UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FERNANDO SILVA

Detecção e Mitigação de Ataques Usando Aprendizado Supervisionado e Orquestração de VNFs Baseadas em Contêineres

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho

Porto Alegre
2021

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Silva, Fernando

Detecção e Mitigação de Ataques Usando Aprendizado Supervisionado e Orquestração de VNFs Baseadas em Contêineres / Fernando Silva. – Porto Alegre: PPGC da UFRGS, 2021.

63 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2021. Orientador: Alberto Egon Schaeffer-Filho.

1. Mitigação de Ataques. 2. Orquestração de VNFs. 3. Contêiner. I. Schaeffer-Filho, Alberto Egon. II. Título.

RESUMO

A virtualização de funções de rede (Network Function Virtualization - NFV) desacopla as funções de rede dos dispositivos físicos, simplificando a implantação de novos serviços. Ao contrário das middleboxes tradicionais, as Virtual Network Functions (VNFs) podem ser dinamicamente implementadas e reconfiguradas sob demanda, colocando desafios de gerenciamento rigorosos aos sistemas em rede. Selecionar VNFs de um repositório definindo onde elas serão colocadas na rede virtualizada e encadeando-as para obter o comportamento desejado são problemas que precisam ser resolvidos por um orquestrador. Nesse sentido, um conjunto de VNFs podem ser instanciadas para desempenhar diversas tarefas, onde destacamos aqui a possibilidade de utilizar VNFs para a mitigação de ataques, buscando garantir a resiliência da rede. Neste trabalho é proposto um mecanismo, chamado Intel-OCNF, que através do uso de aprendizado supervisionado e virtualização baseada em contêineres, permite identificar quais funções de rede devem ser instanciadas com base em dados de monitoramento. A solução proposta busca minimizar o uso de recursos, e agilizar o processo de instanciação e gerenciamento das VNFs, de forma a assegurar a resiliência da rede. O protótipo desenvolvido foi integrado ao orquestrador NFVO, e opera de forma automatizada e sem dependência de ações do operador de rede. Os experimentos foram avaliados com base no desempenho dos algoritmos de categorização do tráfego e de seleção do mitigador, e no desempenho das VNFs em contêineres. Os resultados obtidos nos experimentos demonstraram que com a utilização dos algoritmos propostos é possível alcançar mais de 90% de sucesso na detecção das categorias de ataque. Além disso, com a utilização de virtualização baseada em contêiner é possível provisionar uma VNF com *overhead* muito menor que o de uma VNF baseada em VM.

Palavras-chave: Mitigação de Ataques. Orquestração de VNFs. Contêiner.

Attack Detection and Mitigation Using Supervised Learning and Container-Based VNF Orchestration

ABSTRACT

Network Function Virtualization (NFV) decouples network functions from physical devices, simplifying the deployment of new services. Unlike traditional middleboxes, Virtual Network Functions (VNFs) can be dynamically deployed and reconfigured on demand, posing strict management challenges to networked systems. Selecting VNFs from a repository defining where they will be placed on the virtualized network and chaining them together to get the desired behavior are issues that need to be resolved by an Orchestrator. In this sense, a set of VNFs can be instantiated to perform various tasks, where we highlight here the possibility of using VNFs to mitigate attacks, seeking to ensure network resilience. This work proposes a mechanism, called Intel-OCNF, which, through the use of supervised learning and container-based virtualization, allows to identify which network functions should be instantiated based on monitoring data. The proposed solution seeks to minimize the use of resources, and streamline the process of instantiation and management of VNFs, in order to ensure the resilience of the network. The developed prototype was integrated with the NFVO orchestrator, and it operates in an automated way and without dependence on the actions of the network operator. The experiments were evaluated based on the performance of traffic categorization and mitigator selection algorithms and on the performance of containerized VNFs. The results obtained in the experiments showed that using the proposed algorithms it is possible to reach more than 90% of success in detecting the attack categories. Moreover, by using container-based virtualization, it is possible to provision a VNF with a much smaller overhead than a VM-based VNF.

Keywords: Attack mitigation, VNF Orchestration, Container.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A *Network Function Virtualization* (NFV) está surgindo como um novo paradigma para o fornecimento de funções elásticas de rede por meio de instâncias de funções virtualizadas de rede (*Virtual Network Function* - VNF) executadas em plataformas de computação virtualizadas. Este capítulo contextualiza os principais conceitos de NFV e introduz o tema dessa dissertação.

1.1 Contextualização

A NFV é uma nova arquitetura de rede que usa técnicas de virtualização para consolidar categorias de equipamentos de rede em servidores de propósito geral (ETSI, 2013c). Em uma rede tradicional, cada função distinta é normalmente implementada como um *appliance* especializado baseado em *hardware* proprietário (ETSI, 2014). Já nestes equipamentos de rede invariavelmente há uma quantidade substancial de *software*, mas o *software* e o *hardware* não podem ser separados. A NFV substitui dispositivos de rede dedicados por um *software* que executa em máquinas virtuais ou containers. Uma função de rede, como *firewall* ou balanceador de carga, implantado no ambiente NFV, é conhecida como uma função virtualizada de rede (VNF). A NFV promove inovação rápida de serviços, eficiência operacional aprimorada, uso reduzido de energia, interfaces padronizadas e abertas, maior flexibilidade e melhor eficiência (ETSI, 2013b).

O avanço que o conceito de NFV representa pode causar uma mudança de paradigma para muitas partes interessadas, incluindo operadores de rede, fornecedores de soluções, integradores de serviços, provedores e usuários de serviços em geral. No entanto, várias questões podem dificultar a adoção dessa nova arquitetura, como por exemplo a orquestração das VNFs.

No contexto da NFV, atividades como decidir quais VNFs devem ser instanciadas, onde necessitam ser fisicamente localizadas e como elas são interconectadas, são obtidas usando algoritmos desenvolvidos por operadores de rede. Essas atividades são referidas como seleção, posicionamento e encadeamento, respectivamente, e são os principais desafios dessa arquitetura (Mijumbi et al., 2016). O componente de *software* responsável por executar essas atividades é chamado de orquestrador NFV - *Network Functions Virtualization Orchestrator (NFVO)*. Tradicionalmente, o problema de identificar qual VNF instanciar é tratado manualmente por um operador de rede, que seleciona o conjunto mais

apropriado de VNFs e o utiliza como entrada para abordagens automatizadas de posicionamento e encadeamento.

1.2 Problema

Selecionar VNFs de um repositório, definindo onde elas serão colocadas na rede virtualizada e encadeando-as para obter o comportamento desejado são problemas que precisam ser resolvidos. Outra dificuldade é a necessidade das VNFs lidarem com possíveis problemas na operação da própria rede, adaptando-se a mudanças através de respostas sensíveis e imediatas a determinadas alterações (por exemplo, padrões de tráfego anômalos ou ataques). Isso visa possibilitar maior resiliência à operação da rede, permitindo por exemplo ajustar a quantidade de recursos ou tipos de VNFs instanciadas para entregar um serviço em resposta a alterações nos padrões de tráfego. Atualmente, os operadores de rede tendem a criar manualmente novos serviços, identificando as VNFs a serem implantadas para um novo serviço e implantando-as usando um sistema de suporte operacional (*Operational Support System* - OSS).

Alguns estudos recentes investigaram formas para identificar qual VNF deve ser instanciada, a necessidade de novas instâncias de uma mesma VNF e o dimensionamento de VNF. Na maioria utilizam maquinas virtuais, não fazendo um melhor aproveitamento dos recursos de *software* e *hardware* e não tratam do provisionamento de novas instâncias de VNFs ainda não existentes. Por exemplo, Schardong, Nunes and Schaeffer-Filho (2018) propõem em seu artigo um mecanismo baseado em leilões, onde agentes fazem lances sobre a alocação de VNFs, utilizando VMs e o operador da rede deve manualmente indicar o estado atual da rede, quais serviços devem ser fornecidos e como. Já Zhang et al. (2017) apresentam um mecanismo baseado em uma adaptação das técnicas de aprendizado online para prever futuras cargas de trabalho das *Service Chains*, possibilitando estimar as próximas taxas de tráfego e ajustar a implantação automatizada de VNFs, não prevendo a instanciação de novas VNFs e sim ajustando os recursos de VNFs já existentes. Na proposta de Fei et al. (2018), é apresentada uma abordagem proativa para provisionar novas instâncias para VNFs sobrecarregadas, com base nas taxas de fluxo estimadas, propondo um método de aprendizado online que visa minimizar o erro na previsão das demandas das *Service Chains*, também não prevendo a instanciação de novas VNFs mas sim de VNFs existentes. Zhou and Guo (2017) propõem em seu artigo uma estrutura onde permite alocação e implantação flexíveis de recursos para VNFs e incorpora recur-

sos de monitoramento de eventos, coleta de informações e análise de dados para facilitar a detecção de atividades anômalas, se limitando a funções de mitigação de ataques DDoS.

A predição de VNF, que busca identificar qual *Network Function* (NF) deve ser instanciada, a instanciação de uma nova NF, a orquestração de serviços e de recursos, e a virtualização das NFs são problemas que precisam ser resolvidos para prover resiliência ao ambiente NFV. Os principais desafios são como e com base em quais informações identificar automaticamente qual NF deve ser instanciada, e posteriormente orquestrar esta NF buscando manter a rede resiliente.

1.3 Solução Proposta

Essa dissertação apresenta um mecanismo, chamado *Intelligent Orchestration of Containerized Network Functions for Anomaly Mitigation (Intel-OCNF)*, que através do uso de aprendizado supervisionado permite identificar quais funções de rede devem ser instanciadas com base em dados de monitoramento. O objetivo principal do Intel-OCNF é definir quais funções devem ser instanciadas de forma a assegurar a resiliência da rede contra tráfego malicioso. Para minimizar o uso de recursos, e agilizar o processo de instanciação e gerenciamento, o Intel-OCNF utiliza virtualização baseada em contêineres. Esse mecanismo tem o objetivo de identificar anomalias no tráfego de rede e implantar VNFs a priori para garantir a qualidade do serviço. O protótipo desenvolvido foi integrado ao orquestrador NFVO, para fornecer orquestração inteligente de funções de rede para mitigação de anomalias. As principais contribuições desse trabalho são: (i) Desenvolvimento de um mecanismo baseado em técnicas de *machine learning* para classificar anomalias no tráfego de rede e, com o uso de lógica fuzzy e da expertise do operador de rede, determinação do conjunto ideal de funções virtualizadas para mitigação; (ii) Exploração da viabilidade de uso de contêineres para minimizar o uso de recursos e agilizar o processo de instanciação e gerenciamento; (iii) Investigação e análise das condições de rede que podem ser potencialmente usadas para determinar quais funções manter instanciadas, gerenciando assim de forma automática o ciclo de vida das VNFs.

1.4 Estrutura da Dissertação

A estrutura dessa dissertação é a seguinte. O Capítulo 2 apresenta a fundamentação teórica, trazendo os conceitos de virtualização de função de rede, gerenciamento e orquestração de NFV, containers e aprendizagem supervisionada. O Capítulo 3 é dedicado aos principais trabalhos relacionados. O Capítulo 4 detalha a solução de orquestração inteligente de funções de rede. O Capítulo 5 traz o protótipo e a avaliação dos resultados obtidos. Por fim, o Capítulo 6 aponta as considerações finais e norteia os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo, é apresentado o estado da arte com relação à virtualização de funções de rede, containers e as técnicas de aprendizado de máquina utilizadas para identificar as funções de rede necessárias.

2.1 Network Function Virtualization

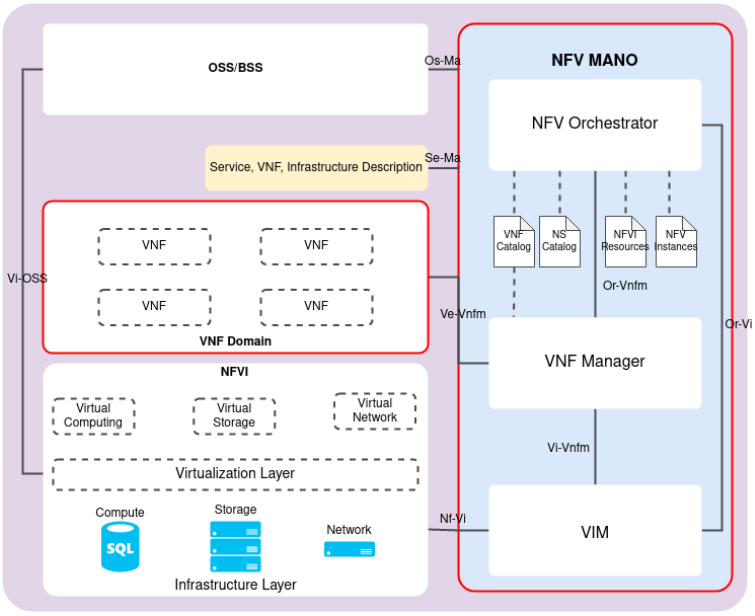
A dependência substancial de redes em seu *hardware* subjacente e a existência de vários dispositivos de *hardware* especializados, por exemplo, *firewalls*, equipamentos de inspeção profunda de pacotes (DPI) e roteadores na infraestrutura de rede, aumentaram os desafios enfrentados pelos provedores de serviços de rede. Os ciclos de vida reduzidos desses tipos de *hardware*, devido ao ritmo acelerado da inovação tendem a multiplicar os investimentos *Operational Expenditure (OPEX)* e *Capital Expenditure (CAPEX)* (CHI-OSI et al., 2012).

A virtualização de funções de rede visa definir um padrão de arquitetura para a substituição de dispositivos de *hardware* por dispositivos virtuais, aprimorando a tecnologia de virtualização padrão de TI para permitir a consolidação de muitos tipos de equipamentos de rede em servidores (Mijumbi et al., 2016). NFV envolve a implementação de funções de rede em *software* que podem ser executadas em uma variedade de *hardware* de servidores de propósito geral, capazes de serem movidas ou instanciadas em vários locais da rede, conforme necessário, sem a necessidade de instalar novos equipamentos. Essa tecnologia pode fornecer benefícios significativos para as operadoras de rede e seus clientes: reduzir os gastos de capital das operadoras e os gastos operacionais por meio de custos reduzidos de equipamentos e redução do consumo de energia; *time-to-market* reduzido para implantar novos serviços de rede; melhor retorno do investimento de novos serviços; maior flexibilidade para escalar, reduzir ou evoluir serviços; abertura para o mercado de utilitários virtuais e novos *softwares*; e oportunidades para testar e implantar novos serviços inovadores com menor risco (Han et al., 2015).

A estrutura arquitetônica da NFV concentra-se nas alterações que provavelmente ocorrerão na rede de uma operadora devido ao processo de virtualização das funções de rede (ETSI, 2013a). Ou seja, a estrutura arquitetônica concentra-se nos novos blocos funcionais e pontos de referência trazidos pela virtualização das redes de uma operadora. A Figura 2.1 mostra a estrutura arquitetônica da NFV representando os componentes,

blocos funcionais e pontos de referência na estrutura.

Figura 2.1: Estrutura arquitetônica de referência NFV



Fonte: Adaptado de (ETSI, 2013a)

Os principais componentes são mostrados por linhas vermelhas, estão no escopo da NFV, são alvos em potencial para padronização e estão disponíveis nas implantações atuais, mas podem precisar de extensões para lidar com a virtualização de funções de rede. Os principais blocos funcionais e os pontos de referência entre esses blocos em sua maioria estão presentes nas implantações atuais, mas outros podem ser adições necessárias para dar suporte ao processo de virtualização e consequente operação. Os blocos funcionais são (ETSI, 2013a):

- *Função virtualizada de rede (VNF)*: Uma VNF é uma virtualização de uma função de rede. Exemplos de NFs são Gateway de Serviço (SGW), Gateway de Rede (PGW), elementos em uma rede doméstica, como Gateway Residencial (RGW) e funções de rede como *firewalls* e sistema de detecção de intrusão (IDS).

- *Sistema de gerenciamento de elementos (EMS)*: O sistema de gerenciamento de elementos executa a funcionalidade de gerenciamento para uma ou várias VNFs.

- *Infraestrutura de NFV, incluindo Hardware e recursos virtualizados*:

  - *Infraestrutura NFV* (NFVI): A infraestrutura NFV abrange todos os componentes de *hardware* e *software* que constroem o ambiente em que as VNFs são implantadas, gerenciadas e executadas. Da perspectiva da VNF, a camada de

virtualização e os recursos de hardware parecem uma única entidade, fornecendo os recursos virtualizados necessários.

- *Recursos de hardware*: Na NFV, os recursos físicos de *hardware* incluem processamento, armazenamento e rede que fornecem recursos às VNFs por meio da camada de virtualização (por exemplo, *hipervisor*). Além disso, os recursos de rede podem abranger domínios diferentes.
- *Camada de virtualização e recursos virtualizados*: A camada de virtualização abstrai os recursos de *hardware* e desacopla o *software* VNF do *hardware* subjacente, garantindo assim um ciclo de vida independente de *hardware* para as VNFs.

- *Gerenciador da Infraestrutura Virtualizada (VIM)*: Do ponto de vista da NFV, o gerenciamento da infraestrutura virtualizada compreende as funcionalidades usadas para controlar e gerenciar a interação de uma VNF com recursos de computação, armazenamento e rede sob sua autoridade, bem como sua virtualização.

- *Orquestrador (NFVO)*: O orquestrador é responsável pela orquestração e gerenciamento da infraestrutura de NFV, recursos de *software* e serviços de rede no NFVI.

- *Gerenciador da VNF (VNFM)*: Um gerenciador de VNF é responsável pelo gerenciamento do ciclo de vida da VNF (por exemplo, instanciação, atualização, consulta, dimensionamento, encerramento). Vários gerenciadores de VNF podem ser implantados; um VNF Manager pode ser implantado para cada VNF ou um VNF Manager pode servir várias VNFs.

