UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE ARRUDA FAUSTINI

Emprego de NFV e aprendizagem de máquina para detectar e mitigar anomalias em redes definidas por software

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho

Porto Alegre
2016

“If debugging is the process of removing software bugs,
then programming must be the process of putting them in.”
— EDSGER DIJKSTRA

AGRADECIMENTOS

Sobretudo agradeço à minha família, em especial aos meus pais Márcia e Erico, que sempre me deram toda força e apoio ao longo desta jornada.

Também agradeço ao meu orientador Alberto por toda a paciência e ajuda ao longo do trabalho.

Aos meus amigos de fora da UFRGS, e também às amizades que fiz aqui dentro, fica aqui o meu muito obrigado.

RESUMO

Uma rede de computadores é dita resiliente quando consegue manter níveis adequados de operação mesmo frente a anomalias, minimizando prejuízos aos usuários. Este trabalho propõe uma coordenação harmônica de diferentes técnicas a fim de promover resiliência para diferentes tipos de anomalias em redes definidas por software (SDN). Em especial, propõe-se que métricas de rede sejam coletadas e agrupadas em perfis, e cada perfil tenha um conjunto de ações que trate os problemas encontrados usando aprendizagem de máquina, virtualização de funções de rede (NFV) e controlador SDN. São abordadas anomalias tipicamente maliciosas, como ataques de negação de serviço, mas também benignas, como balanceamento de tráfego legítimo.

Palavras-chave: SDN. NFV. reinforcement learning. redes de computadores.

Adoption of NFV and machine learning to detect and mitigate anomalies in software-defined networks

ABSTRACT

A computer network is said to be resilient when it is able to keep appropriate levels of operation even against anomalies, minimising damage to users. This work proposes a harmonic coordination of different techniques in order to promote resilience for different kinds of anomalies in software-defined networks (SDN). In particular, it is proposed the gathering of network metrics and their grouping into profiles, each one having a set of actions to handle the encountered problems using machine learning, network functions virtualisation (NFV) and the SDN controller. Typically malicious anomalies are addressed, like denial of service attacks, as well as benign anomalies, such as legit traffic load balancing.

Keywords: SDN, NFV, reinforcement learning, computer networks.

LISTA DE FIGURAS

Figura 2.1 Arquitetura de SDN. 14
Figura 2.2 Arquitetura de NFV. 16
Figura 2.3 Esquema geral de aprendizagem por reforço. 18

Figura 3.1 Arquitetura do modelo 21
Figura 3.2 Exemplo de topologia 24
Figura 3.3 Exemplo de topologia 32
Figura 3.4 Resultado de réplica de servidor 35
Figura 3.5 Situação da rede no momento da requisição de Hc_5 36
Figura 3.6 Situação da rede acomodando os clientes 38

Figura 4.1 Integração Docker + Mininet 41
Figura 4.2 Diagrama de classes. 42
Figura 4.3 Topologia 44
Figura 4.4 Recompensas imediatas obtidas. 45
Figura 4.5 Requisições enviadas e recebidas. 45
Figura 4.6 Topologia de teste. 46
Figura 4.7 Recompensas imediatas obtidas ao longo do tempo 46
Figura 4.8 Recompensas imediatas obtidas ao longo do tempo 47
Figura 4.9 Crescimento da tabela estado-ação e estados visitados 48
Figura 4.10 Crescimento da tabela estado-ação e estados visitados 48

LISTA DE TABELAS

Tabela 3.1 Caminhos possíveis 25
Tabela 3.2 Instância de tabela de estados 25
Tabela 3.3 Instância de tabela de estados 26
Tabela 3.4 Exemplo de tabela de estados 28
Tabela 3.5 Caminhos possíveis 34
Tabela 3.6 Instância de tabela de estados 35
Tabela 3.7 Instância de tabela estado-ação 36
Tabela 3.8 Instância de tabela estado-ação 37
Tabela 3.9 Instância de tabela de estados 37
Tabela 3.10 Instância de tabela de estados 39
Tabela 3.11 Instância de tabela estado-ação 39

LISTA DE ABREVIATURAS E SIGLAS

DoS	Denial of Service
DDoS	Distributed Denial of Service
MDP	Markov Decision Process
NFV	Network Functions Virtualisation
SDN	Software Defined Networks
VNF	Virtual Network Function

SUMÁRIO

1 INTRODUÇÃO **11**
1.1 Objetivos **11**
1.2 Contribuições **12**
1.3 Estrutura **12**
2 FUNDAMENTAÇÃO TEÓRICA **13**
2.1 SDN **13**
2.1.1 OpenFlow 14
2.2 NFV **15**
2.2.1 Ameaças 17
2.3 Aprendizagem de máquina **18**
2.3.1 Aprendizagem por reforço 18
2.3.1.1 Processo de decisão de Markov 19
2.3.1.2 Função valor 20
3 MODELO DE APRENDIZAGEM PARA RESILIÊNCIA EM SDN **21**
3.1 Definições e requisitos **22**
3.2 Modelagem da aprendizagem de máquina **22**
3.2.1 Balanceamento de carga 23
3.2.1.1 Métricas necessárias 23
3.2.1.2 Conjunto de Ações 25
3.2.1.3 Recompensa 26
3.2.2 Carga sobre servidores 27
3.2.2.1 Construção dos estados 27
3.2.2.2 Conjunto de ações 28
3.2.2.3 Recompensa 28
3.2.3 Detecção e mitigação 29
3.2.4 Atualização da tabela estado-ação 30
3.3 Estudo de caso **32**
3.3.1 Passo a passo dos ciclos de aprendizagem 33
4 IMPLEMENTAÇÃO E ANÁLISE DE RESULTADOS **40**
4.1 Implementação do protótipo **40**
4.1.1 Integração do ambiente 40
4.1.2 Protótipo 41
4.2 Experimentos e análise de resultados **43**
4.2.1 Descrição do ambiente e experimentos 43
4.2.1.1 Mitigação e balanceamento 43
4.2.1.2 Balanceamento em topologia maior 46
5 TRABALHOS RELACIONADOS **49**
5.1 Anomalias em SDN **49**
5.2 Aprendizagem por reforço em redes **50**
5.3 Considerações **51**
6 CONCLUSÃO E TRABALHOS FUTUROS **52**
REFERÊNCIAS **54**

1 INTRODUÇÃO

O conceito de redes definidas por software (SDN, ou *software defined networking*, no original em inglês) fornece um novo paradigma para o gerenciamento de recursos de rede. Uma das suas mais destacadas características é a figura de um software controlador que centraliza a lógica de funcionamento da rede. Contudo, um problema em atribuir tanta responsabilidade em um só personagem é o risco de ponto único de falha, o que pode comprometer a resiliência de toda a rede (FONSECA et al., 2012).

Por outro lado, SDN oferece um ambiente propício para a implantação de virtualização de funções de rede (NFV, ou *network functions virtualisation*). (ETSI, 2012). Esta tecnologia permite o desenvolvimento de funções de rede, outrora implementadas em hardware específico, via software executado em hardware de propósito geral. Dessa forma, ações para manter a resiliência podem ser distribuídas entre controlador e funções de rede.

A proposta deste trabalho surge em um momento em que busca-se compreender a cooperação possível entre os universos de SDN e NFV. Visando à resiliência da rede, isto é, mantê-la operante mesmo sob efeito de ameaças, é preciso, contudo, coordenar as responsabilidades de cada personagem, definindo quais tarefas serão incumbidas a controlador e funções de rede e como elas devem ser executadas. Propõe-se a adoção de aprendizagem por reforço (*reinforcement learning*, ou RL) (MITCHELL, 1997; SUTTON; BARTO, 2012; SZEPESVáRI, 2013) para guiar o sistema a aprender a como reagir frente a diferentes tipos de anomalias, tanto via instanciação de diferentes funções de rede ou por ações a partir do controlador.

