.etsi.org/NFV/NFV_White_Paper.pdf>.

EVENS, J. A comparison of containers and virtual machines for use with nfv. tUL, 2015.

FONSECA, R. et al. **IP Options are not an option**. [S.l.], Dec 2005. Disponível em: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-24.html>.

FOUNDATION, O. N. **Software-Defined Networking (SDN) Definition**. Open Networking Foundation, 2016. Acesso em 04 set. 2016. Disponível em: <https://www.opennetworking.org/sdn-resources/sdn-definition>.

GRAHAM, I. M. L. **Network Functions Virtualization Market Worth Over $15 Billion by 2020, Says IHS Markit**. 2016.

HAN, S. et al. Softnic: A software nic to augment hardware. In: **Technical Report UCB/EECS-2015-155**. [S.l.]: EECS Department, University of California, Berkeley, 2015.

HANDLEY, M. Why the internet only just works. **BT Technology Journal**, Kluwer Academic Publishers, Hingham, MA, USA, v. 24, n. 3, p. 119–129, jul. 2006. ISSN 1358-3948. Disponível em: <http://dx.doi.org/10.1007/s10550-006-0084-z>.

HP. Netperf. **URL http://netperf. org**, 2015.

HYPERVISOR, X. P. **Mini-OS**. Xen Project Hypervisor, 2015. Acesso em 9 out. 2016. Disponível em: <http://wiki.xenproject.org/wiki/Mini-OS>.

IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. **IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)**, p. 1–269, July 2008.

INTEL, D. Data plane development kit. **URL http://dpdk. org**, 2014.

ISG, N. E. **ETSI GS NFV 004 V1. 1.1-2013 Network Functions Virtualisation (NFV): Virtualisation Requirements**. [S.l.]: 2013, 2013.

ITU. **ICT Facts and Figures 2016**. International Telecommunication Union, 2016. Acesso em 03 set. 2016. Disponível em: <http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf>.

KIVITY, A. et al. Osv—optimizing the operating system for virtual machines. In: **2014 USENIX Annual Technical Conference (USENIX ATC 14)**. Philadelphia, PA: USENIX Association, 2014. p. 61–72. ISBN 978-1-931971-10-2. Disponível em: <https://www.usenix.org/conference/atc14/technical-sessions/presentation/kivity>.

KOHLER, E. et al. The click modular router. **ACM Trans. Comput. Syst.**, ACM, New York, NY, USA, v. 18, n. 3, p. 263–297, ago. 2000. ISSN 0734-2071. Disponível em: <http://doi.acm.org/10.1145/354871.354874>.

MADHAVAPEDDY, A. et al. Unikernels: Library operating systems for the cloud. In: **Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: ACM, 2013. (ASPLOS '13), p. 461–472. ISBN 978-1-4503-1870-9. Disponível em: <http://doi.acm.org/10.1145/2451116.2451167>.

MARTINS, J. et al. Clickos and the art of network function virtualization. In: **Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2014. (NSDI'14), p. 459–473. ISBN 978-1-931971-09-6. Disponível em: <http://dl.acm.org/citation.cfm?id=2616448.2616491>.

MENDELSOHN, N. et al. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. [S.l.], abr. 2007. Http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

MILLS, D. et al. **Network Time Protocol Version 4: Protocol and Algorithms Specification**. [S.l.], June 2010. Disponível em: <http://www.rfc-editor.org/rfc/rfc5905.txt>.

NETWORKS, J. vsrx services gateway. **Datasheet**, 2016.

NFV, E. G. 002: Network functions virtualisation (nfv); architectural framework, v 1.2.1. **ETSI, December**, 2014.

NFV, G. 001: Network functions virtualisation (nfv); use cases, v 1.1. 1. **ETSI, December**, 2013.

NFV-IFA. Network functions virtualisation; acceleration technologies; vswitch benchmarking and acceleration specification. **ETSI, April**, 2016.

NFV-IFA, G. Network functions virtualisation (nfv); management and orchestration; functional requirements specification. **ETSI, April**, 2016.

NFV-SWA, G. Virtual network functions architecture. **ETSI, December**, 2014.

PAVLICEK, R. **Unikernels - Beyond Containers to the Next Generation of Cloud**. [S.l.]: OŔeilly, 2016.

PRESUHN, R. **Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)**. [S.l.], December 2002. Disponível em: <https://tools.ietf.org/html/rfc3416>.

PRICE, C.; RIVERA, S. Opnfv: An open platform to accelerate nfv. **White Paper**, 2012.

RAO, S. K. Sdn and its use-cases - nv and nfv. **Network**, v. 2, p. H6, 2014.

RICHARDSON, L.; RUBY, S. **RESTful Web Services**. O'Reilly Media, 2008. ISBN 9780596554606. Disponível em: <https://books.google.com.br/books?id=XUaErakHsoAC>.

RIZZO, L. netmap: A novel framework for fast packet i/o. In: **2012 USENIX Annual Technical Conference (USENIX ATC 12)**. Boston, MA: USENIX Association, 2012. p. 101–112. ISBN 978-931971-93-5. Disponível em: <https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo>.

RIZZO, L.; LETTIERI, G. Vale, a switched ethernet for virtual machines. In: **Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies**. New York, NY, USA: ACM, 2012. (CoNEXT '12), p. 61–72. ISBN 978-1-4503-1775-7. Disponível em: <http://doi.acm.org/10.1145/2413176.2413185>.

SHERRY, J. et al. Making middleboxes someone else's problem: Network processing as a cloud service. In: **Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication**. New York, NY, USA: ACM, 2012. (SIGCOMM '12), p. 13–24. ISBN 978-1-4503-1419-0. Disponível em: <http://doi.acm.org/10.1145/2342356.2342359>.

SOL, D. N. Network functions virtualisation (nfv); protocols and data models; restful protocols specification for the ve-vnfm reference point. **ETSI, May**, 2016.

STALLINGS, W. **Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud**. [S.l.]: Addison-Wesley Professional, 2015.

SYSTEMS, B. C. Nfv and sdn are not the same. hereś why. **Brocade Communications Systems**, 2014.

SYSTEMS, C. Cisco integrated services virtual router. **Datasheet**, 2016.

TAYLOR, D.; TURNER, J. Towards a diversified internet. **White paper, November**, 2004.