- *Descritor de serviço de VNF e de infraestrutura*: Este conjunto de dados fornece informações sobre o modelo de implantação da VNF, o VNF *Forwarding Graph*, as informações relacionadas ao serviço e os modelos de informações da infraestrutura de NFV.

- *Sistemas de suporte a operações e negócios (OSS/BSS)*: O OSS é um conjunto de ferramentas de *software* que permite a automação das principais tarefas operacionais, enquanto que o BSS provê o suporte para os serviços relacionados ao negócio.

A estrutura arquitetônica mostra as funcionalidades necessárias para a virtualização e a consequente operação da rede de uma operadora. Mas a arquitetura não especifica

quais funções da rede devem ser virtualizadas, pois essa é uma decisão do proprietário da rede.

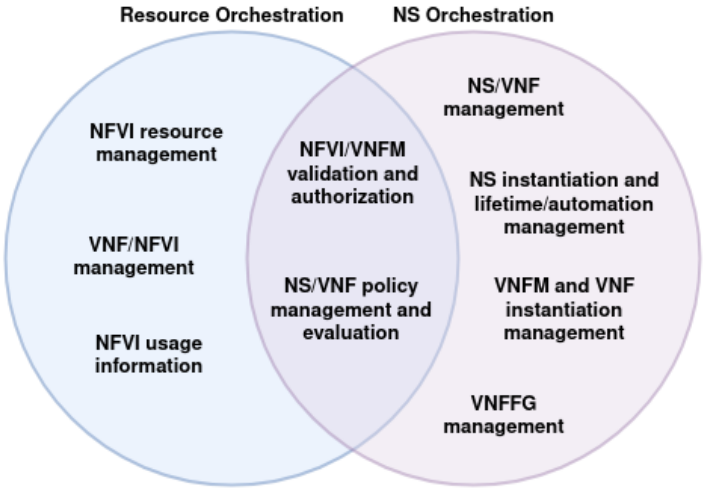
O gerenciamento e orquestração de NFV (*NFV Management and Orchestration - NFV-MANO*) de funções virtualizadas de rede define uma plataforma de serviço composta por um orquestrador (NFVO) e um gerenciador (*VNF Manager (VNFM)*) de VNFs, responsáveis por fornecer serviços de rede. Os serviços de rede podem ser definidos com a instanciação conjunta de VNFs e os encadeamentos (*Service Chains*) entre as diferentes VNFs para realizar conjuntamente uma função mais complexa (Schardong; Nunes; Schaeffer-Filho, 2021). Os componentes da estrutura arquitetônica do NFV-MANO são considerados alvos potenciais para o desenvolvimento em projetos de código aberto e/ou posterior padronização.

O NFVO é um componente essencial da estrutura arquitetural NFV-MANO, que ajuda a padronizar as funções virtualizadas de rede. O NFVO executa orquestração de recursos e orquestração de serviços de rede, bem como outras funções (ERSUE, 2013). Ele é um componente central de uma solução baseada em NFV que reúne diferentes funções para criar um único serviço de orquestração que engloba todo o *framework* e tem um uso de recursos bem organizado. Visa combinar mais de uma função para criar serviços de ponta a ponta. Para este fim, a funcionalidade NFVO pode ser dividida em duas grandes categorias: orquestração de recursos e orquestração de serviços. A orquestração de recursos é usada para fornecer serviços que suportam o acesso aos recursos da NFVI de maneira abstraída, independente de quaisquer *Virtualized Infrastructure Managers* (VIMs), bem como a governança das instâncias da VNF que compartilham recursos da NFVI. Já a orquestração de serviços (NS) lida com a criação de serviços de ponta a ponta, compondo diferentes VNFs e o gerenciamento de topologia das instâncias de serviço de rede.

Conforme o esquema de divisão, representado na Figura 2.2, algumas funcionalidades pertencem a ambas as categorias, mas têm perspectivas diferentes, ou seja, os recursos da NFVI ou o impacto na NS. Além disso, as orquestrações de recursos e serviços focam principalmente em recursos NFVI e instâncias NS, respectivamente. No entanto, eles também consideram o relacionamento com as instâncias VNF correspondentes.

Além disso, conforme especificação do ETSI com relação aos requisitos funcionais do MANO, que são definidos em (ETSI, 2014), as principais categorias de funções operacionais podem ser identificadas. As funções operacionais estão relacionadas com o gerenciamento de NSs e VNFs: gerenciamento da informação, gerenciamento do ciclo de vida, gerenciamento de falhas e gerenciamento de recursos virtuais.

Figura 2.2: As funcionalidades do NFVO dividem-se entre orquestração de recursos e orquestração de NS



Fonte: O Autor

- *O gerenciamento de informações de VNF* inclui o gerenciamento de *VNF package* e de informações de instância da VNF. O gerenciamento de informações da NS compreende o gerenciamento do modelo de implantação, as informações da instância e o desempenho da NS. Em geral, o gerenciamento de informações inclui verificação e validação de integridade e autenticidade, bem como recuperação e coleta de informações e *status* de desempenho.

- *O gerenciamento do ciclo de vida de VNFs e NSs* inclui instanciação, escalonamento, atualização e encerramento de VNFs e NSs, respectivamente.

- *O gerenciamento de falhas* inclui a coleta de notificações de alarme, fornecimento de informações sobre falhas, solicitação de reparos e pré-formação de reparos automatizados ou sob demanda.

- *O gerenciamento de recursos virtuais* consiste no gerenciamento da associação entre os recursos NS/VNF e NFVI através de modelos de comprometimento de recursos (modelo de reserva, modelo de quota e a pedido). Os recursos virtuais relacionados a VNF incluem recursos de computação e armazenamento necessários para os componentes da VNF, bem como recursos de rede necessários para garantir a conectividade intra-VNF. Os recursos virtuais relacionados a NS compreendem redes, sub-redes, portas, endereços, links e regras de encaminhamento e são usados com o objetivo de garantir a conectividade entre VNFs. O gerenciamento dos recursos virtualizados inclui a atribuição, atualização, escalonamento e encerramento. O

NFVO gerencia os recursos pertencentes à infraestrutura virtual cooperando com um ou mais VIMs.

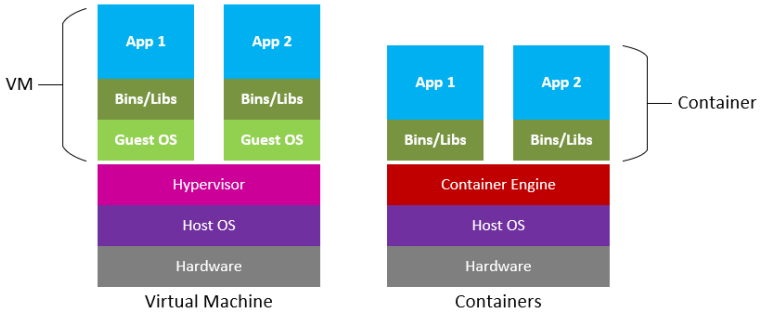
As funções operacionais dos outros elementos do MANO também podem pertencer a algumas das categorias acima. Em particular, o gerenciamento de recursos virtuais (infraestrutura) é executado pelo VIM, enquanto o VNFM também cuida do gerenciamento de recursos virtuais e do gerenciamento de informações/ciclo de vida/falhas no contexto de VNFs.

Em conclusão, podemos resumir as funcionalidades do NFVO como o gerenciamento (direto e/ou indireto) de NSs, VNFs e NFVI, incluindo as relações entre eles, suas várias instâncias e seus diferentes tipos, por meio da coordenação com os outros dois componentes principais do MANO (VNFM e VIM) (GONZALEZ et al., 2018).

2.2 Containers

Os avanços no kernel do Linux levaram ao desenvolvimento da virtualização baseada em *containers*. Um *container* do Linux (LXC) possui uma abordagem diferente de um *hipervisor* e pode ser usado como uma alternativa para virtualização baseada em *hipervisor*. A contêinerização do Linux é uma abordagem na qual se pode executar vários processos de maneira isolada (Joy, 2015), onde é usado apenas um *kernel* para vários ambientes isolados ou sistemas operacionais. Os principais fatores de isolamento para containers são dois recursos do Linux chamados *namespaces* e *cgroups* (CONTAINERS, 2019). Além desses fatores, os LXCs são muito leves porque não virtualizam o *hardware*, em vez disso, todos os *containers* no *host* físico usam o único *kernel* do *host* de maneira eficiente, usando o isolamento do processo.

Figura 2.3: Comparação Virtual Machine e Container



Fonte: O Autor

A Figura 2.3 mostra os componentes na tecnologia de *container*, em comparação

com *virtual machine* (VM). Cada aplicação em um *container* consiste em *Application* e *Libraries* que são compactadas independentemente. Cada *container* é isolado e consiste em seu próprio subsistema independente de rede, memória e sistema de arquivos. O *Container Engine* gerencia esses *containers* (Joy, 2015). Como os *containers* compartilham o mesmo SO, isso permite que centenas de *containers* sejam executados em um único sistema operacional. Isso proporciona ao *container* o benefício de uma implantação rápida com desempenho quase nativo na CPU, memória, disco e rede.

Uma área em que os *containers* são menos eficazes que as VMs é no isolamento. As VMs podem tirar proveito do isolamento de *hardware*, como o fornecido pelas tecnologias VT-d e VT-x da Intel (Che et al., 2010). Esse isolamento impede que as VMs interfiram entre si. O *hipervisor* da VM é colocado acima do sistema operacional *host*, que orquestra os sistemas *guest*. Os sistemas operacionais *guest* são colocados sobre o *hipervisor*. Todas essas complexidades são evitadas com uma tecnologia de *containers*. Diferente das VMs os *containers* são colocados diretamente sobre o sistema operacional do *host* e um *Container Engine* para gerenciar os *containers*.

O objetivo dos *containers* é criar essa independência, a habilidade de executar diversos processos e aplicativos separadamente para utilizar melhor a infraestrutura e, ao mesmo tempo, manter a segurança que se teria em sistemas separados (Joy, 2015).

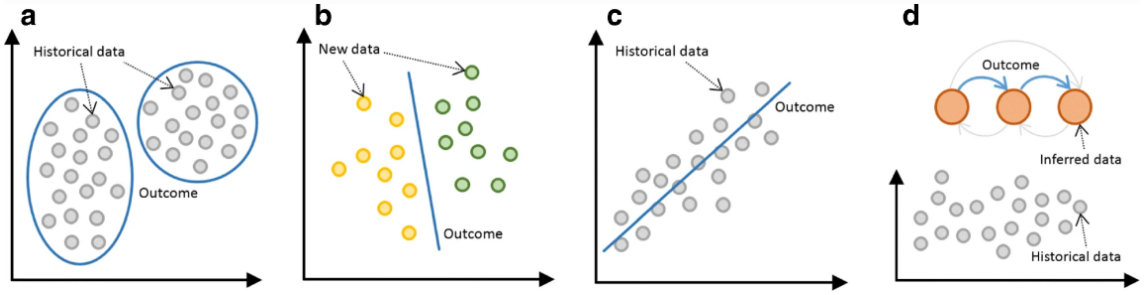
2.3 Supervised Learning

Em 1959, Arthur Samuel cunhou o termo "*Machine Learning* (ML)", como o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados (PUGET, 2016). Existem quatro categorias de problemas que podem se beneficiar do ML: agrupamento, classificação, regressão e extração de regras (BROWNLEE, 2018). Em problemas de agrupamento, o objetivo é agrupar dados semelhantes, enquanto aumenta a diferença entre os grupos. Em problemas de classificação e regressão, o objetivo é mapear um conjunto de novos dados de entrada para um conjunto de saída com valor discreto ou contínuo, respectivamente. Já os problemas de extração de regras são intrinsecamente diferentes, onde o objetivo é identificar relacionamentos estatísticos nos dados (BOUTABA et al., 2018).

Geralmente, o ML é ideal para inferir soluções para problemas que possuem um grande conjunto de dados representativos. Dessa forma, conforme ilustrado na Figura 2.4, as técnicas de ML são projetadas para identificar e explorar padrões ocultos nos dados.

A classificação de anomalias de rede podem ser formulados como um desses problemas que podem se beneficiar do ML. Por exemplo, um problema de mitigação de ataque na rede pode ser formulado para prever os tipos de ataque: *Denial-of-Service (DoS)*, *SQL Injection (Web Attack)*, *PortScan* ou *Infiltration*.

Figura 2.4: Categorias de problemas que se beneficiam do ML. **a** Agrupamento. **b** Classificação. **c** Regressão. **d** Extração de Regra



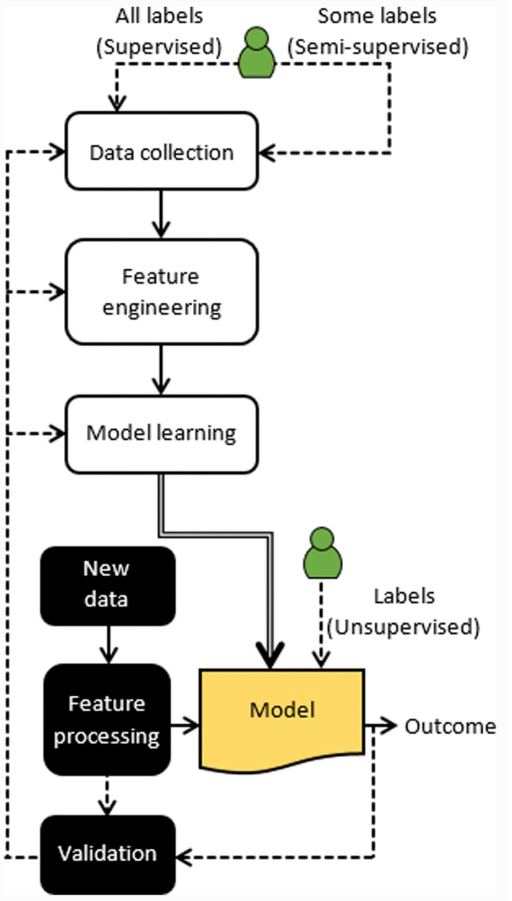
Fonte: (BOUTABA et al., 2018)

Embora existam diferentes categorias de problemas que desfrutam dos benefícios do ML, existe uma abordagem genérica para criar soluções baseadas em ML. A Figura 2.5 ilustra os principais componentes do projeto de soluções baseadas em ML para redes. O *Data collection* refere-se à coleta, geração e/ou definição do conjunto de dados e do conjunto de classes de interesse. A *Feature engineering* é usada para reduzir a dimensionalidade dos dados e identificar recursos discriminantes que reduzem a sobrecarga computacional e aumentam a precisão. Finalmente, as técnicas de ML analisam cuidadosamente os dados e aprendem com o resultado.

O *Supervised Learning* (SL) é uma técnica de aprendizado que é empregada para *"aprender"* a identificar padrões ou comportamentos nos conjuntos de dados de treinamento *"conhecidos"*. SL usa conjuntos de dados de treinamento rotulados para criar modelos. Nele existem vários métodos para rotular conjuntos de dados conhecidos como verdade básica. Normalmente, essa abordagem é usada para resolver problemas de classificação e regressão relacionados à previsão de resultados com valor discreto ou contínuo, respectivamente. O SL é alimentado com uma coleção de instâncias de amostra, pré-classificadas em classes. A saída do processo de aprendizagem é um modelo de classificação que é construído examinando e generalizando a partir das instâncias fornecidas (Nguyen; Armitage, 2008).

O SL se concentra na modelagem dos relacionamentos de entrada/saída e seu objetivo é identificar um mapeamento de recursos de entrada para uma classe de saída. O conhecimento aprendido (por exemplo, pontos em comum entre membros da mesma classe

Figura 2.5: Componentes das soluções baseadas em ML.



Fonte: (BOUTABA et al., 2018)

e diferenças entre os concorrentes) pode ser apresentado como um fluxograma, uma árvore de decisão, regras de classificação, etc., que podem ser usados posteriormente para classificar uma nova instância.

Existem duas etapas principais no SL:

- *Treinamento*: A fase de aprendizado que examina os dados fornecidos (chamados de conjunto de dados de treinamento) e constrói um modelo de classificação.
- *Teste* (também conhecido como classificação): O modelo que foi construído na fase de treinamento é usado para classificar novas instâncias.

Por exemplo na Equação 2.1, TS é um conjunto de dados de treinamento, ou seja, um conjunto de entrada/saída, em que x_i é o vetor de valores dos recursos de entrada correspondentes à i^{th} e y_i é o valor da classe de saída. O nome aprendizado supervisionado deriva do fato de que as classes de saída são predefinidas no conjunto de dados de treinamento. O objetivo da classificação pode ser formulado da seguinte maneira: a partir de um conjunto de dados de treinamento TS, encontre uma função *f(x)* dos recursos de entrada

que melhor prediz o resultado da classe de saída *y* para quaisquer novos valores invisíveis de *x*. Nesse caso, a saída assume seu valor em um conjunto discreto $\{y_1, y_2, \ldots, y_M\}$ que consiste em todos os valores de classe predefinidos. A função *f(x)* é o núcleo do modelo de classificação (Nguyen; Armitage, 2008).

$$\mathrm{TS} = \{< x_1, y_1 >, < x_2, y_1 >, \ldots, < x_N, y_M >\} \tag{2.1}$$

O modelo criado durante o treinamento é aprimorado se fornecermos simultaneamente exemplos de instâncias que pertencem a classes de interesse e instâncias conhecidas por não serem membros das classes de interesse. Isso aumentará a capacidade de o modelo identificar instâncias pertencentes a classes de interesse.

3 TRABALHOS RELACIONADOS

Alguns estudos recentes investigaram a identificação de qual VNF deve ser instanciada, a necessidade de novas instâncias de uma mesma VNF e o dimensionamento nas *Service Chains*. Neste capítulo, são apresentados trabalhos que abordam estes conceitos.