1.1 Objetivos

O presente trabalho tem como objetivo geral investigar como redes definidas por software podem se beneficiar de técnicas baseadas em aprendizagem de máquina e NFV de modo a selecionar estratégias de mitigação para diferentes tipos de anomalias. Entende-se por anomalias não somente a presença de fluxos maliciosos, como no caso de ataques de negação de serviço, como também fluxos benignos mas que combinados podem deteriorar a experiência de usuários (STERBENZ et al., 2010). Por exemplo, a sobrecarga de um servidor devido ao alto número, ainda que legítimo, de requisições, ou o gargalo em um link que transmite muitos pacotes de diferentes hosts. Todos esses casos são considerados anomalias, e portanto devem ser tratados.

Para se atingir o objetivo geral, em específico busca-se a *(i)* investigar o estado da arte

e realizar uma revisão da literatura sobre o uso de aprendizagem por reforço em redes, *(ii)* modelar um agente que adota técnicas de aprendizagem por reforço e aprende a como lidar com diferentes tipos de anomalias, *(iii)* não ficar restrito à maleabilidade que SDN confere aos programadores e administradores de rede, mas adotar também a tecnologia de NFV para auxiliar a detecção e mitigação de ameaças, e *(iv)* avaliar a solução proposta por meio da implementação de um protótipo.

1.2 Contribuições

Espera-se sobretudo que este trabalho contribua para promover a resiliência em redes definidas por software. Em especial, avaliar o emprego de aprendizagem de máquina como técnica principal na tomada de ações a partir da coleta de evidências de anomalias. Não obstante, também é esperado contribuir para o trabalho conjunto das tecnologias SDN e NFV que, apesar de independentes, podem ser complementares. Por fim, não se restringir a uma solução teórica, mas também prover resultados de experimentos práticos, ainda que em ambientes simulados.

1.3 Estrutura

O restante deste trabalho segue a seguinte estrutura: no Capítulo 2 são apresentados os conceitos relativos a redes definidas por software, virtualização de funções de rede e aprendizagem de máquina na próxima sessão. Na sequência, no Capítulo 3 é proposto o modelo de trabalho conjunto de NFV e aprendizagem de máquina para promover a resiliência de uma rede definida por software. Depois, no Capítulo 4 são apresentados o protótipo implementado e os resultados. O Capítulo 5 contém os trabalhos relacionados e o documento é encerrado com as conclusões e prospecções de trabalhos futuros no Capítulo 6.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os principais conceitos que alicerçam este trabalho. A Seção 2.1 aborda SDN e o protocolo OpenFlow. Em seguida, a Seção 2.2 trata sobre a tecnologia NFV. Por fim, Seção 2.3 versa a respeito de aprendizagem de máquina, em especial aprendizagem por reforço.

2.1 SDN

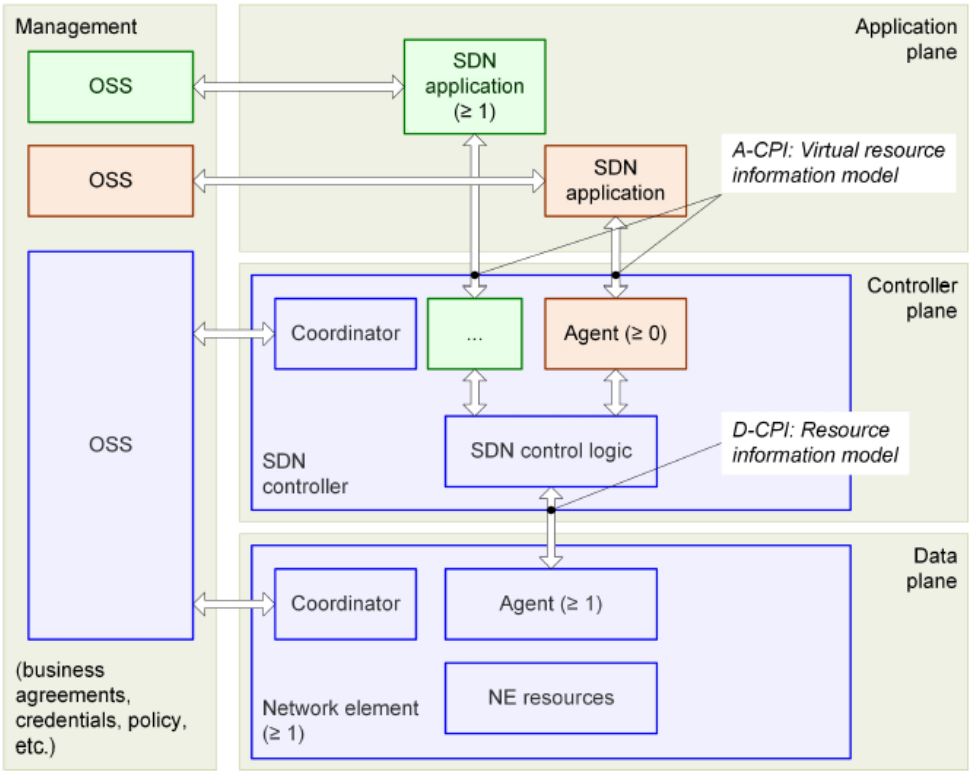
O conceito de SDN surgiu da necessidade de alterar o funcionamento das redes conforme elas adquirem novas demandas. Tradicionalmente, switches são o cérebro da rede e responsáveis tanto pela lógica como pelo ato de encaminhar pacotes. Essa forma, além de descentralizada, une plano de dados e controle, deixando a rede pouco flexível para experimentos e mudanças definitivas. A única maneira de alterar políticas de tráfego é reconfigurando cada dispositivo (SEZER et al., 2013), ou adquirindo novos.

SDN, por outro lado, desacopla o plano de controle do plano de dados. O último é mantido em hardware, mas o primeiro é transferido para um software, portanto, programável e logicamente centralizado (ONF, 2012). A figura do controlador possui visão global da rede e assim é capaz de otimizar o gerenciamento de fluxo com flexibilidade e escalabilidade. Um exemplo é a possibilidade de se alterar dinamicamente estratégias de roteamento.

Sezer et al. (2013) definem o *modus operandi* de SDN da seguinte maneira: quando o pacote de um dado fluxo chega em um switch pelo plano de dados, e não houver regras na tabela de fluxo que indiquem para o switch o que fazer, o pacote é encaminhado ao controlador. Este, por sua vez, além de decidir a ação a ser tomada, instala uma nova regra na tabela de fluxos do switch. Por exemplo, pode determinar que pacotes provenientes de um determinado endereço IP sejam descartados. Ou então, que sejam encaminhados para uma das portas do switch.

Conforme ilustra a Figura 2.1, redes definidas por software são baseadas em três camadas (JMAL; FOURATI, 2014): infraestrutura, controle e aplicação. Infraestrutura, ou camada de dados, corresponde aos dispositivos físicos que compõem a rede, como os switches. A camada de controle é onde reside o software controlador e ela se comunica com as outras camadas através de interfaces. O controlador também possui uma visão global da rede. A camada de aplicação é onde as aplicações que se valem da SDN são executadas, como *firewalls*, balanceadores de fluxo, detectores de intrusos, entre outros.

Figura 2.1: Arquitetura de SDN.



Fonte: (ONF, 2014).

2.1.1 OpenFlow

SDN é um conceito de rede e o protocolo OpenFlow é o padrão *de facto* para comunicação entre plano de controle e plano de dados (NUNES et al., 2014), ainda que existam outros, como ForCES e I2RS (MATIAS et al., 2015). Seu desenvolvimento começou em 2007 e foi padronizado pela Open Networking Foundation. Atualmente, diversas empresas colaboram com o projeto, como Cisco, IBM, HP e outras (JMAL; FOURATI, 2014). O protocolo serve como um canal entre as camadas de infraestrutura e controle. Através de uma interface é possível adicionar ou deletar entradas nas tabelas de fluxos dos switches. Além disso, outra motivação é facilitar inovação ao permitir, por exemplo, a pesquisa de novos protocolos de roteamento, separando o tráfego entre fluxos que utilizam SDN e ordinário (MCKEOWN et al., 2008).

Para uma rede OpenFlow funcionar, é necessário que os switches suportem o protocolo. Tais switches podem ser de dois tipos: dedicados ou habilitados. Um switch é dedicado quando suporta três requisitos: primeiro, os fluxos são encaminhados com base nas tabelas de fluxos. Segundo, os switches também podem enviar pacotes ao controlador por meio de um canal seguro. Isto é geralmente feito com o primeiro pacote de um fluxo, e o controlador decide uma ação que será aplicada a ele e aos demais pacotes do fluxo, que dessa forma não precisarão

atingir o controlador. Terceiro, pacotes podem ser descartados. Switches habilitados são aqueles que, além de suportarem essas três características descritas, também permitem que pacotes percorram a rede seguindo as regras de roteamento tradicionais (MCKEOWN et al., 2008).

No contexto do protocolo OpenFlow, cada entrada em uma tabela de fluxo de um switch possui três partes (MCKEOWN et al., 2008): (i) uma máscara de match definida em termos dos cabeçalhos dos pacotes, (ii) a ação a ser tomada para aquele fluxo e (iii) estatísticas, tais como contagem de bytes e de pacotes, ou a última vez que aquela regra foi acionada para algum pacote. O programador pode se valer dessas estatísticas para, por exemplo, expirar uma regra inativa por algum tempo determinado. Além disso, não é obrigatório usar todas as informações do cabeçalho.

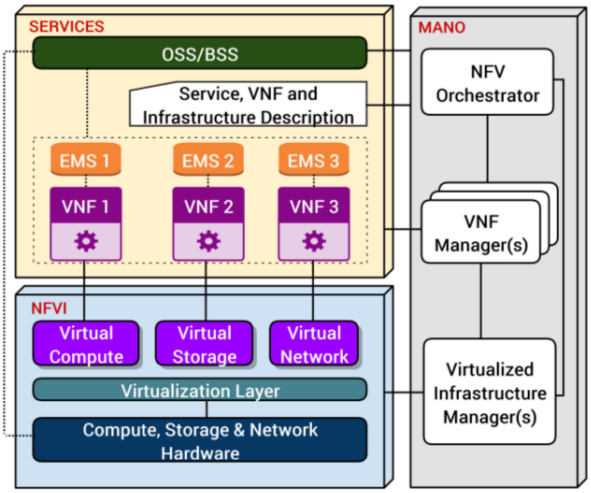
Inicialmente, o protocolo OpenFlow considerava a presença de um único controlador para gerenciar a lógica da rede. Entretanto, arquiteturas mais recentes têm suportado que múltiplos controladores cooperem entre si e estejam distribuídos pela rede para fins de escalabilidade (AHMAD et al., 2015).

2.2 NFV

Operadoras costumam possuir muitas **funções de rede** espalhadas por sua infraestrutura. Entre exemplos de funções de rede pode-se destacar *firewalls*, sistema de detecção de intrusos, monitores, honeypots e outros. O problema é que tais equipamentos, geralmente construídos em hardware dedicado, rapidamente atingem o fim da vida útil. Além disso, mesmo enquanto apresentam desempenho satisfatório, podem facilmente ser ultrapassados por outras tecnologias mais recentes (HERRERA; BOTERO, 2016). Outro problema é a falta de flexibilidade no gerenciamento desses dispositivos. A ideia de virtualização de funções de rede (*Network Functions Virtualisation*, ou NFV) busca portar tais funções para hardware de propósito geral, executado em ambientes virtualizados, como máquinas virtuais. Entre os benefícios estão redução de custos, escalabilidade e flexibilidade para inovação. Contudo, a troca de hardware dedicado para máquinas de propósito geral traz desafios. Entre os principais estão a performance em ambientes virtualizados, resiliência frente a falhas tanto de hardware como de software, entre outros (ETSI, 2012).

A arquitetura NFV é compreendida em três componentes principais, conforme exibe a Figura 2.2. São eles: infraestrutura, serviço e orquestração & gerenciamento (HERRERA; BOTERO, 2016).

Figura 2.2: Arquitetura de NFV.



Fonte: (HERRERA; BOTERO, 2016).

- **Infraestrutura**: conhecida como NFVI (*NFV Infraestructure*) abrange os recursos de hardware e software. Nesta parte está também o ambiente de virtualização, como hipervisor (e.g. Virtual Box) ou contêiner (e.g. Docker).
- **Serviço**: As funções de rede virtualizadas são conhecidas como VNFs (*Virtualised Network Function*). Elas são executadas em ambientes virtualizados, em vez de hardware dedicado, (*middleboxes*)(KING; FARREL; GEORGALAS, 2015). A parte de serviço é um conjunto de VNFs.
- **Orquestração e gerenciamento**: conhecida como NFV-Mano (*management and orchestration*), esta parte foca em tarefas mais específicas da virtualização. Por exemplo, se entre as VNFs estão *firewall* e DPI, NFV-Mano informará onde elas estarão distribuídas pela rede. As VNFs serão também controladas pelo NFV-Mano (por exemplo, determinar, após o resultado da inspeção do DPI sobre um pacote, que o *firewall* o descarte ou o deixe prosseguir).

Apesar dos conceitos de NFV e SDN serem independentes, as tecnologias podem ser complementares. Por exemplo, NFV pode prover a infraestrutura na qual SDN é executada (ETSI, 2012). Além disso, o próprio controlador SDN pode ser uma função virtual de rede, e é possível que, no futuro, ambas tecnologias se tornem menos distinguíveis até se unirem como um único paradigma em redes baseadas em software (ETSI, 2015).

2.2.1 Ameaças

Segurança é um tópico inerente à área de redes, e no caso de SDN e NFV não é diferente. Diversas ameaças, sejam elas já tradicionais da área de redes, ou novas que exploram características específicas dessas tecnologias têm sido estudadas por pesquisadores (AHMAD et al., 2015; MOUSAVI; ST-HILAIRE, 2015; JONES; SIELKEN, 2000). Em especial, abordar-se-á ataques de negação de serviços (DoS) e ataques distribuídos de negação de serviço (DDoS) no contexto de SDN e NFV.

Em ataques DoS, o atacante continuamente envia pacotes com cabeçalhos aleatórios em um estado não responsivo. Os pacotes forçam o controlador a criar novas entradas nas tabela de fluxo, e a grande quantidade de tráfego leva à perda da comunicação entre switch e controlador (FONSECA et al., 2013).

Também é possível que um ataque de DoS foque não no controlador, mas em switches, no plano de dados. Suas tabelas de fluxo têm espaço limitado e, portanto, podem ficar cheias. Uma vez que isso aconteça, e uma instrução para se adicionar uma regra chegue ao switch, uma mensagem de erro é enviada ao controlador e o switch não consegue encaminhar pacotes daquele fluxo (KANDOI; ANTIKAINEN, 2015).

Ataques de negação de serviço distribuídos (DDoS), assim como no caso de ataques DoS, acontecem quando uma quantidade excessiva de pacotes, muitas vezes forjados, são enviados a uma máquina a fim de sobrecarregá-la. A diferença é que não há uma máquina de origem, e sim várias. Os efeitos são análogos aos descritos acima em uma SDN: os switches tendem a enviar uma grande quantidade de pacotes ao controlador e pacotes legítimos acabam não conseguindo chegar ao seu destino porque o controlador fica ocupado (ASHRAF; LATIF, 2014).

VNFs podem auxiliar a rede a manter resiliência. Funções tais como sistemas de detecção de intrusos (IDS) podem identificar hosts mal-intencionados, bem como analisadores de tráfego podem distinguir fluxos maliciosos daqueles legítimos. Entre as ações de mitigação, VNFs podem servir como *firewalls* e barrar pacotes (MACHADO; GRANVILLE; SCHAEFFER-FILHO, 2016), ou ainda balanceadores de tráfego, replicando servidores, por exemplo, para liberar o estresse sobre uma vítima.