Schardong, Nunes and Schaeffer-Filho (2018) propõem um mecanismo baseado em leilões, no qual agentes fazem lances sobre a alocação de VNFs. Agentes cognitivos são incorporados com preferências de um especialista de domínio, permitindo que eles decidam quais ações devem ser tomadas com base nas percepções do ambiente. A forma de virtualização proposta é utilizando *containers*, a orquestração se limita a serviços de NF, instanciando novas funções de forma manual, prevendo qual NF é necessária de forma estática e utilizando a técnica de BDI (crença-desejo-intenção).

Zhang et al. (2017) apresentam um mecanismo baseado em uma adaptação das técnicas de aprendizado online para prever futuras cargas de trabalho das *Service Chains*, possibilitando estimar efetivamente as próximas taxas de tráfego e ajustar a implantação proativa da VNF, propondo a utilização de VMs para virtualização, com orquestração somente de recursos das VNFs. Nessa estratégia, a instanciação de novas funções também é realizada de forma manual, identificando qual NF é necessária de forma estática e empregando técnicas de *Online Learning* e *Online Optimization*, implementadas com base no algoritmo *Adaptive Online Gradient Descent (OGD)*. Baseado nas técnicas e algoritmo mencionado é implementado o POLAR (*Proactive OnLine AlgoRithm*) que combina um módulo de aprendizagem online e um algoritmo de compra de VM online, onde são alocadas as VNFs.

Fei et al. (2018) buscam uma abordagem proativa para provisionar novas instâncias para VNFs sobrecarregadas com antecedência com base nas taxas de fluxo estimadas, propondo um método eficiente de aprendizado online que visa minimizar o erro na previsão das demandas das *Service Chains*, utilizando assim, VMs para virtualização. Nesta abordagem ocorre a orquestração somente dos serviços, realizando a instanciação de novas funções de forma automática, identificando qual NF é necessária de forma estática e propondo um algoritmo baseado em um método de *Online Learning* (ANDERSON, 2008) chamado *The Follow the Regularized Leader (FTRL)*. Este algoritmo proposto possibilita o provedor de NFV aprender a dimensionar instâncias de VNF com capacidades adaptativas para lidar com flutuações de carga de trabalho, com a ajuda de previsão de demanda eficaz.

Zhou and Guo (2017) propõem uma estrutura de mitigação de ataques DDoS com NFV e SDN, que permite alocação e implantação flexíveis de recursos. A solução incorpora recursos de monitoramento de eventos, coleta de informações e análise de dados para facilitar a detecção de atividades anômalas. A solução proposta utiliza VMs para virtualização, orquestrando serviços e recursos, realizando a instanciação de novas funções de forma automática, identificando qual NF é necessária de forma estática e propondo a implementação de um módulo de *Anomaly Detection*, que através do controlador SDN faz o monitoramento de tráfego, coleta de dados e funcionalidades de análise, e é responsável por acionar os mecanismos de mitigação de ataque *DDoS* apropriados.

Pattaranantakul et al. (2016) relatam brevemente uma análise de ameaças no contexto de NFV e identificam os requisitos de segurança correspondentes, com o objetivo de estabelecer uma taxonomia abrangente de ameaças e fornecer uma diretriz para desenvolver contramedidas eficazes de segurança. Assim apresentam uma estrutura de projeto conceitual (SecMANO) para gerenciamento de segurança baseado em NFV e orquestração de serviços, com o objetivo de implantar e gerenciar de forma dinâmica e adaptativa as funções de segurança de acordo com as demandas dos usuários e clientes. Para tal são utilizadas VMs para virtualização, realizando orquestração dos serviços e recursos, realizando a instanciação de novas funções de forma automática e identificando qual NF é necessária de forma estática através da definição de políticas de controle.

Alawe et al. (2018) propõem uma solução para prever a evolução do tráfego na rede com base em redes neurais, buscando escalabilidade para atender às necessidades, em termos de provisionamento de recursos, sem degradar a Qualidade de Serviço (QoS). A solução trata de generalizar redes neurais enquanto acelera o processo de aprendizado usando o agrupamento K-means e um método de Monte-Carlo. Com esta solução são capazes de prever a carga futura combinada com um orquestrador, assim oferecendo provisionamento dinâmico e proativo de recursos. Além disso, combinando essas técnicas, foi possível prever com eficiência a evolução do tráfego, mesmo quando a escala do tráfego muda. A técnica proposta também tem a vantagem de tornar o aprendizado muito mais rápido, graças ao uso da técnica de agrupamento K-means nos dados. No entanto, não é mencionado a forma de virtualização das NFs, pressupondo que foi utilizada VMs por serem no momento a escolha padrão.

Bondan (2019) propõem um mecanismo de detecção de anomalias para orquestradores NFV através da análise da operação de elementos NFV, possibilitando a classificação das ameaças para ajudar operadores de rede a projetar e empregar as contramedidas

mais adequadas para ambientes NFV. A arquitetura proposta permite o design e a implementação de diferentes mecanismos de detecção de anomalias em ambientes NFV. O modulo proposto, o *NFV Security Module* (NSM) é validado através da implementação e avaliação de diferentes mecanismos de detecção de anomalias, projetados para lidar com informações heterogêneas. Os resultados obtidos mostram a eficácia dos mecanismos projetados em face dos conjuntos de dados realísticos de SFCs e VNFs, alcançando precisões acima de 95% e comprovando a viabilidade de usar o NSM como framework para detecção de anomalias em ambientes NFV. A técnica proposta para detecção de anomalias é a *Shannon's Information Entropy-based*. No entanto, não é mencionado a forma de virtualização das NFs, pressupondo que foi utilizada VMs por serem no momento a escolha padrão.

Os trabalhos citados na sua maioria tratam do provisionamento de novas instâncias de VNFs pré-existentes, utilizando virtualização baseada em máquinas virtuais, não tratando da identificação de qual VNF deve ser instanciada e nem do seu ciclo de vida. Além disso, nem todos são voltados para aspectos de segurança, como mitigação de ataques. Os trabalhos voltados para mitigação de ataques e anomalias na rede tratam somente de alguns cenários específicos, como a identificação de uma categoria de ataque ou da mitigação em ambientes específicos, como NFV. Diferentemente, nesta dissertação, é proposto a utilização de aprendizado supervisionado para identificação de anomalias e seleção dinâmica de VNFs de mitigação de diversas categorias de ataques e independente do ambiente de rede, gerenciando de forma automática o ciclo de vida das VNFs, e também usando contêineres como virtualização para uma instanciação mais rápida e com menor uso de recursos. Na Tabela 3.1 são apresentadas as principais questões que cada trabalho se propõe resolver.

Tabela 3.1: Tabela comparativa de trabalhos relacionados

Trabalhos	Virtualização	Orquestração	Instanciação	Predição de VNF	Segurança	Técnica Empregada
(Schardong; Nunes; Schaeffer-Filho, 2018)	Container	Serviço	Manual	Estática	Sim	Belief Desire Intention (BDI)
(Zhang et al., 2017)	VM	Recurso	Manual	Estática	Sim	Adaptive Online Gradient Descent (OGD)
(Fei et al., 2018)	VM	Serviço	Automática	Estática	Não	Follow the Regularized Leader (FTRL)
(Zhou; Guo, 2017)	VM	Serviço / Recurso	Automática	Estática	Sim	Traffic monitoring, data collection and analysis functionalities in SDN controller
(Pattaranantakul et al., 2016)	VM	Serviço / Recurso	Automática	Estática	Sim	Políticas de Controle
(Alawe et al., 2018)	VM	Recurso	Automática	Estática	Não	Convolutional Neural Networks (CNN), Long Short Term Memory (LSTM), K-means clustering e método de Monte-Carlo
(BONDAN, 2019)	VM	Serviço	Automática	Estática	Sim	Shannon's Information Entropy-based

Fonte: O Autor

A Tabela 3.1 está divida em sete colunas, a primeira referenciando os trabalhos e a partir da segunda as principais características e funcionalidades. A segunda coluna identifica como é feita a virtualização nos trabalhos relacionados, por exemplo, máquina virtual (VM), container, unikernel, etc. Já a terceira coluna indica o tipo de orquestração que é utilizada, orquestração de serviços ou recursos, orquestração de NF ou de recursos da NFVI, respectivamente. Na quarta coluna é caracterizado o tipo de instanciação e identifica se a instância de uma nova NF é feita de forma manual por um operador ou através de um algoritmo de forma automática. A quinta coluna identifica predição de VNF, que é a funcionalidade que trata de prever qual NF deve ser instanciada, por exemplo, a predição estática é quando o operador de rede seleciona qual VNF deve ser instanciada para mitigar determinado ataque, já a predição dinâmica é quando utiliza informações coletadas sobre o ataque para decidir a VNF de mitigação. Já na sexta coluna é identificado se a solução proposta é voltada para tratar algo relacionado a segurança, com o aspecto de mitigação de ataques. Na sétima e última coluna a técnica empregada é apresentada, que busca identificar qual técnica é utilizada para desenvolver o algoritmo utilizado.

4 INTELLIGENT ORCHESTRATION OF CONTAINERIZED NETWORK FUNCTIONS FOR ANOMALY MITIGATION

Nesse capítulo é descrito o Intel-OCNF, um mecanismo para orquestração de VNFs baseado em aprendizado supervisionado, que busca manter a resiliência da infraestrutura de um provedor de serviços de nuvem. O Intel-OCNF tem o objetivo de identificar anomalias no tráfego de rede e implantar VNFs à priori para garantir a qualidade do serviço e minimizar custos de recursos. Nas próximas seções será detalhada a proposta, começando pelo cenário de uso, a visão geral de como funcionará a instanciação de funções sob demanda e por fim a especificação dos componentes que compõem a arquitetura do Intel-OCNF.

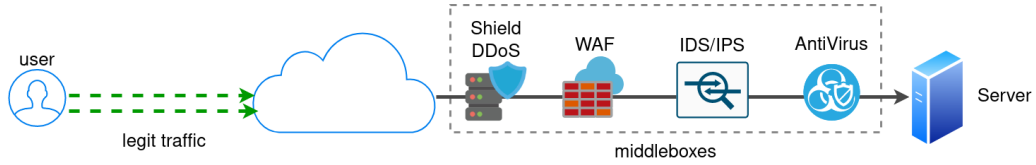
4.1 Cenário de Uso

Em geral, as soluções comerciais de segurança disponíveis são baseadas principalmente em assinaturas, nas quais o tráfego de rede é comparado a padrões de ataques predeterminados (PALOAUTO-NETWORKS, 2020), (TREND-MICRO, 2020). Geralmente essas soluções oferecem respostas automatizadas à intrusão, mas em muitos casos, exigem que um operador interprete um comportamento anômalo e tome ações para mitigá-lo. Além disso, elas também não oferecem o mesmo nível de automatização como o abordado neste trabalho.

Atualmente, as empresas estão migrando cada vez mais para serviços de infraestrutura na nuvem, forçando uma mudança na maneira como as redes corporativas são construídas, resultando em serviços ainda mais necessários para manter a segurança e a resiliência destes ambientes. Como alternativa, buscando diminuir os custos em OPEX e CAPEX, a virtualização de serviços de rede representa uma grande oportunidade de baixo custo para as empresas, e de negócios para os provedores de serviços. Com base neste cenário, esta dissertação propõe que ao contratar um serviço de infraestrutura uma empresa terá ao seu dispor serviços baseados em NFV que possibilitam instanciar funções de rede conforme a necessidade. Para isso o provedor disponibilizará também um serviço que identifica possíveis fluxos maliciosos de forma automatizada, podendo reconhecer um ataque e instanciar funções para mitigá-lo. A disponibilização de VNFs para a empresa como serviço é comparável à abordagem de *Software as a Service (SaaS)*.

Na Figura 4.1 pode ser visto de forma ilustrativa o funcionamento de uma solução tradicional, onde *middleboxes* são utilizadas.

Figura 4.1: Exemplo de caso de uso solução tradicional



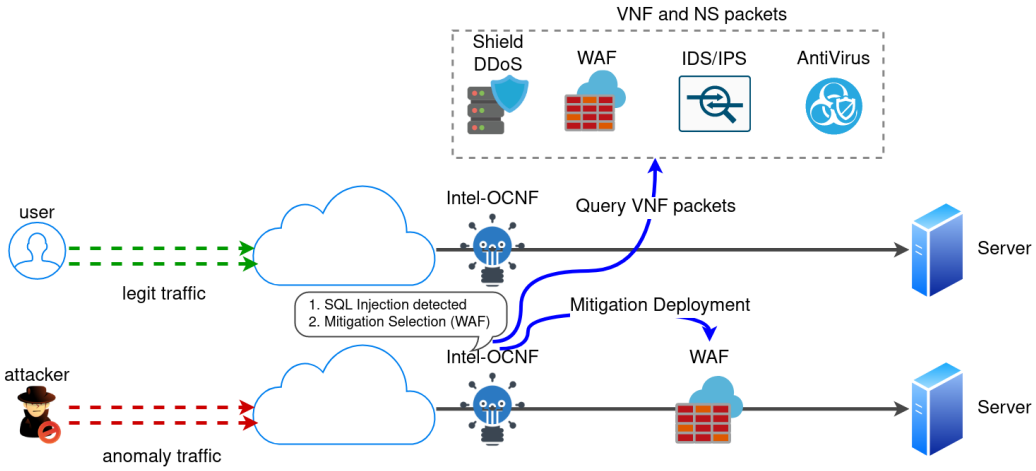
Fonte: O Autor

No modelo tradicional vemos em funcionamento a solução onde a empresa contrata serviços que julga necessários para manter a resiliência de seu ambiente. Estes serviços estão habilitados e mesmo que não estejam em uso ficarão alocados e gerando custos.

Já na Figura 4.2 é utilizada a solução proposta com NFV, que é diferente da solução tradicional nos seguintes aspectos:

- Utilização de técnicas de *machine learning* para classificar anomalias no tráfego de rede e determinar o conjunto ideal de funções virtualizadas para mitigação;
- Emprego de virtualização baseada em *containers* para agilizar o processo de instanciação e gerenciamento;
- Monitoramento das condições da rede para determinar quais funções manter.

Figura 4.2: Exemplo de caso de uso solução NFV



Fonte: O Autor

Para exemplificar o funcionamento da solução proposta, foi ilustrado como é a composição de uma infraestrutura com os serviços de mitigação de anomalias. No modelo tradicional, quando identificadas as possíveis anomalias, os serviços de mitigação

são selecionados por um operador. Serviços como um *WAF*, um *Shield DDoS*, um *Antivírus* ou um *IDS/IPS* poderiam ser selecionados, dependendo do tipo de anomalia. Na solução proposta a identificação de anomalias e a seleção de mitigadores são feitas de forma automatizada.

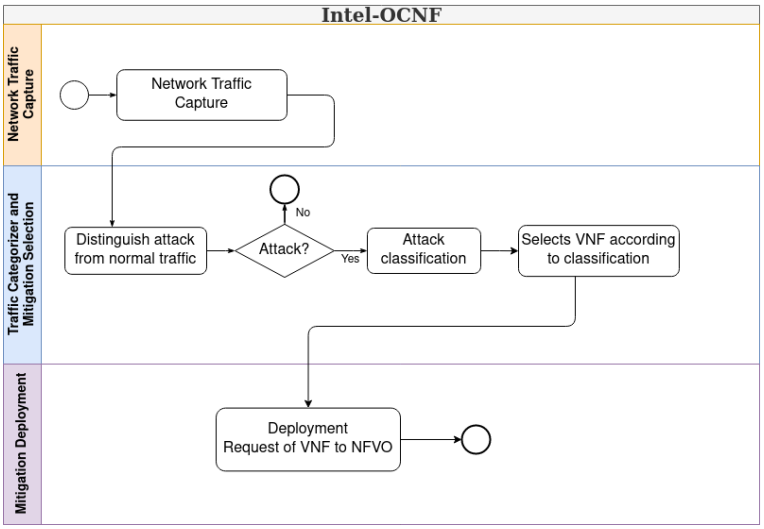
Um exemplo de caso de uso, seria o Intel-OCNF identificar uma anomalia no tráfego que explora uma vulnerabilidade de *SQL Injection* em determinada aplicação. Isso levaria por exemplo à seleção de uma VNF de *WAF* capaz de mitigar o ataque, e o correspondente *deploy* desta função. O Intel-OCNF também é responsável pelo gerenciamento do ciclo de vida da VNF, identificando por exemplo se foi detectada uma anomalia na qual já existe uma VNF de mitigação instanciada e se ela não é mais necessária ou precisa ser escalada. No modelo proposto, a empresa pagaria somente sob demanda pelo consumo de recursos de computação utilizados para executar as instâncias de VNFs. Com isso ela teria custos somente quando necessitasse instanciar um serviço, diminuindo os custos de ter um ou mais profissionais especializados para identificar a necessidade de determinado serviço e com alocação de recursos que não estão em uso integralmente. Nesse contexto, o provedor de serviços através do Intel-OCNF é responsável por identificar a necessidade de novas VNFs, implantar, configurar, atualizar e gerenciar a operação das instâncias para garantir a resiliência do ambiente.

4.2 Visão Geral da Estratégia: Funções de Resiliência Sob Demanda

Em um cenário onde podemos automatizar medidas de mitigação a possíveis ataques que podem impactar a resiliência da infraestrutura, instanciando funções sob demanda e de baixo custo, o Intel-OCNF vem com a proposta de se integrar às implementações existentes de NFVO. Seu objetivo é fornecer orquestração inteligente de funções de rede para mitigação de anomalias, utilizando virtualização baseada em contêineres para minimizar o uso de recursos, e agilizar o processo de instanciação e gerenciamento. Na Figura 4.3 é ilustrada uma visão resumida do fluxo de funcionamento do Intel-OCNF até se integrar ao NFVO. O funcionamento do mecanismo inicia pela captura do tráfego de rede, em seguida o tráfego é encaminhado para o componente que separa o que é tráfego normal do que pode ser um tráfego anômalo. Logo após, é feita a classificação, assim, é possível identificar qual tipo de ataque está ocorrendo, selecionar a VNF necessária para mitigar e requisitar ao orquestrador o *deploy* desta função.