2.3 Aprendizagem de máquina

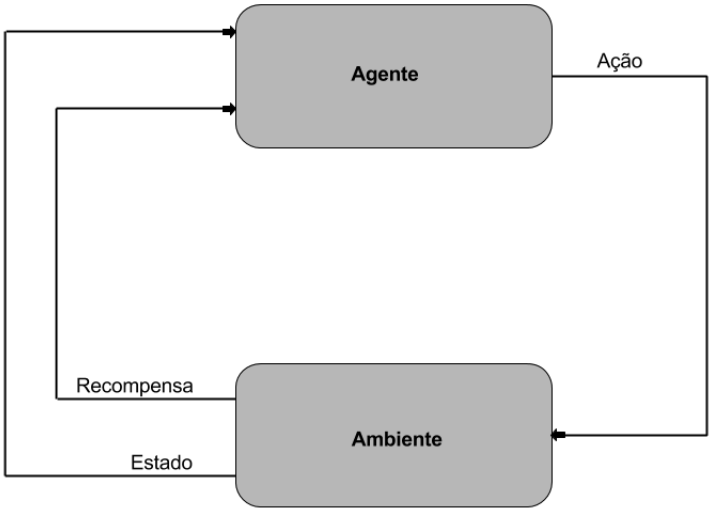
Aprendizagem de máquina é uma subárea da inteligência artificial. É dito que um programa aprende a partir de uma experiência **E**, com respeito a um conjunto de tarefas **T** e medido pela performance **P**, se sua performance **P** nas tarefas em **T**, melhora conforme a experiência **E** (MITCHELL, 1997).

A aprendizagem de máquina, por sua vez, possui três ramificações: aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem por reforço (RUSSELL; NORVIG, 2003). Este trabalho focará na última delas.

2.3.1 Aprendizagem por reforço

Aprendizagem por reforço envolve um agente que analisa um ambiente e interage com ele realizando ações. A partir de uma ação tomada no instante *t*, no próximo momento *t+1* ele muda de estado e recebe uma recompensa. A recompensa expressa aquilo que se quer atingir, não como atingir. Por exemplo, em um jogo de xadrez, a recompensa positiva seria ganhar a partida, e a negativa seria perdê-la. O ato de derrubar peças adversárias não devem ser recompensado, senão o agente poderia maximizar a recompensa derrubando várias peças ao custo de perder o jogo (SUTTON; BARTO, 2012).

Figura 2.3: Esquema geral de aprendizagem por reforço



Fonte: O autor.

A cada instante tempo o agente atualiza o mapeamento de estado para probabilidade de se selecionar tal ação. Este mapeamento é chamado de política, denotada por π, sendo a política ótima π_*. A chance da ação a ser tomada no estado s é expressa por $\pi_t(a|s)$ (SUTTON;

BARTO, 2012).

2.3.1.1 Processo de decisão de Markov

Isto posto, uma maneira de **definir** os problemas de aprendizagem por reforço é através dos processos de decisão de Markov (*Markov decision process*, ou MDP), e entre as técnicas utilizadas para **resolvê-los** estão programação dinâmica e algoritmos de aprendizagem de diferença temporal (*temporal difference learning*) (SZEPESVáRI, 2013). Geralmente, programação dinâmica é usada quando as dinâmicas do ambiente são conhecidas. Senão, ou quando tal abordagem se torna inviável devido ao custo de espaço frequentemente envolvido com programação dinâmica em grandes escalas, costuma-se adotar a segunda abordagem (MALIALIS, 2014).

MDP fornece um *framework* matemático muito usado na modelagem das dinâmicas de um ambiente sob diferentes ações. Existem diversas definições formais de MDP na literatura (SZEPESVáRI, 2013; RUSSELL; NORVIG, 2003; MALIALIS, 2014; SUTTON; BARTO, 2012). Este trabalho usará a definição apresentada por Malialis (2014), que engloba os principais itens de aprendizagem de máquina (estados, ações e as dinâmicas do ambiente): um MDP é definido por uma 4-upla $\langle S, A, R, P \rangle$ em que:

- **S**: conjunto finito de estados.
- **A**: conjunto finito de ações.
- **R**: função de recompensa: retorna a recompensa esperada no estado s_{t+1}, quando a ação a é executada no estado s.
- **P**: função de probabilidade de transição: retorna a probabilidade de se atingir o estado s_{t+1} quando a ação a é executada no estado s. As funções de recompensa e probabilidade compõem a dinâmica do ambiente.

Idealmente os estados devem apresentar a propriedade de Markov, isto é, devem reter toda a informação relevante, o que pode incluir informações de estados passados. Por exemplo, em um jogo de xadrez a configuração de todas as peças no tabuleiro contém toda a informação relevante. Muita informação do que aconteceu ao decorrer da partida foi perdida, mas o essencial para um jogador decidir o que fazer está presente (SUTTON; BARTO, 2012). Isso permite prever o próximo estado e a recompensa a partir do estado atual e da ação (MALIALIS, 2014).

2.3.1.2 Função valor

Os problemas de aprendizagem por reforço costumam envolver a estimação de funções valor. Elas procuram mostrar o quão bom é um agente estar em um determinado estado, ou realizar uma certa ação em um estado. São destacados dois tipos de funções (SUTTON; BARTO, 2012):

- **Função valor-estado**: $V^{\pi}(s)$ denota o valor de um estado s sob uma política π. É definida pelo retorno esperado ao iniciar pelo estado s e aplicar a política π. Formalmente:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}|s_t = s\right\} \tag{2.1}$$

  onde R é a recompensa, γ é o fator de desconto, s_t é o estado no *time step t*, k corresponde às iterações com o ambiente e E_{π} é a recompensa esperada caso o agente siga a política π.

- **Função ação-valor**: $Q(s, a)$ é definida como o retorno esperado ao executar uma ação a a partir um estado s e então seguir uma política π.

$$Q^{\pi}(s, a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}|s_t = s, a_t = a\right\} \tag{2.2}$$

  onde α é a taxa de aprendizagem e E_{π} é a recompensa esperada caso o agente inicie pelo estado s, tome a ação a e então siga a política π. Os demais símbolos mantêm o significado da equação anterior.

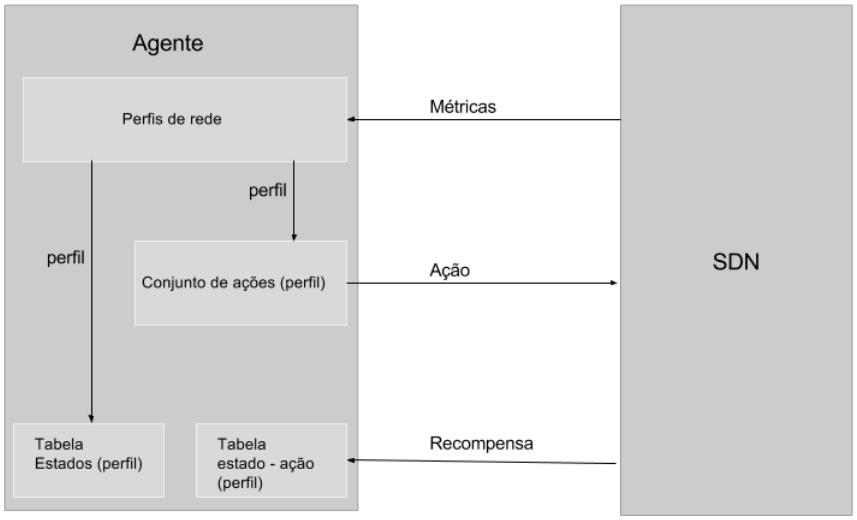
Ambas podem ser estimadas por experiência. Por exemplo, ao manter um registro do retorno obtido para cada estado, a média encontrada tende a convergir para o valor do estado $V^{\pi}(s)$. Se o mesmo for feito, mas além do estado o agente guardar qual ação tomou, a média encontrada tende a convergir para o valor $Q^{\pi}(s, a)$ (SUTTON; BARTO, 2012).

Conforme já foi frisado, o objetivo do agente é maximizar a recompensa. Entretanto, se a interação com com o ambiente nunca cessar, a maximização pode tender ao infinito, mas ela deve sempre ser um número. Surge então a necessidade de se aplicar um fator de desconto, chamado de γ. O desconto determina o quanto futuras recompensas influenciarão na maximização. Por exemplo, se γ for 0, o agente só levará em consideração recompensas imediatas. Se for maior que zero, mas menor que 1, haverá uma convergência para um número, e a cada passo *k* a recompensa valerá γ^{k-1} vezes menos (SUTTON; BARTO, 2012). As funções valor sob a política ótima π_* são denotadas por $V^{*}(s)$ e $Q^{*}(s, a)$.

3 MODELO DE APRENDIZAGEM PARA RESILIÊNCIA EM SDN

Este trabalho propõe a adoção de técnicas de aprendizagem por reforço em conjunto com infraestruturas baseadas em SDN/NFV para tornar redes definidas por software mais resilientes frente a anomalias. O modelo de aprendizagem por reforço é integrado à arquitetura de NFV por meio de um agente de RL que reside no orquestrador. Este agente recebe métricas de rede e, a partir delas, constrói perfis de rede. Cada perfil possui um conjunto de ações associado, bem como uma tabela de estados e uma tabela estado-ação. A cada ciclo de aprendizagem, o agente seleciona um perfil, executa uma ação disponível e recebe uma recompensa para atualizar a tabela estado-ação daquele perfil, conforme exibe a Figura 3.1.

Figura 3.1: Arquitetura do modelo



Fonte: O autor.

Ao escolher uma ação para um perfil, o agente automaticamente realiza uma ação vazia para todos os demais perfis. Isso acontece para evitar que, ao executar duas ou mais ações em um mesmo ciclo, uma delas afete os efeitos da outra. A cada perfil pode ser conferida uma prioridade, de forma que se uma anomalia for detectada em dois ou mais perfis, o agente dará prioridade a um deles ao escolher uma ação para mitigá-la.

Dois perfis podem precisar ser mesclados para o sistema funcionar harmonicamente. Por exemplo, um determinado perfil de rede pode ter, em seu conjunto de ações, a busca por caminhos alternativos entre dois hosts caso a rota selecionada apresente alto tráfego em decorrência da presença de outros fluxos. Outro perfil pode ter entre suas ações a instanciação de uma réplica de um serviço ou função de rede em outro ponto da topologia (e.g. CDN para servidores de *streaming* caso um servidor apresente estresse para atender a um número muito alto de requisições). Entretanto, replicar um conteúdo implica em desviar a rota de certos hosts

para ele. Nestes casos, os perfis devem ser unidos, de forma que tenham a mesma prioridade e o agente aprenda quais ações são melhores para quais situações (neste exemplo, somente desviar a rota, ou instanciar um novo recurso e desviar a rota), por meio de uma recompensa em comum, conforme será visto em um exemplo na Seção 3.3.

Para o restante deste capítulo, serão vistos algumas definições e requisitos na Seção 3.1. Em seguida, a modelagem do sistema que aprende por reforço é mostrada na Seção 3.2. Por fim os conceitos são exemplificados com um estudo de caso na Seção 3.3.

3.1 Definições e requisitos

A rede é definida por um conjunto de links E e um conjunto de nós H, que por sua vez compreende outros quatro subconjuntos: H_s são servidores, H_c são clientes, H_v são VNFs e S são switches. Os servidores podem ser vítimas de ataques e anomalias por parte de clientes.

Um caminho entre dois hosts é compreendido por uma sequência de switches. Assume-se que todos os links que conectam os switches entre si podem trafegar a mesma quantidade de bits em um intervalo de tempo, e os links que conectam hosts a switches tem uma capacidade menor. Nada é assumido a respeito de switches e VNFs, a não ser a condição de não estarem comprometidos.

Não há restrições acerca da topologia. Ela pode ser uma árvore, linha ou conter ciclos. Além disso, o agente de RL deve mitigar anomalias sem necessariamente ter sido treinado previamente para tal. Desde de que um perfil para determinada anomalia tenha sido modelado e implementado, cabe ao agente aprender a como adotar ações daquele perfil de forma a resolver problemas encontrados. O sistema deve, portanto, ser flexível e permitir um número arbitrário de perfis.

3.2 Modelagem da aprendizagem de máquina

Esta seção demonstra como diferentes métricas de rede são usadas para a construção de perfis, bem como os conjuntos de ações e recompensas possíveis para cada perfil. Não há limite de quantos perfis de rede podem existir. Entretanto, um perfil pode estar relacionado a outro, e neste caso uma união entre eles deve ser realizada. Em específico, neste trabalho são estudados três perfis:

1. Balanceamento de carga

2. Carga sobre servidores
3. Ataques de negação de serviço

Cada perfil será visto em detalhes nas próximas seções. A Seção 3.2.1 modela estados, ações e recompensa do perfil de balanceamento de carga. O mesmo é feito nas seções 3.2.2 e 3.2.3 para os perfis de carga sobre servidores e ataques de negação de serviço, respectivamente. Ao final, a Seção 3.2.4 mostra como a recompensa é usada para atualizar as tabelas estado-ação.

3.2.1 Balanceamento de carga

Switches se conectam a links, que são responsáveis por transmitir pacotes. Se os links incidentes a um switch apresentam alta vazão, a fila de espera (*buffer*) do switch pode encher, fazendo com que pacotes sejam descartados. Isso pode prejudicar a experiência de usuários na rede. Portanto, a fim de evitar essa sobrecarga, pode ser interessante encaminhar pacotes por caminhos alternativos. A ideia é que, caso esteja menos congestionada, uma rota mais longa pode ser mais vantajosa do que outra mais curta.

A seguir será vista a modelagem do agente de RL para o perfil de balanceamento de tráfego. Primeiro, na Seção 3.2.1.1 serão apresentadas quais métricas precisam ser coletadas para a modelagem deste perfil. A seguir, na Seção 3.2.1.2, serão mostradas quais são as ações disponíveis para este perfil. Por último, na Seção 3.2.1.3, será abordada a recompensa para a interação do agente com o ambiente neste perfil.

3.2.1.1 Métricas necessárias

Links se conectam aos switches por meios de portas. O agente de RL, a cada ciclo de aprendizagem, coleta informações das tabelas de fluxo para determinar quantos bytes chegaram aos switches por meio de cada link. Para fazer isso, o agente soma os campos *tx_byte*[1] e *rx_byte*[2] de todas entradas de mesma *in_port*. Os campos *tx_byte* e *rx_byte* das tabelas de fluxo são cumulativos, isto é, não possuem a informação de quantos bytes trafegaram em um determinado tempo. Contudo, tal restrição é facilmente contornada calculando a diferença entre os valores encontrados nos ciclos atual e anterior.

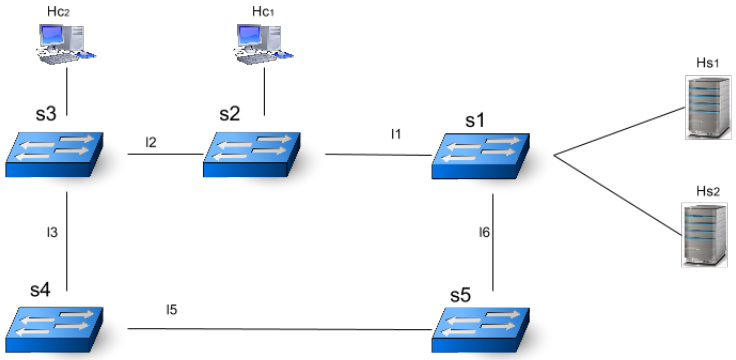
Um caminho é dito congestionado (ou alto) se algum de seus links apresenta tráfego igual ou superior a 80% da capacidade máxima. Senão, é considerado médio se algum de seus

[1]Bytes transmitidos por aquela porta
[2]Bytes recebidos por aquela porta

links apresenta tráfego igual ou superior a 30% da capacidade máxima. Senão, é considerado baixo. Observe que um link pode pertencer a dois caminhos simultaneamente. Se o tráfego gerado por cada fluxo for pequeno, mas somados deixarem o link muito carregado, ambos caminhos apresentarão status congestionado.

A fim de desviar o tráfego entre dois hosts, o agente de RL é auxiliado pelo controlador. Este possui visão global da rede e calcula, para cada host, os caminhos possíveis para os demais hosts e nomeia os caminhos, de forma a identificá-los. Cada estado em que o agente se encontra para este perfil é, dessa forma, uma matriz, em que cada linha possui três colunas: o status de tráfego do caminho (alto, médio ou baixo), a identificação do caminho (uma string que contém o host de origem e o caminho em si) e o identificador do estado. Por exemplo, a Tabela 3.2 mostra um exemplo de conjunto de estados para a topologia apresentada na Figura 3.2.