Para automatizar o processo de instanciação de novas VNFs, o componente Intel-

Figura 4.3: Visão geral do fluxo



Fonte: O Autor

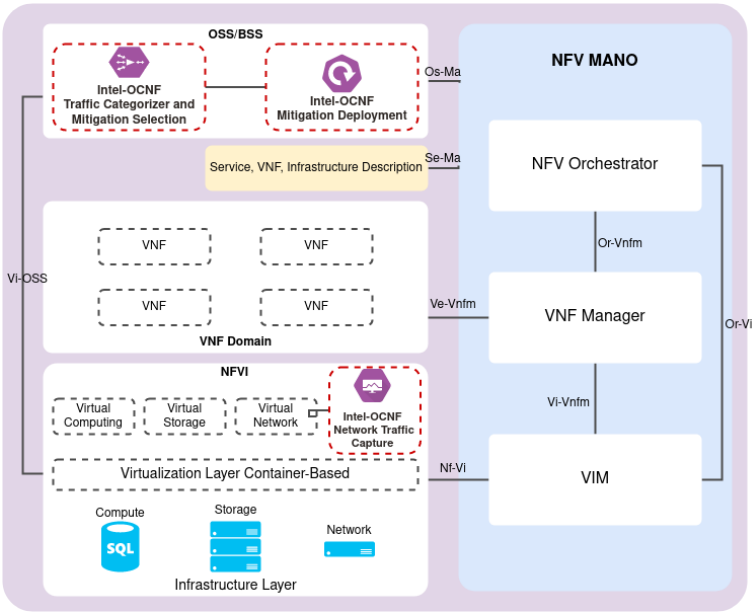
OCNF estende as funcionalidades do NFVO através das APIs disponíveis por implementações baseadas nas especificações da arquitetura de APIs RESTful da NFV-MANO *NFV-SOL-005* (Chatras, 2018), conforme ilustrado na Figura 4.4, passando a ser o responsável por identificar quais funções de rede devem ser instanciadas e requisitar o *deploy* das mesmas. Esta identificação se dá através da coleta de informações do tráfego de rede, que passa por análises capazes de identificar possíveis anomalias e determinar qual VNF é necessária instanciar. Estas análises podem ser feitas utilizando inúmeras técnicas, no entanto, nesta dissertação é proposto a utilização da técnica de aprendizado de máquina *supervised learning*, pois em trabalhos como de Salman et al. (2017) é comprovada sua eficiência em aplicações de categorização de anomalias em ambientes *multi-cloud*, assim como em sistemas de detecção de intrusão (MODI et al., 2013).

Na arquitetura especificada pela ETSI os componentes que integram o Intel-OCNF são distribuídos da seguinte forma:

- *Network Traffic Capture*: componente que fica na camada da NFVI conectado a *Virtual Network* para capturar o tráfego de rede;
- *Traffic Categorizer and Mitigation Selection*: componente externo, funciona independente da arquitetura da ETSI, pode ficar no OSS;
- *Mitigation Deployment*: componente que se conecta via API à arquitetura MANO da ETSI, pode ficar no OSS.

Através da integração com o orquestrador NFV é possível por exemplo requisitar a criação de uma nova VNF, instanciar uma nova VNF, escalar uma instância de uma

Figura 4.4: Integração com NFVO



Fonte: O Autor

VNF de forma incremental ou até encerrar uma instância de uma VNF. Estes recursos são implementados através de APIs, baseadas na especificação da ETSI de protocolos RESTful, para gerenciamento do ciclo de vida das *Network Services* (ETSI, 2020). Desta forma é possível integrar o Intel-OCNF a qualquer implementação de NFV MANO que siga as especificação da ETSI.

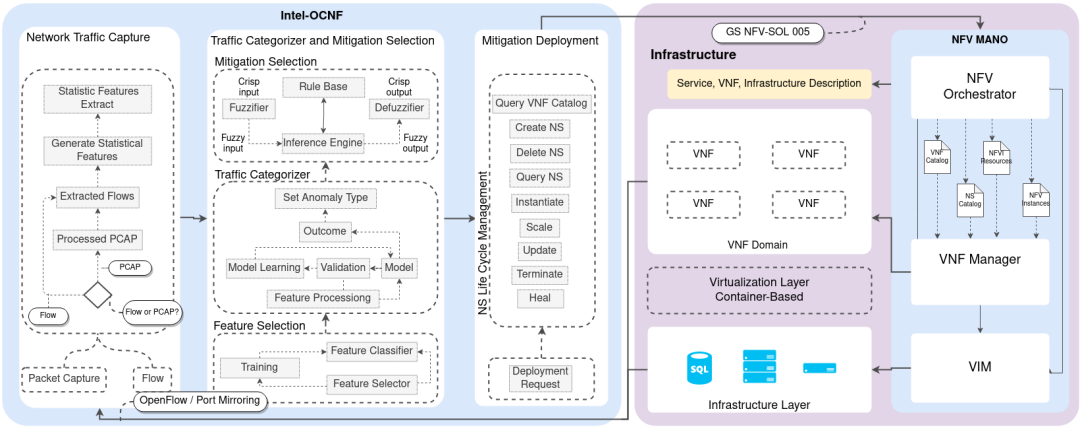
4.3 Arquitetura do Intel-OCNF

Nesta seção é apresentada a arquitetura proposta, detalhando o Intel-OCNF, responsável por aplicar técnicas de aprendizado de máquina para detectar possíveis anomalias na rede e identificar qual VNF deve ser instanciada. Esta arquitetura é ilustrada na Figura 4.5, exemplificando como pode ser feita a captura do tráfego de rede, a categorização do tráfego, seleção e *deploy* de mitigadores, e a integração com a arquitetura NFV da ETSI (ETSI, 2013a).

4.3.1 Network Traffic Capture

O *Network Traffic Capture* é responsável por fazer a captura do tráfego de rede, sendo que esta captura pode ser feita através do controlador *OpenFlow* ou de *Packet Cap-*

Figura 4.5: Detalhamento da arquitetura proposta



Fonte: O Autor

ture com o conceito de *Port Mirroring*, para executar tal operação ele é composto pelos submódulos *Processed PCAP* e *Extracted Flows*. Abaixo são detalhados os submódulos e em quais abordagens eles são utilizados.

- *Processed PCAP*: Com a utilização de *Packet Capture* é capturado o tráfego em formato de pacotes, em seguida é feito o processamento, onde é realizada a conversão para tráfego em fluxos. O processamento das características dos arquivos PCAP é feito utilizando métodos de geração de fluxo, que resumem o tráfego utilizando os cabeçalhos. Estes métodos coletam informações de pacotes com características comuns, como endereços IP e números de portas, agregam-os em fluxos e calculam algumas estatísticas, como o número de pacotes por fluxo. Os métodos utilizados se baseiam nas RFC 2722 e 2724, nas quais é documentado que um fluxo de tráfego é definido como um equivalente lógico para uma chamada ou conexão em associação com um grupo de elementos especificado pelo usuário (BROWNLEE; MILLS; RUTH, 1999) (BROWNLEE, 1999).

- *Extracted Flows*: O *Extracted Flows* é utilizado tanto pela abordagem com *OpenFlow* ou *Packet Capture*. Ele faz a extração das informações dos fluxos, tais como: valor dos contadores de *bytes* e pacotes que são utilizados para a criação do conjunto de características. Além dos contadores, ele também coleta os endereços IP e portas TCP/UDP de origem e destino e o protocolo da camada de transporte, de modo que cada fluxo seja identificado univocamente (Silva et al., 2015).

Com as informações recebidas do *Network Traffic Capture* é gerado o conjunto de características estatísticas de cada fluxo. Para tal, estas informações passam pelo sub-

módulo *Generate Statistical Features*, que gera características estatísticas utilizando contadores presentes nas tabelas de contadores do protocolo *OpenFlow* ou através de um analisador de fluxo bidirecional e pelo submódulo *Statistic Features Extract* que faz a extração das estatísticas geradas.

Abaixo é especificado o funcionamento do *Generate Statistical Features* e do *Statistic Features Extract* com *OpenFlow* e *Packet Capture*:

- *OpenFlow*: Baseado na pesquisa de Silva et al. (2015), para gerar e extrair as características estatísticas de fluxos de tráfego podemos utilizar dois contadores nativos, presentes na tabela de contadores do protocolo *OpenFlow*: *Byte Counter* e *Packet Counter*. Para estender os contadores nativos do *OpenFlow*, são utilizados outros dois valores criados internamente: contador de amostras, que salva a quantidade de vezes que um determinado fluxo foi amostrado na rede; e tempo entre requisições enviadas ao controlador. Com estas informações é possível criar quatro características básicas: *bytes* por segundo; pacotes por segundo; comprimento dos pacotes; e intervalo entre a chegada de dois pacotes. Com o objetivo de obter um maior entendimento sobre o comportamento dos tráfegos, estas quatro características são expandidas em três grupos de características: *Características Escalares*, *Características Estatísticas* e *Características Complexas*.

- *Packet Capture*: Para gerar e extrair as estatísticas é usado um analisador de fluxo bidirecional (Biflow). No *Biflow* gerado, o primeiro pacote determina as direções de avanço (origem ao destino) e retrocesso (destino à origem), portanto, as características estatísticas, como *Duration*, *Number of packets*, *Number of bytes*, *Length of packets*, etc, também são calculadas separadamente na direção destino e origem. Desta forma podem ser extraídas mais de 80 características.

4.3.2 Traffic Categorizer and Mitigation Selection

O *Traffic Categorizer and Mitigation Selection* é o componente responsável por categorizar o tráfego malicioso, buscando identificar o tipo de anomalia. Além de determinar a precisão da detecção de anomalias, ele também distingue diferentes tipos de ataques individualmente para obter uma contramedida e defesa ideal, podendo assim, identificar qual VNF poderá mitigar o ataque detectado.

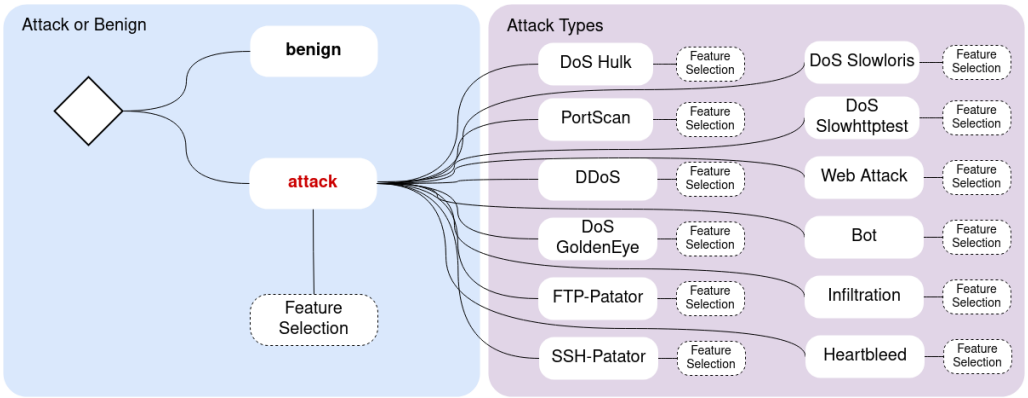
O ponto-chave para a construção de um classificador ML é a identificação do me-

nor conjunto de características necessárias para atingir os objetivos de precisão, um processo conhecido como *Feature Selection* (LI; LI; LIU, 2017). A *Feature Selection* visa encontrar o melhor subconjunto de características que são relevantes para uma tarefa. Neste trabalho é proposto a utilização do algoritmo *Random Forest Regressor* (RFR) baseado em *Randomized Decision Trees* para identificar as melhores características. O algoritmo *Random Forest Regressor* (SEGAL, 2004) é capaz de realizar tarefas de regressão com o uso de árvores de decisão múltiplas e uma técnica chamada *bootstrap* e *aggregation*, comumente conhecida como *bagging*. A ideia básica por trás disso é combinar várias árvores de decisão para calcular o peso de importância das características, em vez de depender de árvores de decisão individuais. Neste processo, na fase de treinamento do algoritmo propomos uma abordagem em duas etapas para otimização da seleção de características (KOSTAS, 2018), conforme ilustrado na Figura 4.6 e descrito abaixo:

- *Seleção de características entre fluxo de ataque e benigno*: Esta abordagem na seleção de características é aplicada para todo o conjunto de dados, coletando todos os tipos de ataques em um único rótulo: "attack". Portanto, as características selecionadas são utilizadas para classificar qualquer tipo de ataque.

- *Seleção de características de acordo com os tipos de ataque*: Após a classificação dos ataques é feita a separação dos fluxos por tipo, isolando um tipo de ataque de outros. Assim, é possível selecionar as características ideais de cada um.

Figura 4.6: Processo de seleção de características



Fonte: O Autor

O conjunto de características a ser usado consiste em combinar as 4 características com o maior peso de importância alcançado para cada ataque. Este conjunto único de características é usado para criar os modelos de detecção de anomalias e seleção da VNF de mitigação. Para tal, são utilizados quatro algoritmos baseados em *supervised*

learning (*Naive Bayes* (NB), *Random Forest* (RF), *K Nearest Neighbours* (KNN) e *Quadratic Discriminant Analysis* (QDA)). Estes algoritmos foram escolhidos considerando sua utilização em diferentes formas de classificação. Por exemplo, normalmente NB é utilizado para classificar frequência das informações, RF para classificação de conjuntos, KNN para medidas de similaridade e QDA para análise discriminante quadrática. Embora vários tipos de técnicas de ML estejam disponíveis, as abordagens de aprendizado supervisionado são as técnicas mais populares e comumente usadas (Saravanan; Sujatha, 2018).

Algorithm 1 Traffic Categorizer and Mitigation Selection Pseudocode

1: ***dataset*** := (x_1, y_1),...,(x_n, y_n) the dataset is labeled with anomalies and their mitigators
2: ***train, test*** := train_test_split(*dataset*) split the dataset into training dataset and testing dataset
3: ***features*** := $(x_1, x_2, x_3$,...)
4: ***algorithms*** := $(a_1, a_2, a_3$,...) list of algorithm that will be applied
5: **function** TRAFFICCATEGORIZER($train, features, algorithms$)
6: $result \leftarrow \theta$
7: **for** $i \in 1$, ..., $algorithms$ **do** apply each algorithm
8: $r_i \leftarrow$ LEARN($train, features, algorithms^{(i)}$)
9: $result \leftarrow result \leq \{r_i\}$
end for
10: $result \leftarrow$ MITIGATIONSELECTION($result[anomaly], result[mitigators]$)
11: **return** $result$
end funcion
12: **function** LEARN($train, features, algorithm$)
13: algorithm($train, features$)
14: **return** The learned
end funcion
15: **function** MITIGATIONSELECTION($anomaly, mitigators$)
16: apply Fuzzy Logic System
17: $result \leftarrow$ [a, $\{v_1, v_2, \ldots, v_N\}$]
18: **return** $result$ The Anomaly and Mitigation VNF
end funcion

Fonte: O Autor

Como abordagem para obter os melhores resultados na classificação, é proposto a utilização de um conjunto de algoritmos, baseado no conceito de *ensemble learning* (KRAWCZYK et al., 2017). De acordo com o pseudocódigo ilustrado no Algoritmo 1, como pré-condição o algoritmo espera um conjunto de dados de treinamento ***dataset*** (linha 1), um conjunto de características ***features*** (linha 3) e a listagem dos algoritmos que serão utilizados ***algorithms*** (linha 4). O *dataset* de entrada é separado entre dados de treinamento e dados de teste (linha 2). O algoritmo funciona da seguinte maneira: serão utilizados quatro algoritmos (Naive Bayes, Random Forest, K Nearest Neighbours e QDA), os mesmos serão aplicados na fase de treinamento dos modelos e será utilizado o algoritmo com melhor resultado obtido após o treinamento (linha 6-9 e linha 12-14). Após o tráfego categorizado é feita a seleção das VNFs de mitigação (linha 15-18) aplicando o conceito de *fuzzy logic* (PAUNOVIĆ et al., 2018), retornando como resultado a anomalia encontrada e as funções de mitigação.

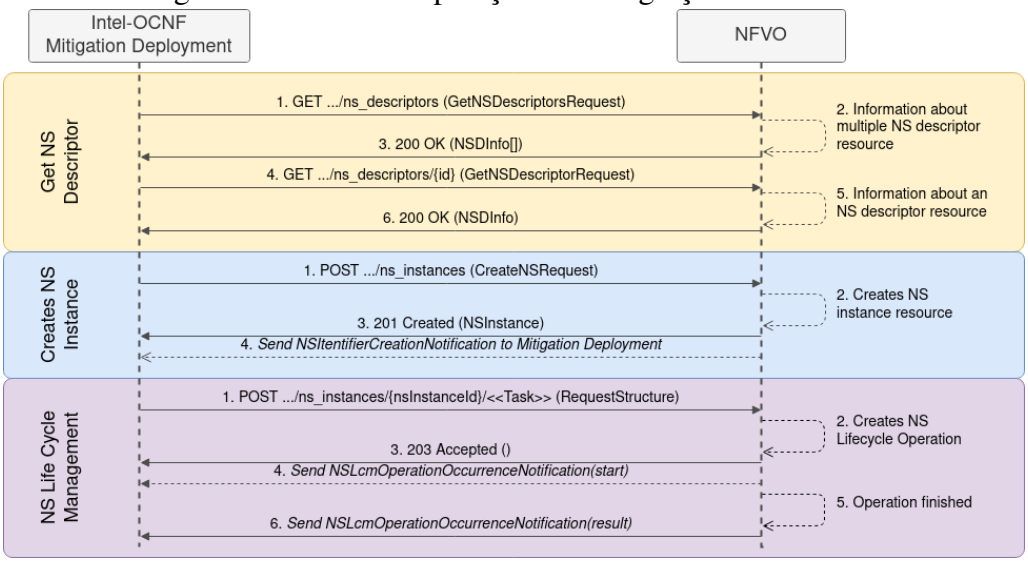
Mais especificamente, com base nos modelos de treinamento, é determinado o tipo

de anomalia e através de um FLS (Fuzzy Logic System) as VNFs de mitigação, p. ex., *S = (a, V) onde V =* $\{v_1, v_2, \ldots, v_N\}$, *a* é a identificação da categoria da anomalia e v_1 a VNF de mitigação. Cada anomalia identificada é submetida ao FLS, compreendendo a etapa de *fuzzification*, responsável por transformar as entradas do sistema em etapas fuzzy; *rule base*, usado para armazenar o conjunto de regras e as condições If-Then usadas para controlar a tomada de decisão; *inference engine*, responsável, por processar as informações; e *defuzzification*, responsável por transformar os valores de saída. O FLS identifica as VNFs que mais se adéquem para mitigar a anomalia, com base nas regras fornecidas por um especialista, p. ex., `IF anomaly=(DoS com grau de incerteza 0.75) THEN vnf=(Shield DoS e Firewall)`, onde `anomaly` pode ser descrita por um conjunto *A=(*$a_1, a_2, ..., a_n$*)*, sendo A, composto por um ou mais termos e seu grau de incerteza, conectados por operadores lógicos AND ou OR, e `vnf` pode ser descrita pelo conjunto *V=(*$v_1, v_2, ..., v_n$*)*, sendo V o conjunto de VNFs de mitigação. Não necessariamente cada anomalia será mitigada por uma única VNF. Por exemplo, identificado um ataque *DoS* que pode ser mitigado por duas VNFs, uma chamada *Shield DoS* e outra chamada *Firewall*, *a = DoS* e *V =* $\{v_1, v_2\}$, onde v_1 é o *Shield DoS* e v_2 é o *Firewall*. Com o resultado da categorização de tráfego, identificação das anomalias e as VNFs de mitigação, é possível instanciar dinamicamente as VNFs.

4.3.3 Mitigation Deployment and NS Life Cycle Management

O *Mitigation Deployment* é o componente que faz a conexão com o NFVO e requisita as ações necessárias para o *deploy* das VNFs. A conexão com o NFVO é realizada através das APIs disponíveis por implementações baseadas nas especificações da arquitetura de APIs RESTful da NFV-MANO *NFV-SOL-005* (Chatras, 2018). Uma das especificações da *NFV-SOL-005* é a interface *NS Life Cycle Management (LCM)*, que implementa o conjunto de ações de gerenciamento do ciclo de vida da NS, desde a criação de uma instância até sua remoção. A Figura 4.7 ilustra a sequência de troca de mensagens da integração entre o componente *Mitigation Deployment* e o *NFVO* durante a criação de uma NS e o gerenciamento do seu ciclo de vida.

Figura 4.7: Fluxo de operações de integração com NFVO.



Fonte: O Autor

4.3.3.1 Get NS Descriptor

O processo de *deploy* é iniciado enviando uma requisição ao NFVO, para consultar os *NS Descriptors (NSD)*. Essa requisição retorna todos os *NS Descriptors* disponíveis. Cada *NS Descriptor* é identificado através de uma *tag* composta por tipo e versão. O *Mitigation Deployment* seleciona o *NS Descriptor* que deve ser utilizado e envia uma nova requisição ao NFVO, passando o *NSD Info ID*. Esta nova requisição busca as informações do *NS Descriptor*, e com estas informações é iniciado o processo de criação da instância da NS.

4.3.3.2 Creates NS Instance

O processo de criação de instâncias da NS é iniciado enviando uma requisição ao recurso *ns_instances*. O corpo da mensagem contém, entre outros itens, uma referência ao *VNF Descriptor* (VNFD) que o *VNF Manager* (VNFM) precisará ler para determinar a sequência de ações a serem executadas. Após a criação da instância da NS é retornando o *NS Instance ID*, utilizado nas próximas requisições para gerenciar o ciclo de vida das *NSs*.

4.3.3.3 NS Life Cycle Management

O ciclo de vida da instância de uma NS é gerenciado enviando requisições aos recursos da *ns_instances* criada. Como podemos ver na Figura 4.7, esse fluxo é aplicável

às seguintes operações, sendo que todas as operações listadas utilizam o mesmo recurso, mudando somente o nome da ação nas marcações em **'«...»'** e os parâmetros enviados:

- *Instantiate NS*: Com este recurso é possível fazer a instanciação de uma NS criada anteriormente. Usando como parâmetro o *InstantiateNsRequest*, conforme definido na *NFV-SOL-005 cláusula 6.5.2.11*;
- *Scale NS*: Com este recurso é possível fazer o dimensionamento de uma instância NS. Usando como parâmetro o *ScaleNsRequest*, conforme definido na *NFV-SOL-005 cláusula 6.5.2.14*;
- *Update NS*: Com este recurso é possível atualizar uma instância NS. Usando como parâmetro o *UpdateNsRequest*, conforme definido na *NFV-SOL-005 cláusula 6.5.2.12*;
- *Heal NS*: Com este recurso é possível recuperar uma instância NS, para casos que necessitam a correção de uma instância. Utiliza como parâmetro o *HealNsRequest*, conforme definido na *NFV-SOL-005 cláusula 6.5.2.13*;
- *Terminate NS*: Com este recurso é possível encerrar uma instância NS. Utiliza como parâmetro o *TerminateNsRequest*, conforme definido na *NFV-SOL-005 cláusula 6.5.2.15*.

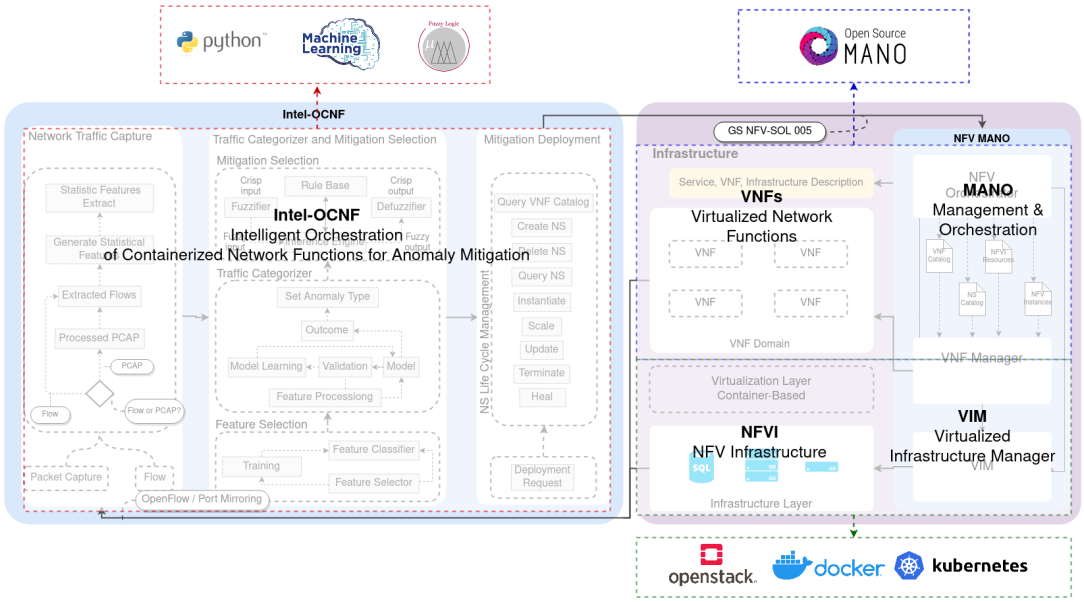
5 PROTÓTIPO E AVALIAÇÃO

Neste capítulo, são detalhados o protótipo e a avaliação do Intel-OCNF, começando pela implementação, definição de cenário, topologia, dataset, e por fim apresentando as métricas e os resultados dos experimentos avaliados.

5.1 Implementação

O protótipo foi construído usando a linguagem *Python*[1], a plataforma *Open Source MANO - OSM v. 8.0.4*[2], e as ferramentas, *Docker*[3], *Kubernetes*[4] e *OpenStack*[5]. Na Figura 5.1 é apresentado o protótipo, ilustrando as tecnologias utilizadas na implementação. Python foi usado para implementar os algoritmos de seleção de características, detecção de anomalias e mitigação. O OSM é usado, pois, implementa a stack MANO e sua plataforma já suporta emulação de VIM (Vim-emu) (Peuster; Karl; van Rossem, 2016) baseado em containers utilizando Docker e Kubernetes, e também implementa a *North Bound Interface (NBI)* API baseada na especificação ETSI SOL005. Os componentes implementados são detalhados na Figura 5.2.

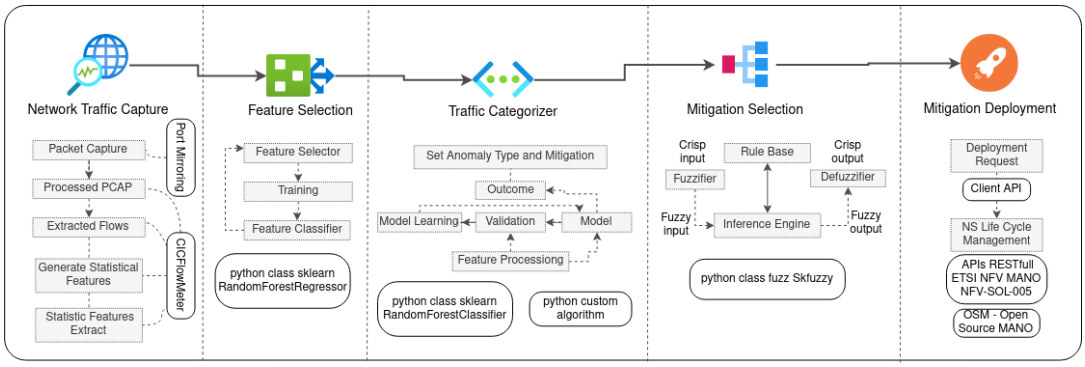
Figura 5.1: Protótipo e definição das tecnologias utilizadas na implementação



Fonte: O Autor

[1] https://www.python.org/
[2] https://osm.etsi.org/
[3] https://www.docker.com/
[4] https://kubernetes.io
[5] https://www.openstack.org/

Figura 5.2: Detalhamento da implementação dos componentes



Fonte: O Autor

Na implementação do componente *Network Traffic Capture*, é utilizado o conceito de *Port Mirror*, espelhando uma porta do *switch* para enviar uma cópia dos pacotes trafegados por ele para o componente, após a captura dos pacotes. Através da ferramenta *CICFlowMeter* (LASHKARI. et al., 2017) é feito o processamento dos PCAP e extraída as informações dos fluxos. Para implementar o algoritmo de *Feature Selection* é utilizada a classe *RandomForestRegressor* da biblioteca Sklearn[6] baseada no algoritmo *Random Forest*. Na implementação do *Traffic Categorizer* são utilizadas as classes *GaussianNB*, *RandomForestClassifier*, *KNeighborsClassifier* e *QDA*, ambas da biblioteca Sklearn baseadas nos algoritmos (Naive Bayes, Random Forest, K Nearest Neighbours e QDA). A *Mitigation Selection* é feita através de um FLS (Fuzzy Logic System) utilizando a classe *Fuzz* da biblioteca Skfuzzy[7]. No *Mitigation Deployment* foi desenvolvido um *client* com a linguagem *Python* para consumir a API do OSM e fazer o gerenciamento do *deploy* e do ciclo de vida das VNFs. O protótipo implementado está disponível publicamente no GitHub[8].

5.2 Avaliação Experimental

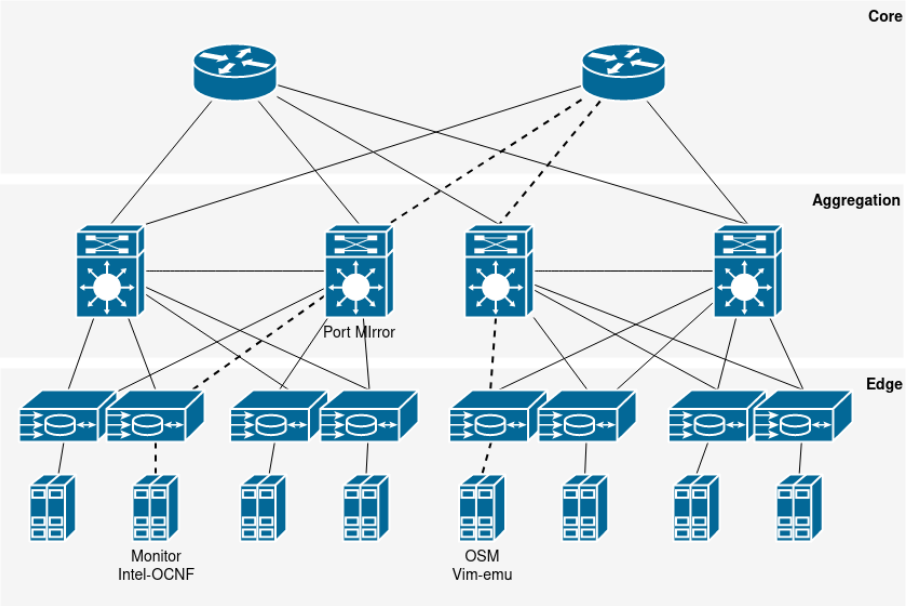
Nesta seção, é apresentada em detalhes a avaliação experimental do protótipo, iniciando com o detalhamento do cenário, topologia, dataset, as métricas aplicadas para medir os experimentos e por fim, trazendo uma avaliação dos resultados.

[6]https://scikit-learn.org/
[7]https://scikit-fuzzy.github.io/scikit-fuzzy/
[8]https://github.com/fernandodebrando/Intel-OCNF

5.2.1 Setup: Cenário, Topologias, Datasets

O cenário utilizado para a avaliação experimental usa dois computadores para emular o ambiente, um para executar o OSM e outro para executar o Intel-OCNF (onde são executados os algoritmos de aprendizado de máquina). O computador utilizado para executar o OSM possui um processador *AMD EPYC 7282 16-Core Processor CPU @ 2.79GHz X 4*, *8 GB* de memória e sistema operacional *Linux Ubuntu 18.04 64-bit*. Já o computador onde é executado o componente Intel-OCNF possui um processador *Intel® Core™ i7-8550U CPU @ 1.80GHz X 8*, *8 GB* de memória e sistema operacional *Linux Fedora 33 (Workstation Edition) 64-bit*. O cenário se baseia na ideia de prover um serviço que identifique possíveis fluxos de tráfego maliciosos em uma rede, reconhecendo um ataque e instanciando funções de rede para mitigá-lo.

Figura 5.3: Topologia utilizada para os experimentos



Fonte: O Autor

Para a realização dos experimentos, foi definida uma topologia *FatTree* conforme Figura 5.3. Esta topologia é composta por um *Monitor* que coleta o tráfego do *switch* e envia para o Intel-OCNF que faz a classificação, identificação do ataque e da VNF que pode mitigá-lo. Como conjunto de dados para realização dos experimentos foi utilizado o *dataset* CIC-IDS2017 (SHARAFALDIN; LASHKARI; GHORBANI, 2018). Este *dataset* foi usado na fase de implementação para treinamento e testes dos algoritmos de aprendizado de máquina. Este conjunto de dados consiste em fluxos de rede rotulados, incluindo cargas úteis de pacotes completos em formato *pcap*. Trata-se de um *dataset*

atualizado, que oferece uma ampla diversidade de ataques e está publicamente disponível para pesquisadores. O conjunto de dados contém 2.830.743 fluxos de tráfego, incluindo diversos tipos de ataques. A distribuição desses fluxos pode ser vista na Tabela 5.1.

Tabela 5.1: Distribuição de registros de fluxo no conjunto de dados CIC-IDS2017

Tipo	Registros	Tipo	Registros
BENIGN	2.359.289	DoS slowloris	5.796
DoS Hulk	231.073	DoS Slowhttptest	5.499
PortScan	158.930	Web Attack	2.180
DDoS	41.835	Bot	1.966
DoS GoldenEye	10.293	Infiltration	36
FTP-Patator	7.938	Heartbleed	11
SSH-Patator	5.897		

Fonte: O Autor

Durante o processo de aprendizado de máquina, os dados são necessários para que o aprendizado ocorra. Além dos dados necessários para o treinamento, os dados de teste são necessários para avaliar o desempenho do algoritmo e determinar como ele funciona. O algoritmo adquire uma habilidade nos dados de treinamento e a aplica aos dados de teste. O resultado dos dados de teste é o desempenho do algoritmo de aprendizado de máquina (GÉRON, 2017). No entanto, o conjunto de dados CIC-IDS2017 usado não contém dados dedicados a treinamento e teste, mas contém um único conjunto de dados desagregado. Portanto, os dados foram divididos em partes de dados de treinamento e teste. Através do conceito de validação cruzada, implementado com a classe *KFold*[9] disponível no *Sklearn*, os dados foram divididos em 5 subconjuntos diferentes. Os subconjuntos foram usados para treinar os algoritmos, sendo que o último subconjunto foi usado como dados de teste. A seguir, é detalhado como foi utilizado o *dataset* para treinamento e testes dos algoritmos de seleção de características e categorização do tráfego. Para otimização da seleção de características são utilizadas duas abordagens, conforme detalhado na Subseção 5.2.1.1 e 5.2.1.2.

5.2.1.1 Seleção de Características entre Fluxo de Ataque e Benigno

Uma abordagem na seleção de característica é aplicar *Random Forest Regressor* a todo o conjunto de dados, classificando todas as categorias de ataques em um único rótulo "attack". Portanto, este conjunto de dados contêm apenas as tags de "attack"e "benign". Como resultado deste processo, a lista de características obtidas por peso de importância é mostrada na Tabela 5.2 e no gráfico da Figura 5.4.