Figura 3.2: Exemplo de topologia



Fonte: O autor.

Para esta topologia, os caminhos possíveis calculados pelo controlador são os indicados pela Tabela 3.1. Ela mostra que rotas entre hosts diferentes que passam pelos mesmos switches são nomeadas de maneira igual. Exemplos são os caminhos de Hc_1 para Hs_1 e de Hc_1 para Hs_2. Quando a rota passa pelos switches *s2* e *s1*, ambos são nomeados por $Hc_1 - P3$. Quando a rota passa pelos switches *s2, s3, s4, s5* e *s1*, recebem o nome de $Hc_1 - P4$.

Suponha que o host Hc_1 esteja trocando pacotes com o host Hs_1 através do caminho $Hc_1 - P3$. Esta conexão gera um tráfego pequeno no link *l1*. Suponha também que em seguida o cliente Hc_2 estabeleça uma conexão com o host Hs_2 por meio do caminho $Hc_2 - P3$. Esta conexão acarreta em tráfego também pequeno nos links *l2* e *l1*. Contudo, sobre o link *l1* pesam duas conexões. Ainda que cada uma apresente baixa vazão, somadas elas podem formar um alto tráfego. A Tabela 3.2 mostra como ficariam os estados neste cenário.

A medida que novos caminhos são explorados, novos estados são adicionados na tabela de estados. A próxima seção mostrará como quais ações estão disponíveis ao agente e como

Tabela 3.1: Caminhos possíveis

Origem	**Destino**	**Caminho**	**Identificador do caminho**
Hs_1	Hs_2	s1	$Hs_1 - P1$
Hs_1	Hc_1	s1, s2	$Hs_1 - P2$
Hs_1	Hc_1	s1, s5, s4, s3, s2	$Hs_1 - P3$
Hs_1	Hc_2	s1, s2, s3	$Hs_1 - P4$
Hs_1	Hc_2	s1, s5, s4, s3	$Hs_1 - P5$
Hs_2	Hs_1	s1	$Hs_2 - P1$
Hs_2	Hc_1	s1, s2	$Hs_2 - P2$
Hs_2	Hc_1	s1, s5, s4, s3, s2	$Hs_2 - P3$
Hs_2	Hc_2	s1, s2, s3	$Hs_2 - P4$
Hs_2	Hc_2	s1, s5, s4, s3	$Hs_2 - P5$
Hc_1	Hc_2	s2, s3	$Hc_1 - P1$
Hc_1	Hc_2	s2, s1, s5, s4, s3	$Hc_1 - P2$
Hc_1	Hs_1	s2, s1	$Hc_1 - P3$
Hc_1	Hs_1	s2, s3, s4, s5, s1	$Hc_1 - P4$
Hc_1	Hs_2	s2, s1	$Hc_1 - P3$
Hc_1	Hs_2	s2, s3, s4, s5, s1	$Hc_1 - P4$
Hc_2	Hc_1	s3, s2	$Hc_2 - P1$
Hc_2	Hc_1	s3, s4, s5, s1, s2	$Hc_2 - P2$
Hc_2	Hs_1	s3, s2, s1	$Hc_2 - P3$
Hc_2	Hs_1	s3, s4, s5, s1	$Hc_2 - P4$
Hc_2	Hs_2	s3, s2, s1	$Hc_2 - P3$
Hc_2	Hs_2	s3, s4, s5, s1	$Hc_2 - P4$

Fonte: O autor

Tabela 3.2: Instância de tabela de estados

Estado	*Caminho*	*Status*
0	$Hc_1 - P3$	Baixo
1	$Hc_1 - P3$	Alto
	$Hc_2 - P3$	Alto

Fonte: O autor

elas podem alterar a tabela de estados.

3.2.1.2 Conjunto de Ações

O agente de RL tem a sua disposição, em cada *time step*, o estado atual e o conjunto de caminhos possíveis calculado pelo controlador. De posse dessas informações, ele pode solicitar ao controlador modificações nas regras nas tabelas de fluxo dos switches de forma que um caminho seja trocado por outro. Mais especificamente, para o perfil de balanceamento de carga, as ações possíveis consistem em:

1. Ação vazia.

2. Aplicar o caminho mínimo entre dois hosts.
3. Alterar um caminho aplicado entre dois hosts.

O agente somente poderá escolher a terceira ação caso um ou mais links em um caminho supere 80% da sua capacidade de vazão. Se isso não acontecer, pode escolher a ação vazia ou aplicar o caminho mínimo entre dois hosts, caso ele não esteja sendo usado. No exemplo da Seção 3.2.1.1, por exemplo, o agente pode desviar o caminho entre Hc_2 e Hs_2. Em vez de ele seguir o caminho $Hc_2 - P3$, que passa pelos switches *s3*, *s2* e *s1*, poderia adotar um caminho mais longo, mas que evitasse o congestionado link *l1*. Um caminho assim seria o $Hc_2 - P4$, que passa pelos switches *s3 s4*, *s5* e *s1*. A tabela de estados seria atualizada, por exemplo, conforme a Tabela 3.3.

Tabela 3.3: Instância de tabela de estados

Estado	*Caminho*	*Status*
0	$Hc_1 - P3$	Baixo
1	$Hc_1 - P3$	Alto
	$Hc_2 - P3$	Alto
2	$Hc_1 - P3$	Baixo
	$Hc_2 - P4$	Baixo

Fonte: O autor

3.2.1.3 Recompensa

O agente de RL tem dois objetivos: *(i)* evitar que um link em um caminho supere 80% da capacidade total de vazão, e *(ii)* adotar sempre que possível o menor caminho possível.

A recompensa conferida ao agente é de 1 se após executar a ação não houver caminhos que apresentem links com 80% ou mais da capacidade total de vazão -1 caso contrário. Ou, formalmente:

$$R = \begin{cases} -1, & \text{se há link com tráfico superior a 80\% da capacidade de vazão} \\ 1, & \text{caso contrário} \end{cases} \tag{3.1}$$

Intuitivamente, o agente procurará realizar o balanceamento de forma que não haja sobrecarga nos links. Para isso, pode ser necessário alterar a rota de determinados fluxos de forma que eles não sigam o caminho mínimo entre origem e destino. Entretanto, em algum momento o agente terá que retornar os fluxos desviados para o caminho mínimo. Ele pode fazer isso prematuramente, e ser punido por isso, ou recompensado. Com o passar dos ciclos, aprenderá

quais estados permitem o retorno ao caminho mínimo.

3.2.2 Carga sobre servidores

Servidores fornecem serviços para clientes consumirem. Por mais que um determinado serviço fornecido pelo servidor possa ser paralelizado para atender diversos pedidos, o servidor pode ficar sobrecarregado caso receba muitas requisições. Para contornar este problema, VNFs que replicam este serviço são instanciadas em outros pontos na rede e absorvem parte das requisições que sobrecarregavam o servidor ou seu link de acesso. A seguir será vista a modelagem dos estados para este caso, bem como o conjunto de ações e a recompensa.

3.2.2.1 Construção dos estados

A construção do perfil de carga sobre servidores ocorre por meio de três métricas:

1. O tráfego no link de acesso entre servidor e switch.
2. A relação de servidores ou VNFs de réplicas.
3. Os caminhos incidentes ao servidor ou réplicas.

Como o tráfego em um link possui muitos valores possíveis, ele é dividido em três faixas: *baixo*, ou $[0\%, 30\%)$ de uso; *médio*, ou $[30\%, 80\%)$; e *alto*, ou $[80\%, 100]$.

A relação de VNFs é coletada pelo próprio catálogo de VNFs do orquestrador NFV. As VNFs não são, contudo, colocadas de forma totalmente aleatória pela rede. Elas são conectadas somente aos switches que fazem parte de caminhos utilizados para se chegar ao servidor.

Por exemplo, a partir da Figura 3.2, se o link que dá acesso ao servidor Hs_1 para o switch *s1* está sobrecarregado em decorrência de conexões dos clientes Hc_1 e Hc_2, que utilizam os caminhos $Hc_1 - P3$ e $Hc_2 - P3$ (vide Tabela 3.1), que são seus caminhos mínimos, os switches que podem receber réplicas deste servidor são *s3, s2* e *s1*. Ao acrescentar uma réplica, pode ser necessário atualizar a tabela de caminhos possíveis.

O número de estados possíveis dependerá do tamanho da rede: quanto maior ela for, mais estados podem existir da combinação das três métricas supracitadas. A Tabela 3.4 mostra um exemplo em que uma réplica é instanciada no switch *s2* e uma nova rota é acrescentada entre o cliente Hc_2 e esta nova réplica no estado 2.

A tabela ganha entradas a medida em que o número de VNFs é alterada, a carga no link de acesso muda, ou caminhos são modificados. Naturalmente, não há a necessidade de pré-

Tabela 3.4: Exemplo de tabela de estados

Estado	*Servidor*	*Status link*	*caminhos incidentes*
0	Hs_1	Médio	$Hc_1 - P3$
1	Hs_1	Alto	$Hc_1 - P3$
	Hs_1	Alto	$Hc_2 - P3$
2	Hs_1	Médio	$Hc_1 - P3$
	Hs_3	Baixo	$Hc_2 - P5$
...	...	...	...

Fonte: O autor

calcular todas as combinações possíveis, e sim utilizar estruturas de dados que dinamicamente acrescentem os novos estados a medida em que eles forem verificados pela primeira vez.

3.2.2.2 Conjunto de ações

Para o perfil de carga nos servidores, as ações possíveis consistem em:

1. Instanciar uma réplica de servidor.
2. Remover uma réplica instanciada.
3. Ação vazia.

A remoção de uma réplica só ocorre quando o link de acesso ao servidor reporta baixo tráfego. Quando há alta vazão, espera-se que uma necessidade maior de instanciações do que em média vazão. O agente vai aprender quantas réplicas são necessárias instalar e em quais switches para balancear o tráfego da rede de acordo com a carga sobre o link de acesso dos servidores.

3.2.2.3 Recompensa

O agente de RL tem dois objetivos: *(i)* manter o link de acesso do servidor ao switch em um estado de baixo tráfego *(ii)* usando o mínimo de recursos possível.

A recompensa conferida ao agente é de 1 se após executar a ação o link de acesso do servidor ao switch estiver em um estado de baixo tráfego ou -1 caso contrário. Ou, formalmente:

$$R = \begin{cases} 1, & \text{se tráfego em link de acesso for baixo} \\ -1, & \text{caso contráro} \end{cases} \tag{3.2}$$

Intuitivamente, conforme o volume de tráfego varia, o agente procura a configuração mínima de VNFs que alivie o estresse sobre os links de acesso dos servidores à rede de forma a

mantê-los com status de baixa vazão.

É possível que o agente execute uma ação que leve um servidor que apresentava baixo consumo para médio consumo, por exemplo. Isso acontece porque tal link estava com baixo nível de tráfego por causa das VNFs instanciadas. Ao explorar uma configuração com menos VNFs para aquela situação de tráfego, o link ficou sobrecarregado e o agente foi punido por isso com uma recompensa negativa. No longo prazo, aprenderá a como reagir frente a cada configuração de tráfego.

3.2.3 Detecção e mitigação

Ao contrário de balanceamento de carga ou réplica de servidores, em que o agente deve aprender a melhor maneira de balancear o tráfego ou onde instanciar uma réplica, em um cenário de ataque não existe outra alternativa senão mitigar os fluxos maliciosos. Tentar absorver o ataque balanceando o tráfego pode se tornar uma tarefa impraticável, ou demandar a instanciação de muitos recursos. Faz sentido que este perfil receba uma prioridade maior do que os outros dois abordados anteriormente. O agente deve ser capaz de distinguir fluxos maliciosos de legítimos, de forma que mitigue os ataques e deixe os trabalho de aprendizagem somente para os pacotes legítimos. Este perfil não adota aprendizagem de máquina em sua solução, e assim não há uma tabela estado-ação. Técnicas mais avançadas de detecção e mitigação de ataques de negação de serviço existem, mas para o escopo deste trabalho se adota uma heurística mais simples para lidar com eles. Este perfil possui somente dois estados, portanto:

1. Malicioso
2. Legítimo

Quando este perfil se encontra no estado malicioso, a única ação possível é mitigar o ataque. Ou seja, em uma situação de ataque, o foco do agente passa a ser derrubar o ataque. Enquanto um ataque não for detectado, o agente executa a ação vazia para este perfil, podendo executar outras ações em outros perfis de prioridade mais baixa, se houverem.

O controlador mantém registro da média de requisições que cada servidor recebe, bem como seu desvio padrão. Ele obtém essa informação verificando nas tabelas de fluxo dos switches quais fluxos são destinados a quais servidores. O controlador também mantém em uma tabela um registro a respeito de a quais switches cada host, identificado pelo endereço IP, se conecta. Uma aplicação executando no plano de aplicações do controlador coleta essas informações e, caso em três *time steps* consecutivos o número de requisições recebido por um servidor

seja superior a sua média mais um desvio, um ataque é considerado em curso. O próximo passo da aplicação é mitigar o ataque, selecionando os fluxos considerados maliciosos.

Para isso, é adotada a seguinte heurística: sendo *r* o número normal de requisições que o servidor recebe, e c o número de clientes que está conectado a ele, a aplicação assume que em média cada cliente deve solicitar r/c requisições. Clientes acima desse limiar são barrados. Busca-se no fluxo a informação relativa ao endereço IP de origem desses clientes e consulta-se a tabela para saber em quais switches eles estão conectados. VNFs de *firewall* são instanciadas em tais switches, e recebem do agente ordem de barrar os fluxos vindos dos hosts suspeitos.

3.2.4 Atualização da tabela estado-ação

Após realizar uma ação para um perfil que adota aprendizagem por reforço, o agente recebe uma recompensa, que informa o quão boa, ou ruim, foi aquela decisão para aquele estado. A forma como o agente usa a recompensa para atualizar a tabela estado-ação segue o algoritmo Sarsa (SUTTON; BARTO, 2012). Ele considera para a aprendizagem transições entre pares estado-valor durante um episódio. Cada letra de seu nome é uma alusão aos elementos que compõem a aprendizagem: $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$, ou estado e ação no *time step* atual, a recompensa adquirida no *time step seguinte*, e o par estado-ação no estado seguinte. A Equação 3.3 mostra como a atualização dos valores dos pares estado-ação é realizada.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.3}$$

onde α é a taxa de aprendizagem, γ é fator de desconto para recompensas, r é a recompensa e $Q(s, a)$ é a tabela com os valores aprendidos para cada tupla estado-ação.

Devido à natureza do comportamento de uma rede, em que seu estado não necessariamente é determinado pelas ações do agente no *time step* anterior (por exemplo, um ataque de negação de serviço pode começar e terminar subitamente), somente recompensas imediatas são levadas em consideração. Dessa forma, $\gamma = 0$, e a fórmula de atualização da tabela estado-ação é levemente alterada, conforme mostra a Equação 3.4.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} - Q(s_t, a_t)] \tag{3.4}$$

O algoritmo, em pseudo-código, é apresentado a seguir:

Enquanto aprende, o agente se depara com um *trade-off* entre adotar uma ação cuja recompensa é conhecida ou explorar novas possibilidades e adquirir mais conhecimento sobre

Algorithm 1: SARSA

Input: Q-table
Output: Q-table atualizada

$\forall s \in S, \forall a \in A, Q(s,a) = 0$ /* Inicializar Q(s,a) */
foreach *episódio* **do**
  Escolha $a \in A$ disponível em $s \in S$ segundo uma política (e.g. ϵ-*greedy*)
  foreach *time step do episódio* **do**
    Execute a ação a, observe s', r
    Escolha $a' \in A$ disponível em $s' \in S$ segundo uma política (e.g. ϵ-*greedy*)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r - Q(s,a)]$
    $S \leftarrow S', A \leftarrow A'$

as consequências de suas ações. Uma técnica popular e efetiva que lida com este dilema é ϵ-*greedy* (SUTTON; BARTO, 2012). Nela, um fator ϵ indica o quanto um agente irá preferir testar novas possibilidades frente a opções já conhecidas. Por exemplo, se $\epsilon = 0.1$, significa que em 10% dos casos o agente tentará uma abordagem desconhecida.