[9]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

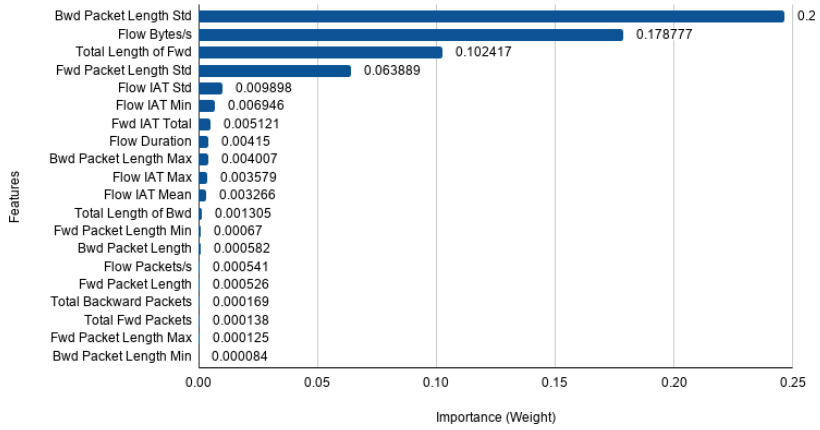
Tabela 5.2: Peso de importância das características de acordo com rótulos *attack* e *benign*

Features	Importance
Bwd Packet Length Std	0.246627
Flow Bytes/s	0.178777
Total Length of Fwd Packets	0.102417
Fwd Packet Length Std	0.063889
Flow IAT Std	0.009898
Flow IAT Min	0.006946
Fwd IAT Total	0.005121
Flow Duration	0.004150
Bwd Packet Length Max	0.004007
Flow IAT Max	0.003579

Features	Importance
Flow IAT Mean	0.003266
Total Length of Bwd Packets	0.001305
Fwd Packet Length Min	0.000670
Bwd Packet Length Mean	0.000582
Flow Packets/s	0.000541
Fwd Packet Length Mean	0.000526
Total Backward Packets	0.000169
Total Fwd Packets	0.000138
Fwd Packet Length Max	0.000125
Bwd Packet Length Min	0.000084

Fonte: O Autor

Figura 5.4: Pesos de importância das características de acordo com rótulos *attack* e *benign*



Fonte: O Autor

5.2.1.2 Seleção de Características de Acordo com os Tipos de Ataque

Nesta abordagem, um conjunto é criado para cada categoria de ataque, isolando um ataque de outros. Estes conjuntos contêm todos os fluxos identificados com a tag "attack"e os fluxos de dados identificados como "benign"selecionado aleatoriamente (30% de ataque, 70% benigno). A distribuição das quatro características com o valor mais significativo para cada ataque pode ser vista na Tabela 5.3.

Quando a distribuição de características é examinada, há uma ou duas que se destacam (*Fwd Packet Length Max* e *Bwd Packet Length Mean*) para quase todas as categorias de ataque. Por outro lado, as características dos ataques Heartbleed e SSH-Patator são bastante diferentes. Para esses ataques, existem algumas características com valores muito próximos uns dos outros, não somente uma ou duas características dominantes, comparados aos outros ataques, como pode ser visto na Figura 5.5.

Por exemplo, no ataque PortScan, duas características (*Flow Bytes/s*, *Total Length of Fwd Packets*) se destacam. Quando o ataque PortScan é examinado, é identificado

Tabela 5.3: Distribuição das quatro características com o valor mais significativo para cada tipo de ataque

Attack / Feature	Importance
Bot	
Bwd Packet Length Mean	0.338129
Flow IAT Max	0.024407
Flow Duration	0.008495
Flow IAT Min	0.007067
DDoS	
Bwd Packet Length Std	0.471368
Total Backward Packets	0.093576
Fwd IAT Total	0.010827
Flow Duration	0.006320
DoS GoldenEye	
Flow IAT Max	0.413073
Bwd Packet Length Std	0.111039
Flow IAT Min	0.054021
Total Backward Packets	0.044895
DoS Hulk	
Bwd Packet Length Std	0.514306
Fwd Packet Length Std	0.069838
Fwd Packet Length Max	0.008542
Flow IAT Min	0.001716
DoS Slowhttptest	
Flow IAT Mean	0.653023
Fwd Packet Length Min	0.101739
Bwd Packet Length Mean	0.029976
Total Length of Bwd Packets	0.007153
DoS slowloris	
Flow IAT Mean	0.473856
Total Length of Bwd Packets	0.057384
Bwd Packet Length Mean	0.053022
Total Fwd Packets	0.021895

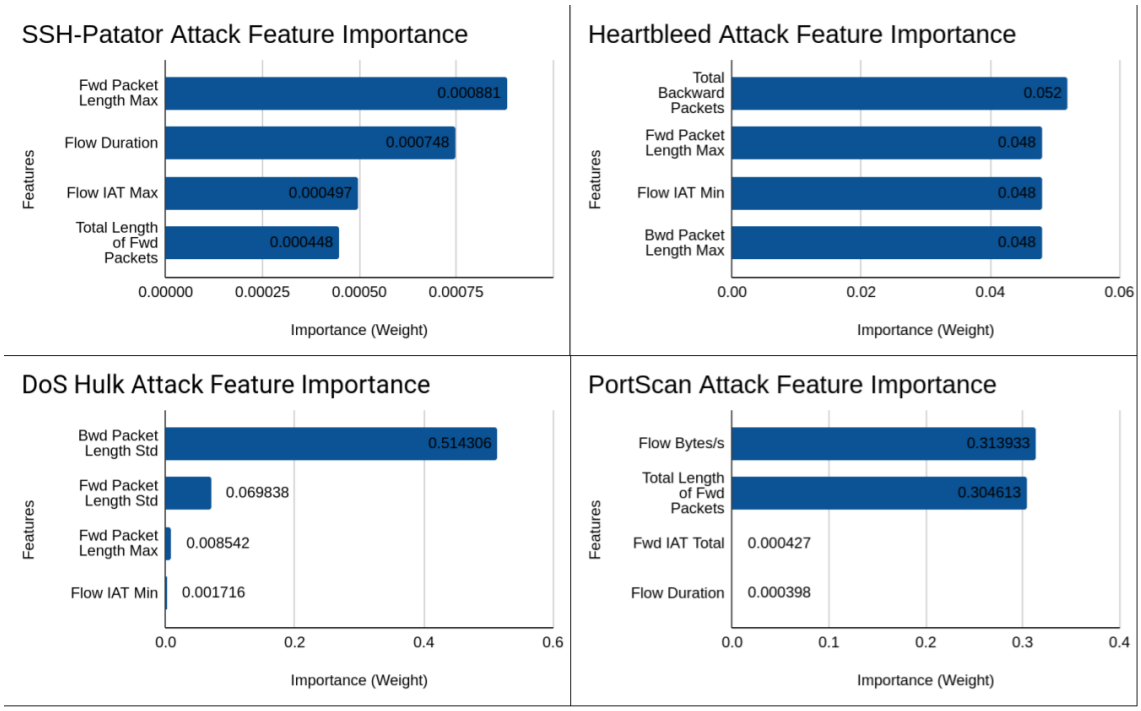
Attack / Feature	Importance
FTP-Patator	
Fwd Packet Length Max	0.063671
Fwd Packet Length Std	0.022751
Fwd Packet Length Mean	0.002179
Total Length of Bwd Packets	0.000746
Heartbleed	
Total Backward Packets	0.052
Fwd Packet Length Max	0.048
Flow IAT Min	0.048
Bwd Packet Length Max	0.048
Infiltration	
Fwd Packet Length Max	0.204751
Fwd Packet Length Mean	0.163696
Flow Duration	0.052250
Total Length of Fwd Packets	0.026823
PortScan	
Flow Bytes/s	0.313933
Total Length of Fwd Packets	0.304613
Fwd IAT Total	0.000427
Flow Duration	0.000398
SSH-Patator	
Fwd Packet Length Max	0.000881
Flow Duration	0.000748
Flow IAT Max	0.000497
Total Length of Fwd Packets	0.000448
Web Attack	
Bwd Packet Length Std	0.007255
Total Length of Fwd Packets	0.006046
Flow Bytes/s	0.003366
Flow IAT Max	0.002102

Fonte: O Autor

que geralmente o comportamento do invasor é enviar o maior número de pacotes possível sem cargas ou com poucas cargas (CHOI; LEE; KIM, 2009). Assim, o atacante acelera o processo e torna o ataque mais eficaz e não consome sua largura de banda desnecessariamente. Neste contexto, espera-se que a característica *Total Length of Fwd Packets* tenha um valor muito baixo quando comparado aos fluxos benignos.

Por outro lado, o ataque SSH-Patator não revela nenhuma característica óbvia. Os valores de importância são muito próximos uns dos outros. A razão mais importante para isso é que o ataque SSH-Patator não consiste em uma única etapa, mas em uma estrutura complexa com três etapas (*Scanning*, *Brute-Force* e *Die-off*) (HELLEMONS et al., 2012). O ataque SSH-Patator contém os ataques PortScan (Scanning/Probe) e Brute-Force.

Figura 5.5: Gráficos de pesos de importância de características de ataques SSH-Patator, Heartbleed, DoS HULK e PortScan



Fonte: O Autor

5.2.1.3 Algoritmo de Categorização do Tráfego

Assim como na seleção de características, duas abordagens diferentes foram usadas para aplicar os algoritmos de aprendizado de máquina ao conjunto de dados. Na primeira abordagem, são utilizados os conjuntos criados na *seleção de características de acordo com os tipos de ataque* (Subseção 5.2.1.2) e as características obtidas na mesma subseção. Esses dados contêm 30% de ataque e 70% de dados benignos e cada um deles nomeados de acordo com a categoria de ataque que o mesmo representa. Na segunda abordagem, todo o conjunto de dados é usado como um único conjunto. Todos os ataques contidos neste conjunto são coletados sob um único nome comum "attack". Portanto, os dados neste conjunto contêm apenas as tags de "attack"e "benign". O conjunto de características a ser usado consiste em combinar as 4 características com o maior peso de importância alcançado para cada ataque. Esta escolha se dá porque as 4 características com maior peso de importância representam, em 9 das 12 categorias de ataques, mais de 95% do peso total da importância das características. Assim, 4 características foram obtidas de cada uma das 12 categorias de ataque, resultando em um grupo de 48 características. Após, foram removidas as repetições, resultando em uma lista de 18 características. A lista dessas características pode ser vista na Tabela 5.4.

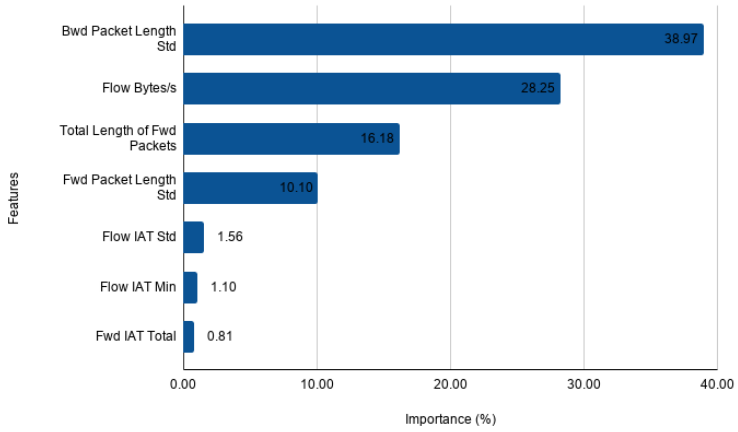
Tabela 5.4: Lista de características para todos os tipos de ataque.

Feature List		
Bwd Packet Length Max	Flow IAT Mean	Fwd Packet Length Min
Bwd Packet Length Mean	Flow IAT Min	Fwd Packet Length Std
Bwd Packet Length Std	Flow IAT Std	Total Backward Packets
Flow Bytes/s	Fwd IAT Total	Total Fwd Packets
Flow Duration	Fwd Packet Length Max	Total Length of Bwd Packets
Flow IAT Max	Fwd Packet Length Mean	Total Length of Fwd Packets

Fonte: O Autor

Outra forma de selecionar as características é usá-las com alta ponderação de acordo com as pontuações de importância obtidas para todo o conjunto de dados na abordagem *seleção de características entre fluxo de ataque e benigno* (Subseção 5.2.1.1), sem usar todas as 18 características. Foi definido um valor limite para o peso das características, em 0,8%. Desta forma, 97% do peso total da importância das características será coberto selecionando apenas 7 características. As 11 características restantes constituem apenas 3% do peso total de significância. Se as características com peso de 0,8% e acima forem selecionadas, as seguintes características serão usadas, conforme Figura 5.6.

Figura 5.6: 7 principais características, com peso acima de 0,8%



Fonte: O Autor

5.2.1.4 Algoritmo de Seleção do Mitigador

Após a identificação dos ataques com a categorização do tráfego, é feita a seleção das VNFs de mitigação, através de um *Fuzzy Logic System*. Primeiro, precisamos definir as entradas e saídas do FLS:

- **Entradas - Ataques**
  - Universo (intervalo de valores *crisp*): qual o nível de pertinência deste ataque,

considerando uma escala entre 0 e 1;
  - Conjunto *fuzzy* (intervalo de valores *fuzzy*): baixo, médio e alto.

- **Saídas - Mitigadores**
  - Universo: qual a VNF de mitigação para determinado ataque, numa escala entre 0 e 100;
  - Conjunto *fuzzy*: baixo, médio e alto.

Como entrada podemos ter um ataque DoS com um intervalo de valores *fuzzy* (baixo, médio e alto) que representam o nível de pertinência, e para cada valor *fuzzy* é definido um conjunto de valores *crisp* em uma escala entre 0 e 1. Por exemplo, `baixo = [0, 0.25, 0.5]`, `medio = [0.3,0.5,0.7]` e `alto = [0.6,0.8,1]`. A representação destes valores pode ser observado na Figura 5.7a. Já, como saída precisamos fornecer os possíveis mitigadores deste ataque DoS, para isto é fornecido um conjunto *fuzzy* (baixo, médio e alto) e para cada valor *fuzzy* é definido um conjunto de valores *crisp* em uma escala entre 0 e 100, que representa o nível de pertinência do mitigador selecionado para o ataque. Por exemplo, `baixo = [0, 25, 45,60]`, `medio = [40, 50, 70,80]` e `alto = [75, 80,95,100]`. Nas Figuras 5.7b e 5.7c pode ser observado a representação destes valores para dois tipos de VNF de mitigação, *Shield DoS* e *Firewall*.

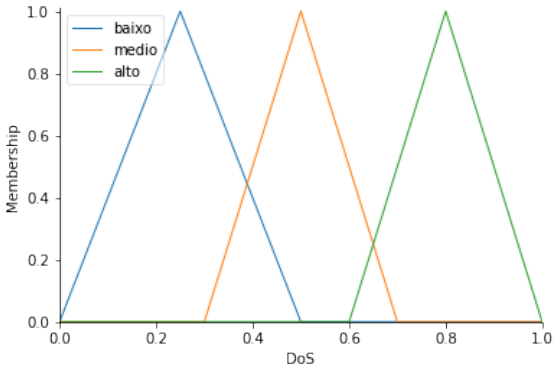
Em seguida são criadas as regras, por exemplo:

- `regra1 = ctrl.Rule(dos['medio'] | dos['alto'], shield_dos['alto'])`
- `regra2 = ctrl.Rule(dos['medio'] | dos['alto'], firewall['alto'])`

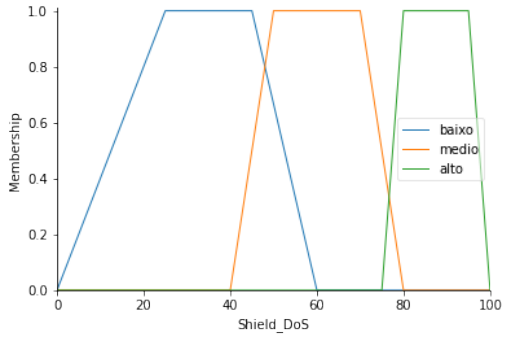
Para exemplificar a implementação foram definidas duas regras. Na **regra1** é definido que se um ataque *DoS* tem pertinência *média* ou *alta*, a VNF de mitigação recomendada seria a *Shield_DoS*. Já na **regra2** é definido que se um ataque *DoS* tem pertinência *média* ou *alta*, a VNF de mitigação recomendada seria a *Firewall*. Com isto vimos que para mitigar um ataque *DoS* é recomendada a utilização de duas VNFs, *Shield_DoS* e *Firewall*. Estas regras são definidas previamente por um operador de rede.

Agora que temos a definição das entradas, saídas e as regras, podemos fornecer um valor de entrada como exemplo para verificar qual será a saída do sistema. Para ilustrar o funcionamento usamos como entrada um ataque *DoS* com pertinência de 0.75. Como retorno para este exemplo, vimos que é 87.50% indicado a utilização das VNFs de mitigação *Shield_DoS* e *Firewall*, conforme a Figura 5.8.

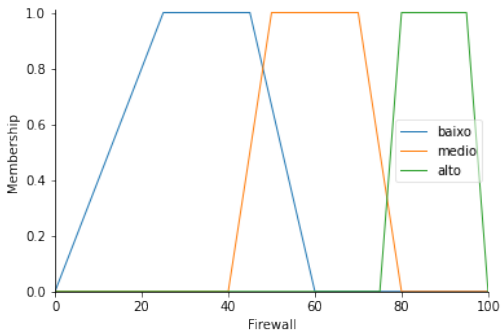
Figura 5.7: Funções de pertinência para as variáveis de ataque DoS, mitigadores DoS Shield e Firewall



(a) Ataque DoS



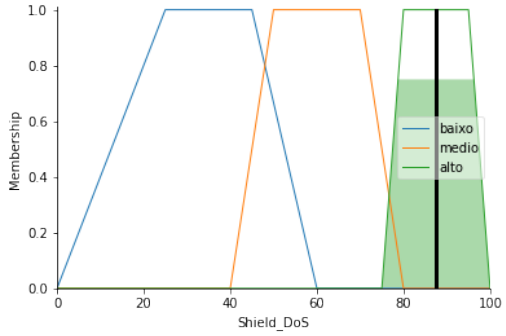
(b) Mitigador Shield DoS



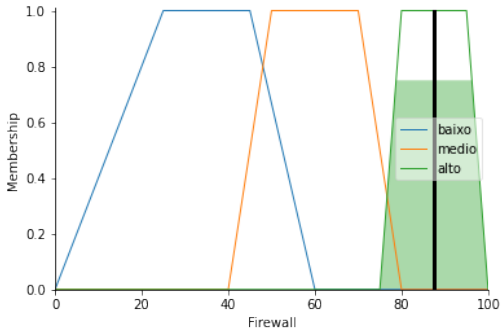
(c) Mitigador Firewall

Fonte: O Autor

Figura 5.8: Saída resultante quando temos como entrada um ataque DoS com a pertinência de 0.75



(a) Mitigador Shield DoS



(b) Mitigador Firewall

Fonte: O Autor

5.2.2 Métricas

Os resultados foram avaliados com base em dois aspectos da implementação, o algoritmo de categorização de tráfego e o desempenho das VNFs em *containers*.

5.2.2.1 Algoritmo de Categorização do Tráfego e Seleção do Mitigador

O algoritmo de categorização do tráfego e seleção do mitigador foi avaliado com base em cinco critérios (*accuracy*, *precision*, *f1-score*, *recall*, *processing time*). Todos esses critérios (excluindo o *processing time*) têm um valor entre 0 e 1. Quando se aproxima de 1, o desempenho aumenta, enquanto que, quando se aproxima de 0, diminui (Bhuyan; Bhattacharyya; Kalita, 2014).

- **Accuracy**: a proporção de dados categorizados com sucesso em relação ao total de dados, conforme Equação 5.1;

$$Accuracy = \frac{\mathrm{TN} + \mathrm{TP}}{\mathrm{FP} + \mathrm{TN} + \mathrm{TP} + \mathrm{FN}} \tag{5.1}$$

- **Precision**: a proporção de dados classificados bem-sucedidos como ataque para todos os dados classificados como ataque, conforme Equação 5.2;

$$Precision = \frac{\mathrm{TP}}{\mathrm{FP} + \mathrm{TP}} \tag{5.2}$$

- **Recall (Sensitivity)**: a proporção de dados classificados como um ataque para todos os dados de ataque, conforme Equação 5.3;

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{5.3}$$

- **F-measure (F-score/F1-score)**: a média harmônica de sensibilidade e precisão. Este conceito é usado para expressar o sucesso geral, conforme Equação 5.4;

$$F\text{-}measure = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}} \tag{5.4}$$

- **Processing Time**: tempo de processamento do algoritmo, em segundos.

Outro método utilizado para avaliar é a comparação entre os algoritmos de aprendizado de máquina utilizados (*Naive Bayes, Random Forest, K Nearest Neighbours e*

QDA). Com isso, objetiva-se observar a eficácia e o desempenho de diferentes algoritmos de aprendizado em diferentes categorias de ataques, para utilizar aquele com melhor resultado obtido após o treinamento.

5.2.2.2 Desempenho de VNFs em Container

Para avaliar o desempenho das VNFs em *containers*, foi utilizada uma VNF do *Snort*[10], um serviço de IDS (Intrusion detection System), em duas plataformas diferentes: *container* e VM. Nas duas plataformas as configurações foram as seguintes: SO Linux Ubuntu 18.04 LTS 64-bit, processador *Intel® Core™ i7-8550U CPU @ 1.80GHz X 8* e *4 GB* de memória. O objetivo principal é entender o impacto das plataformas de virtualização no desempenho das VNFs.

Com o objetivo de estudar o impacto das tecnologias de virtualização e o desempenho real dos serviços, bem como descobrir as limitações das plataformas, foi utilizada a ferramenta de benchmarking **ab**[11] **(Apache HTTP server benchmarking tool)** (BENCHMARK, 2021). A ferramenta *ab* foi instalada no sistema operacional *host* para enviar solicitações aos servidores e coletar os resultados do experimento. A avaliação de desempenho considerou os seguintes critérios:

- **Tamanho da imagem** utilizada nas plataformas, para determinar o armazenamento necessário;
- **Tempo de inicialização**, para determinar o tempo entre solicitação e o deploy;
- **Utilização de CPU**, para determinar a capacidade de escalabilidade do serviço;
- **Utilização de memória**, para determinar a capacidade de escalabilidade do serviço.
- **Delay médio conforme número de requisições**, para determinar a latência com tamanho de pacote fixo para um número crescente de requisições.
- **Delay médio conforme tamanho do pacote**, para determinar a latência com número de requisições fixas em relação ao tamanho crescente de pacote de dados.
- **Tempo por requisição**, para determinar o tempo necessário para processar cada requisição.
- **Taxa de transferência**, para determinar a taxa transferência para um número crescente de requisições.

Nos experimentos o tamanho da imagem e o tempo de inicialização da instância da VNF em execução são comparados. Em seguida, a utilização da CPU e o consumo

[10]https://www.snort.org/
[11]https://httpd.apache.org/docs/2.4/programs/ab.html

de memória são examinadas no estado ocioso e em utilização. Também é avaliada a latência da VNF, avaliando por número de requisições, tamanho de pacote, tempo por requisição e taxa de transferência. Na VM é instalado o *htop*[12] para medir estatísticas sobre o desempenho. As estatísticas do *container* são obtidas através do *docker stats*[13].

5.2.3 Experimentos

Nesta subseção, são apresentados os resultados dos experimentos de acordo com o planejado (Subseção 5.2.2). Iniciando com os resultados obtidos da avaliação do algoritmo de categorização do tráfego e seleção do mitigador e por fim avaliando o desempenho de VNFs em *containers*.

5.2.3.1 Algoritmo de Categorização do Tráfego e Seleção do Mitigador

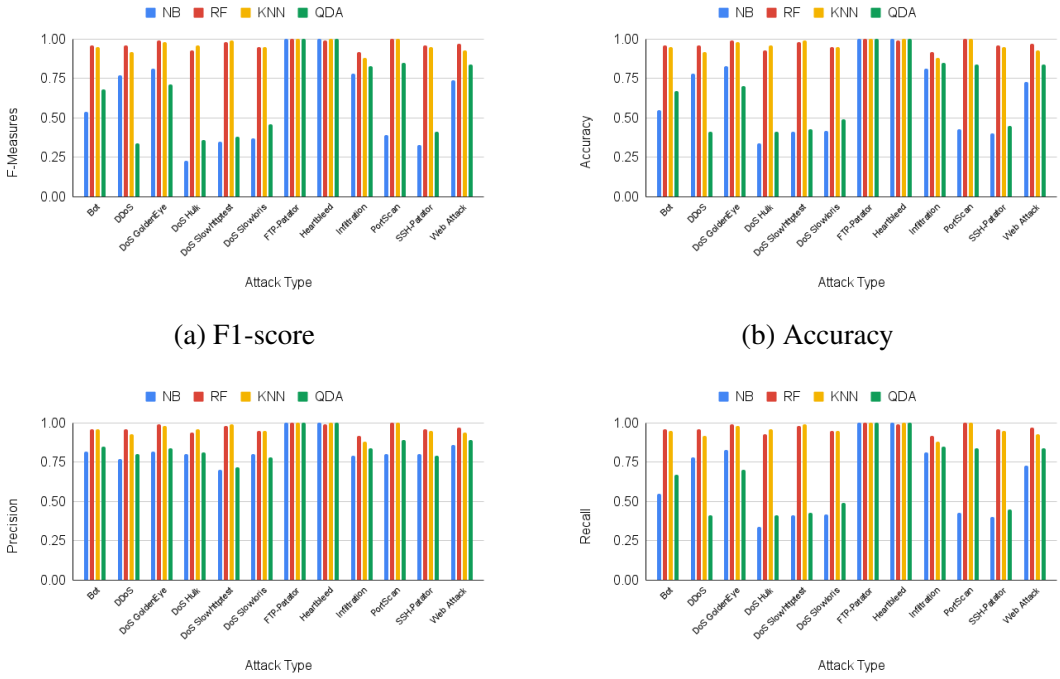
Os quatro algoritmos de aprendizado de máquina foram aplicados as 12 categorias de ataques. Os resultados são apresentados na Figura 5.9.

Ao analisar os resultados, nota-se que os algoritmos Random Forest e KNN só não alcançaram mais de 90% de sucesso na detecção de uma categoria de ataque, levando em consideração as medidas *F1-score*, *accuracy*, *precision* e *recall*. Entre esses 2 algoritmos, o *Random Forest*, que é o mais bem-sucedido, classificou 8 das 12 categorias de ataques com a pontuação mais alta. Um ponto interessante é que *Naive Bayes* se destaca em 5 das categorias de ataques quando se trata de velocidade, como pode ser observado no comparativo da Tabela 5.5, onde também é apresentado o tempo de processamento de cada algoritmo (os piores resultados são apresentados em sublinhado e os melhores em negrito).

Diferente do experimento anterior onde os algoritmos são avaliados para cada categoria de ataque, agora todo o conjunto de dados é usado como um único *dataset*. Todos os ataques contidos neste *dataset* são coletados sob um único nome comum, "attack". Os quatro algoritmos de aprendizado de máquina são aplicados a este conjunto de dados. Neste método, duas abordagens são usadas. Na primeira, as características criadas para cada categoria de ataque, conforme detalhado na abordagem *seleção de características de acordo com os tipos de ataque* (Subseção 5.2.1.2). Na segunda abordagem, são utilizadas as 7 características com peso acima de 0,8% obtidas na *seleção de características entre*

[12]https://htop.dev/
[13]https://docs.docker.com/engine/reference/commandline/stats/

Figura 5.9: Comparativo dos resultados segundo tipos de ataques e algoritmos de aprendizado de máquina



(a) F1-score

(b) Accuracy

(c) Precision

(d) Recall

Fonte: O Autor

fluxo de ataque e benigno (Subseção 5.2.1.1).

A Tabela 5.6 mostra os resultados obtidos usando 18 características extraídas com base na primeira abordagem. Quando os resultados são examinados, verifica-se que o algoritmo de melhor desempenho é o KNN com pontuação de 0,96. Depois vêm *Random Forest* com 0,94. O algoritmo de pontuação mais baixa é o QDA com uma pontuação de 0,30. A pontuação do QDA é cerca de 0,40 pontos menor do que o algoritmo com a pontuação mais próxima (*Naive Bayes*). Do ponto de vista da velocidade, NB e QDA são os algoritmos mais rápidos. Embora KNN tenha a pontuação de desempenho mais alta, é visivelmente mais lento do que outros algoritmos.

Na segunda abordagem os algoritmos são avaliados usando as 7 características selecionadas, conforme apresentado na Tabela 5.7. Os valores que mudam são destacados em *itálico*. Quando a Tabela 5.6 e a Tabela 5.7 são comparadas, não há mudança significativa na avaliação do algoritmo *Random Forest*, com base na *F1-score*. No entanto, *Naive Bayes*, KNN e QDA aumentaram 2, 1 e 11 pontos, respectivamente. Do ponto de vista da velocidade, os tempos de execução de todos os algoritmos são significativamente reduzidos. A razão para essa redução no tempo de execução é a quantidade de características que são usadas para cada abordagem. Essa diminuição na quantidade de

Tabela 5.5: Distribuição dos resultados segundo tipo de ataque, algoritmo de aprendizado de máquina e tempo de execução

Attack Names	Accuracy				F1-Score				Processing Time (s)			
	NB	RF	KNN	QDA	NB	RF	KNN	QDA	NB	RF	KNN	QDA
Bot	0.55	**0.96**	0.95	0.67	0.54	**0.96**	0.95	0.68	**0.0031**	0.0281	0.0176	0.0225
DDoS	0.78	**0.96**	0.92	0.41	0.77	**0.96**	0.92	0.34	0.0491	0.4187	**0.92**	0.0541
DoS GoldenEye	0.83	**0.99**	0.98	0.70	0.81	**0.99**	0.98	0.71	**0.0094**	0.0884	0.1172	0.0141
DoS Hulk	0.34	0.93	**0.96**	0.41	0.23	0.93	**0.96**	0.36	0.3127	3.4215	219.6141	**0.353**
DoS Slowhttptest	0.41	0.98	**0.99**	0.43	0.35	0.98	**0.99**	0.38	0.0073	0.0516	0.0845	**0.062**
DoS Slowloris	0.42	**0.95**	0.95	0.49	0.37	**0.95**	0.95	0.46	**0.0059**	0.0513	0.0419	0.0078
FTP-Patator	**1.0**	1.0	1.0	1.0	**1.0**	1.0	1.0	1.0	**0.0078**	0.0544	0.2547	0.0083
Heartbleed	**1.0**	0.99	1.0	1.0	**1.0**	0.99	1.0	1.0	0.0047	0.0107	**0.0031**	0.0031
Infiltration	0.81	**0.92**	0.88	0.85	0.78	**0.92**	0.88	0.83	0.0031	0.0016	0.0031	**0.0016**
PortScan	0.43	**1.0**	1.0	0.84	0.39	**1.0**	1.0	0.85	**0.2033**	2.2375	49.3462	0.2342
SSH-Patator	0.40	**0.96**	0.95	0.45	0.33	**0.96**	0.95	0.41	0.0078	**0.068**	0.0483	0.0084
Web Attack	0.73	**0.97**	0.93	0.84	0.74	**0.97**	0.93	0.84	0.0044	0.0317	0.0174	**0.0042**

Fonte: O Autor

Tabela 5.6: Aplicação dos algoritmos às características obtidas na abordagem por tipos de ataques.

Machine Learning Algorithms	Evaluation Criteria				
	F1-score	Precision	Recall	Accuracy	Processing Time (s)
Naive Bayes	0.79	0.80	0.78	0.78	**4.5763**
Random Forest	0.94	0.95	0.94	0.94	24.7398
K Nearest Neighbours	**0.96**	0.96	0.97	0.97	1967.0539
QDA	0.30	0.84	0.31	0.31	6.6496

Fonte: O Autor

características reduziu o tempo de execução dos algoritmos de aprendizado de máquina. Na Figura 5.10 o comparativo dos resultados dos algoritmos usando 7 e 18 características podem ser vistos.

Tabela 5.7: Avaliação dos algoritmos conforme características selecionadas em todo o dataset.

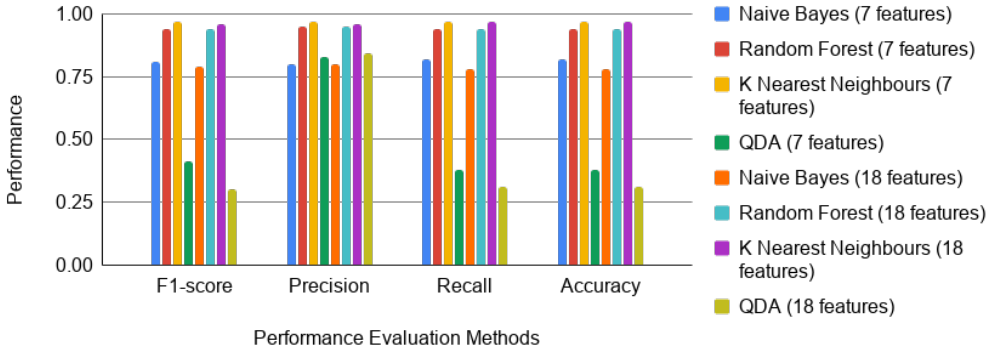
Machine Learning Algorithms	Evaluation Criteria				
	F1-score	Precision	Recall	Accuracy	Processing Time (s)
Naive Bayes	*0.81*	0.8	0.82	0.82	*1.6258*
Random Forest	0.94	0.95	0.94	0.94	*20.512*
K Nearest Neighbours	*0.97*	0.97	0.97	0.97	*1038.2531*
QDA	*0.41*	0.83	0.38	0.38	*1.9025*

Fonte: O Autor

5.2.3.2 Desempenho de VNFs em Container

Analisando os resultados da avaliação de desempenho das VNFs em diferentes plataformas de virtualização, podemos observar que o tamanho da imagem do contêiner (120 MB) é significativamente menor do que na VM. O principal motivo vem do fato de que os contêineres contêm apenas as dependências necessárias para executar o ser-

Figura 5.10: Comparativo dos resultados dos algoritmos com 7 e 18 características.



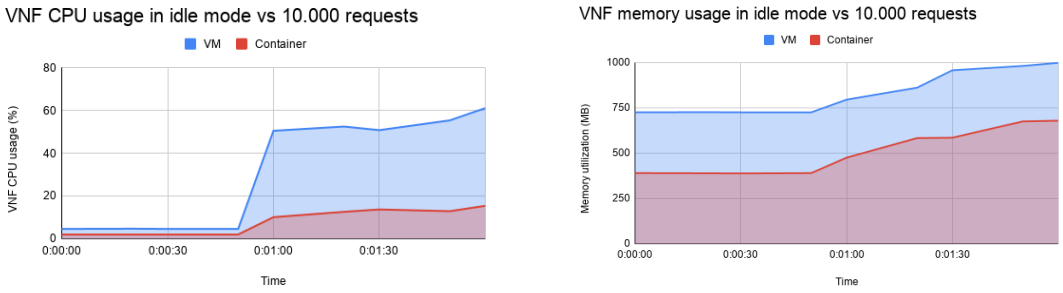
Fonte: O Autor

viço. A VM tem seu tamanho muito maior devido ao sistema operacional que está sendo executado dentro dela, podendo atingir cerca de 14 GB. Quando é analisado o tempo de inicialização da VNF, a VNF em contêiner mostra-se muito mais rápida que a VNF baseada em VM – o contêiner registrou um tempo de inicialização de 2,47 segundos, enquanto a VM registrou um tempo de inicialização de 5,95 minutos. Isso ilustra que a VNF em contêiner apresenta um provisionamento muito mais rápido do que a VNF baseada em VM.

Na Figura 5.11 é comparado o consumo de CPU e de memória, na execução em VM e em contêiner. No primeiro minuto, é mostrada a utilização da CPU e da memória da VNF em modo ocioso. A utilização é comparativamente maior para VM devido ao kernel *guest* e sistema operacional *guest*, respectivamente. Já o contêiner mantém a CPU menos ocupada devido a seu gerenciamento de memória eficiente usando *cgroups*. A partir do primeiro minuto, são exibidas informações após o início da execução do teste de *benchmark*, com 1.000 requisições de 10 clientes em paralelo. Como esperado, o consumo de CPU no contêiner apresenta desempenho entre 70-80% superior que a VM e no consumo de memória entre 40-50% melhor que a VM, devido à VM sofrer penalidade na sobrecarga adicionada pelo *hipervisor*.

Para avaliar a latência média das operações SET e GET na VNF de Snort são utilizadas quatro abordagens. Primeiro, o número de requisições enviadas à VNF é ampliado e o tamanho dos dados é fixado em 32 KBytes. Como pode ser inferido na Figura 5.12a, aumentar o número de requisições não afeta o desempenho da VNF. Na segunda abordagem, o impacto do tamanho dos dados é avaliado gerando 1000 requisições e aumentando o tamanho do pacote das requisições de 16 a 2048 KB. Embora o atraso aumente ligeiramente enquanto aumenta o tamanho dos dados dos pacotes, o desempenho da plataforma

Figura 5.11: Resultado da avaliação do uso de CPU e de memória da VNF Snort em container e VM.

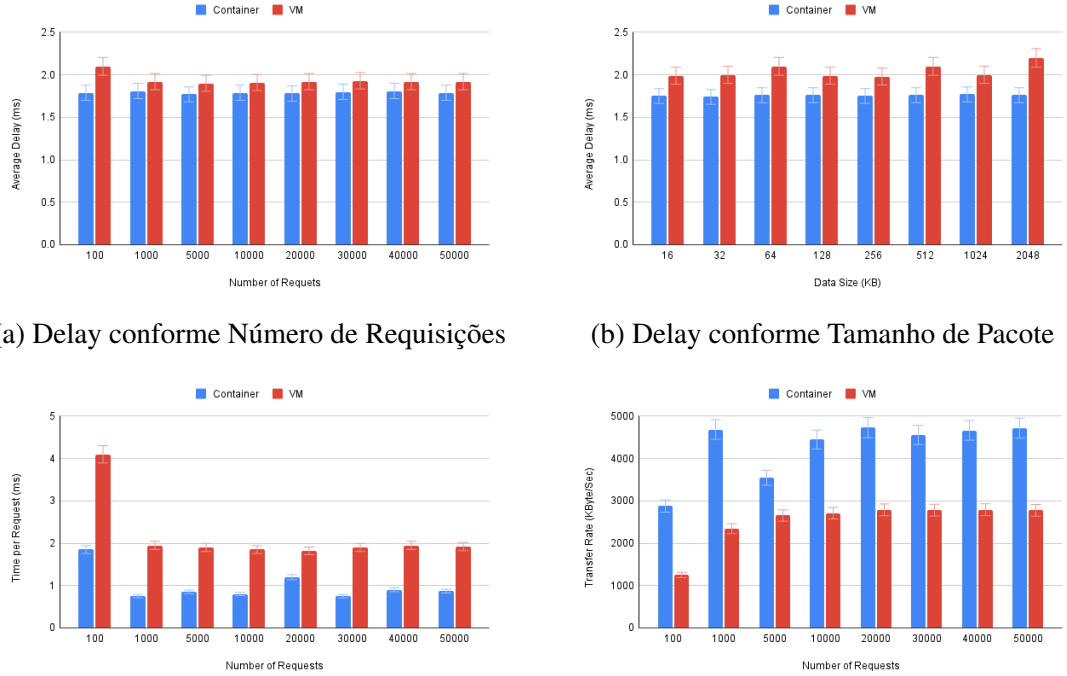


(a) CPU (b) Memória

Fonte: O Autor

de virtualização permanece semelhante ao primeiro caso. Este efeito é mostrado na Figura 5.12b.

Figura 5.12: Resultado da avaliação de latência da VNF Snort por número de requisições, tamanho de pacote, tempo por requisição e taxa de transferência.



(a) Delay conforme Número de Requisições (b) Delay conforme Tamanho de Pacote

(c) Tempo por Requisição (d) Taxa de Transferência

Fonte: O Autor

Na terceira abordagem, foi avaliada a VNF Snort em termos de tempo necessário para atender cada requisição e a taxa de transferência. Conforme representado na Figura 5.12c, o tempo por requisição para VM é mais alto e crescente pelo incremento no número de requisições geradas. Portanto, o atraso para atender requisições é maior em VM, enquanto o *container* se mantêm com um desempenho melhor. Na quarta abordagem, os

mesmos experimentos são realizados para avaliar a taxa de transferência. Como esperado dos resultados anteriores, a VM tem um desempenho pior com uma taxa de transferência mais baixa. Na Figura 5.12d pode-se observar que a taxa de transferência da VM é menor que a do *container*.

6 CONCLUSÃO

Nesse trabalho foi proposta uma possível extensão da arquitetura NFV, para fornecer gerenciamento e orquestração de VNFs para garantir a resiliência da infraestrutura. Um novo componente, o *Intel-OCNF*, foi projetado para integrar a arquitetura NFV, de modo a fornecer orquestração inteligente de funções de rede para mitigação de anomalias da rede. Ao implementar o protótipo desta solução, foi demonstrada a viabilidade da abordagem, implementando técnicas de aprendizado de máquina supervisionado para detecção de anomalias, categorização de ataques e seleção de mitigadores.

6.1 Contribuições

Atualmente, a identificação, a predição, a instanciação e a orquestração de VNFs são problemas que precisam ser resolvidos para prover escolha e orquestração inteligente de VNFs para garantir a operação da rede, de forma correta e eficiente. Os principais desafios são como e com base em quais informações identificar automaticamente qual VNF deve ser instanciada, e posteriormente orquestrar essa VNF.

Neste trabalho foi apresentado um mecanismo, chamado *Intelligent Orchestration of Containerized Network Functions for Anomaly Mitigation (Intel-OCNF)*, que através do uso de aprendizado supervisionado permite identificar quais funções de rede devem ser instanciadas com base em dados de monitoramento. O objetivo principal do Intel-OCNF é definir quais funções devem ser instanciadas de forma a assegurar a resiliência da rede contra tráfego malicioso. Para minimizar o uso de recursos, e agilizar o processo de instanciação e gerenciamento, o Intel-OCNF utiliza virtualização baseada em contêineres. Esse mecanismo tem o objetivo de identificar anomalias no tráfego de rede e implantar VNFs a priori para garantir a qualidade do serviço. O protótipo desenvolvido foi integrado ao orquestrador *Network Functions Virtualization Orchestrator (NFVO)* da plataforma *Open Source MANO (OSM)*, para fornecer orquestração inteligente de funções de rede para mitigação de anomalias. Este trabalho propôs um conjunto de contribuições no contexto de mitigação de ataques e NFV, abordando classificação e mitigação de anomalias no tráfego de rede, uso de virtualização leve e automatização do gerenciamento do ciclo de vida das VNFs. A primeira contribuição propôs o desenvolvimento de um mecanismo baseado em técnicas de *machine learning* para classificar anomalias no tráfego de rede, combinado com a técnica de *fuzzy logic* e com a expertise do operador de rede

para determinar o conjunto ideal de funções virtualizadas para mitigação. A segunda contribuição explorou a viabilidade de uso de contêineres para minimizar o uso de recursos e agilizar o processo de instanciação e gerenciamento. Por fim, a terceira contribuição tratou da investigação e análise das condições de rede que podem ser potencialmente usadas para determinar quais funções manter instanciadas, gerenciando assim de forma automática o ciclo de vida das VNFs.

Os experimentos foram avaliados com base no desempenho do algoritmo de categorização do tráfego e seleção do mitigador e da comparação entre os algoritmos de aprendizado de máquina utilizados (*Naive Bayes, Random Forest, K Nearest Neighbours e QDA*). Também foi avaliado o desempenho das VNFs em contêineres, considerando o tempo de inicialização da VNF, tamanho da imagem, consumo de CPU e memória, latência de processamento das requisições e taxa de transferência. Os resultados obtidos na avaliação dos experimentos demonstraram que com a utilização dos algoritmos selecionados (Naive Bayes, Random Forest, K Nearest Neighbours e QDA) foi possível alcançar mais de 90% de sucesso na detecção das categorias de ataque e com um conjunto de regras aplicadas a *fuzzy logic* determinar a seleção da VNF de mitigação. Além disso, com a utilização de virtualização baseada em contêiner foi possível provisionar uma VNF com *overhead* muito menor que o de uma VNF baseada em VM.

6.2 Trabalhos Futuros

Existem várias possíveis direções de pesquisas futuras. Uma possibilidade seria combinar novas técnicas de identificação de tráfego malicioso na rede com as já propostas neste trabalho, podendo tornar mais eficiente a classificação dos ataques. Outra possibilidade de pesquisa futura seria a utilização de novos algoritmos de *supervised learning*, além dos 4 já utilizados, buscando também melhorar a eficiência na classificação e mitigação dos ataques.

Além disso, apesar dos experimentos demonstrarem bom desempenho por parte dos detectores de anomalias de tráfego de rede, pretende-se também avaliar a eficiência da VNF selecionada e instanciada pelo Intel-OCNF para mitigar a anomalia detectada, considerando o tempo entre um tráfego malicioso ser detectado e uma nova VNF mitigar este ataque e qual o custo computacional para capturar o tráfego e aplicar os algoritmos de aprendizado.

Outra oportunidade de pesquisa é estender a avaliação dos experimentos para

outras formas de virtualização, como, por exemplo, *Unikernels*, *Kernel-based Virtual Machine (KVM)* e *Kata containers*, buscando medir o desempenho de cada plataforma e entender seus benefícios para o contexto de NFV. Como trabalhos futuros, também pretende-se fornecer uma especificação detalhada para integração da solução ao NFVO, possibilitando assim estender a avaliação para outras plataformas NFV-MANO. Além disso, a intenção é ampliar a avaliação da proposta com outros *datasets*, a fim de avaliar a generalidade da solução.

REFERÊNCIAS

Alawe, I. et al. An efficient and lightweight load forecasting for proactive scaling in 5g mobile networks. In: **2018 IEEE Conference on Standards for Communications and Networking (CSCN)**. [S.l.: s.n.], 2018. p. 1–6.

ANDERSON, T. **The theory and practice of online learning**. [S.l.]: Athabasca University Press, 2008.

BENCHMARK, A. **The Apache Software Foundation, ab-Apache HTTP server benchmarking tool**. 2021.

Bhuyan, M. H.; Bhattacharyya, D. K.; Kalita, J. K. Network anomaly detection: Methods, systems and tools. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 303–336, 2014.

BONDAN, L. Nfv environments security through anomaly detection. 2019.

BOUTABA, R. et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. **Journal of Internet Services and Applications**, SpringerOpen, v. 9, n. 1, p. 16, 2018.

BROWNLEE, J. **Practical Machine Learning Problems**. 2018. Available from Internet: <https://machinelearningmastery.com/practical-machine-learning-problems/>.

BROWNLEE, N. **RTFM: New Attributes for Traffic Flow Measurement**. [S.l.], 1999.

BROWNLEE, N.; MILLS, C.; RUTH, G. **Traffic flow measurement: Architecture**. [S.l.], 1999.

Chatras, B. On the standardization of nfv management and orchestration apis. **IEEE Communications Standards Magazine**, v. 2, n. 4, p. 66–71, 2018.

Che, J. et al. A synthetical performance evaluation of openvz, xen and kvm. In: **2010 IEEE Asia-Pacific Services Computing Conference**. [S.l.: s.n.], 2010. p. 587–594.

CHIOSI, M. et al. Network function virtualization: An introduction, benefits, enablers, challenges & call for action. In: **SDN and OpenFlow World Congress**. [S.l.: s.n.], 2012.

CHOI, H.; LEE, H.; KIM, H. Fast detection and visualization of network attacks on parallel coordinates. **computers & security**, Elsevier, v. 28, n. 5, p. 276–288, 2009.

CONTAINERS, L. **Infrastructure for container projects.** 2019. [Online; accessed 11-Agosto-2019]. Available from Internet: <https://linuxcontainers.org/>.

ERSUE, M. Etsi nfv management and orchestration-an overview. In: **Proc. of 88th IETF meeting**. [S.l.: s.n.], 2013.

ETSI. **Network Functions Virtualisation (NFV) Release 2;Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point**. ETSI, 2020. Available from Internet: <https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.07.01_60/gs_nfv-sol005v020701p.pdf>.

ETSI, G. Network functions virtualisation (nfv): Architectural framework. **ETsI Gs NFV**, v. 2, n. 2, p. V1, 2013.

ETSI, G. Network functions virtualisation (nfv); use cases. **V1**, v. 1, p. 2013–10, 2013.

ETSI, I. **Network Functions Virtualization–Introductory White Paper v2. 0**. [S.l.]: Oct, 2013.

ETSI, N. Network functions virtualisation (nfv); management and orchestration. **NFV-MAN**, v. 1, p. v0, 2014.

Fei, X. et al. Adaptive vnf scaling and flow routing with proactive demand prediction. In: **IEEE INFOCOM 2018 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2018. p. 486–494.

GÉRON, A. Hands-on machine learning with scikit-learn and tensorflow: Concepts. **Tools, and Techniques to build intelligent systems**, 2017.

GONZALEZ, A. J. et al. Dependability of the nfv orchestrator: State of the art and research challenges. **IEEE Communications Surveys Tutorials**, v. 20, n. 4, p. 3307–3329, 2018.

Han, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, v. 53, n. 2, p. 90–97, Feb 2015.

HELLEMONS, L. et al. Sshcure: a flow-based ssh intrusion detection system. In: SPRINGER. **IFIP International Conference on Autonomous Infrastructure, Management and Security**. [S.l.], 2012. p. 86–97.

Joy, A. M. Performance comparison between linux containers and virtual machines. In: **2015 International Conference on Advances in Computer Engineering and Applications**. [S.l.: s.n.], 2015. p. 342–346.

KOSTAS, K. Anomaly detection in networks using machine learning. **Research Proposal**, v. 23, 2018.

KRAWCZYK, B. et al. Ensemble learning for data stream analysis: A survey. **Information Fusion**, v. 37, p. 132–156, 2017. ISSN 1566-2535. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1566253516302329>.

LASHKARI., A. H. et al. Characterization of tor traffic using time based features. In: INSTICC. **Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,**. [S.l.]: SciTePress, 2017. p. 253–262. ISBN 978-989-758-209-7.

LI, Y.; LI, T.; LIU, H. Recent advances in feature selection and its applications. **Knowledge and Information Systems**, Springer, v. 53, n. 3, p. 551–577, 2017.

Mijumbi, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, Firstquarter 2016.

MODI, C. et al. A survey of intrusion detection techniques in cloud. **Journal of Network and Computer Applications**, v. 36, n. 1, p. 42 – 57, 2013. ISSN 1084-8045. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1084804512001178>.

Nguyen, T. T. T.; Armitage, G. A survey of techniques for internet traffic classification using machine learning. **IEEE Communications Surveys Tutorials**, v. 10, n. 4, p. 56–76, Fourth 2008.

PALOAUTO-NETWORKS. **What is an Intrusion Prevention System?** 2020. Available from Internet: <https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-prevention-system-ips>.

Pattaranantakul, M. et al. Secmano: Towards network functions virtualization (nfv) based security management and orchestration. In: **2016 IEEE Trustcom/BigDataSE/ISPA**. [S.l.: s.n.], 2016. p. 598–605.

PAUNOVIĆ, M. et al. Two-stage fuzzy logic model for cloud service supplier selection and evaluation. **Mathematical Problems in Engineering**, Hindawi, v. 2018, 2018.

Peuster, M.; Karl, H.; van Rossem, S. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In: **2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2016. p. 148–153.

PUGET, J. F. **Cognitive Computing**. 2016. Available from Internet: <https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning?lang=en>.

Salman, T. et al. Machine learning for anomaly detection and categorization in multi-cloud environments. In: **2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)**. [S.l.: s.n.], 2017. p. 97–103.

Saravanan, R.; Sujatha, P. A state of art techniques on machine learning algorithms: A perspective of supervised learning approaches in data classification. In: **2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)**. [S.l.: s.n.], 2018. p. 945–949.

Schardong, F.; Nunes, I.; Schaeffer-Filho, A. Providing cognitive components with a bidding heuristic for emergent nfv orchestration. In: **NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2018. p. 1–9. ISSN 2374-9709.

Schardong, F.; Nunes, I.; Schaeffer-Filho, A. NFV resource allocation: a systematic review and taxonomy of VNF forwarding graph embedding. **Computer Networks**, v. 185, p. 107726, 2021. ISSN 1389-1286.

SEGAL, M. R. Machine learning benchmarks and random forest regression. 2004.

SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: **ICISSP**. [S.l.: s.n.], 2018. p. 108–116.

Silva, A. S. d. et al. Identification and selection of flow features for accurate traffic classification in sdn. In: **2015 IEEE 14th International Symposium on Network Computing and Applications**. [S.l.: s.n.], 2015. p. 134–141.

TREND-MICRO. **Enterprise Intrusion Prevention (IPS) Software and Solutions**. 2020. Available from Internet: <https://www.trendmicro.com/en_us/business/products/network/intrusion-prevention.html>.

Zhang, X. et al. Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization. In: **IEEE INFOCOM 2017 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2017. p. 1–9.

Zhou, L.; Guo, H. Applying nfv/sdn in mitigating ddos attacks. In: **TENCON 2017 - 2017 IEEE Region 10 Conference**. [S.l.: s.n.], 2017. p. 2061–2066.