Inicialmente, os valores das tabelas estado-ação, chamados de *q-values*, são inicializados com 0. Como exemplo de demonstração, supondo três recompensas positivas e duas negativas para um determinado par <estado, ação> (o estado *Si* por ser qualquer um dos estados de exemplo do perfil de balanceamento, e a ação *Aj* pode ser a troca de um caminho por outro, por exemplo), os valores da tabela para aquele par seriam os seguintes (supondo $\alpha = 0.1$):

- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.1$
- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.19$
- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.271$
- Q(S_i, A_j) $= 0 + 0.1 * (-1 - 0) = 0.143$
- Q(S_i, A_j) $= 0 + 0.1 * (-1 - 0) = 0.028$

Com uma taxa de aprendizagem $\alpha = 0.9$, as mudanças nos valores são muito mais bruscas. Rapidamente, valores antes positivos se tornam negativos.

- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.9$
- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.99$
- Q(S_i, A_j) $= 0 + 0.1 * (1 - 0) = 0.999$
- Q(S_i, A_j) $= 0 + 0.1 * (-1 - 0) = -0.8$
- Q(S_i, A_j) $= 0 + 0.1 * (-1 - 0) = -0.98$

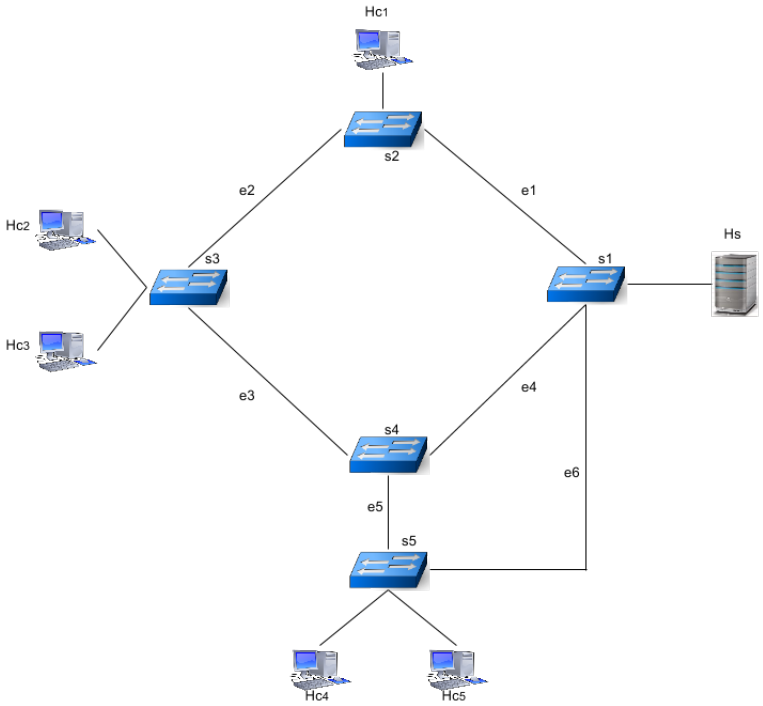
Em um rede de computadores, a tomada de uma ação em um determinado estado pode resultar em uma recompensa negativa, mas em outro momento a mesma ação no mesmo es-

tado pode gerar uma recompensa positiva. Isso acontece porque o ambiente é estocástico, e justamente no momento que o agente escolhe realizar uma ação, um evento inesperado pode acontecer e ser o fator dominante para a recompensa. Contudo, no longo prazo, ao executar várias vezes a mesma ação naquele estado, tais anomalias tendem a perturbar pouco o *q-value*, principalmente se a taxa de aprendizagem for baixa - o que é adotada daqui para frente.

3.3 Estudo de caso

A seguir é apresentado um estudo de como o sistema proposto se comportará em um determinado episódio composto por n *time steps*, com uma topologia conforme a exibida na Figura 3.3.

Figura 3.3: Exemplo de topologia



Fonte: O autor.

Inicialmente não há VNFs de réplicas instanciadas na rede, que apresenta um servidor H_s conectado ao switch $s1$, 5 hosts clientes $\{Hc_1...Hc_5\}$ conectados aos switches *s2, s3* e *s5* e seis links $\{e_1...e_6\}$ que conectam os switches entre si. O agente de RL interage com o ambiente a cada *time step*, que pode ter um tempo arbitrário (e.g. *2 s*). Configura-se $\alpha = 0.1$, uma taxa de aprendizagem baixa, mas que, se um determinado par estado-ação apresentar um alto valor, significa que por várias iterações a recompensa foi positiva, o que reforça ao agente a evidência de que tal ação naquele estado é uma boa alternativa. Opta-se por $\epsilon = 0.4$, de forma

que o agente procurará explorar prioritariamente uma ação conhecida, mas em momentos não tão raros buscará explorar novas possibilidades e adquirir mais conhecimento sobre o ambiente. Por fim, conforme definido na Seção 3.2.4, $\gamma = 0$.

O servidor oferece arquivos de vídeo a clientes. Serão analisados o comportamento da rede para os três perfis destacados nas seções 3.2.1, 3.2.2 e 3.2.3. O perfil de ataque terá prioridade 0, ou seja, será o mais prioritário: se um ataque ocorrer, o estado daquele perfil passa a ser malicioso, e a política adotada pelo agente será mitigar o ataque.

Os dois perfis restantes, de balanceamento de carga e réplica de servidor, são relacionados: replicar um servidor em outro ponto da rede implica também em balancear parte do tráfego para aquela nova réplica. Portanto, eles recebem a mesma prioridade e têm suas tabelas unidas, conforme a Tabela 3.6. O objetivo do agente é manter a rede operante utilizando a menor quantidade de recursos possível. Ou seja, persiste a ideia de sempre que possível remover réplicas de servidores e adotar o caminho mínimo. Além disso, o fato de o perfil de ataque ter prioridade máxima faz com que o agente balanceie e replique servidores somente para fluxos legítimos, evitando o uso de recursos para atender fluxos maliciosos.

A recompensa é é negativa caso, após executar uma ação, algum link de acesso de servidor a switch apresente status alto ou médio, ou caso algum link entre switches esteja em status alto. Ela é positiva se isso não ocorrer.

$$R = \begin{cases} -1, & \text{se há link de acesso de servidor a switch em status alto ou médio} \\ -1, & \text{se há link entre switches em status alto} \\ 1, & \text{caso contrário} \end{cases} \tag{3.5}$$

3.3.1 Passo a passo dos ciclos de aprendizagem

É apresentado a seguir um exemplo simulado de execução do algoritmo para um episódio. Primeiramente há uma situação de normalidade, seguida de um aumento legítimo no tráfego, que é balanceado com rotas alternativas e instanciação de réplicas, e depois ocorre um ataque ao servidor, seguido de ações para evitar que os links fiquem sobrecarregados.

Para a rede exibida na Figura 3.3, o controlador inicialmente calcula os caminhos possíveis. Por questões de concisão e legibilidade, a Tabela 3.5 mostra somente os caminhos possíveis entre os clientes e o servidor.

Suponha que no instante *t = 0* o cliente Hc_1 estabeleça conexão com o servidor Hs por

Tabela 3.5: Caminhos possíveis

Caminho	**Switches**
$Hc_1 - P1$	s2, s1
$Hc_1 - P2$	s2, s3, s4, s1
$Hc_1 - P3$	s2, s3, s4, s5, s1
$Hc_2 - P1$	s3, s2, s1
$Hc_2 - P2$	s3, s4, s1
$Hc_2 - P3$	s3, s4,, s5, s1
$Hc_3 - P1$	s3, s2, s1
$Hc_3 - P2$	s3, s4, s1
$Hc_3 - P3$	s3, s4,, s5, s1
$Hc_4 - P1$	s5, s1
$Hc_4 - P2$	s5, s4, s1
$Hc_4 - P3$	s5, s4, s3, s2, s1
$Hc_5 - P1$	s5, s1
$Hc_5 - P2$	s5, s4, s1
$Hc_5 - P3$	s5, s4, s3, s2, s1

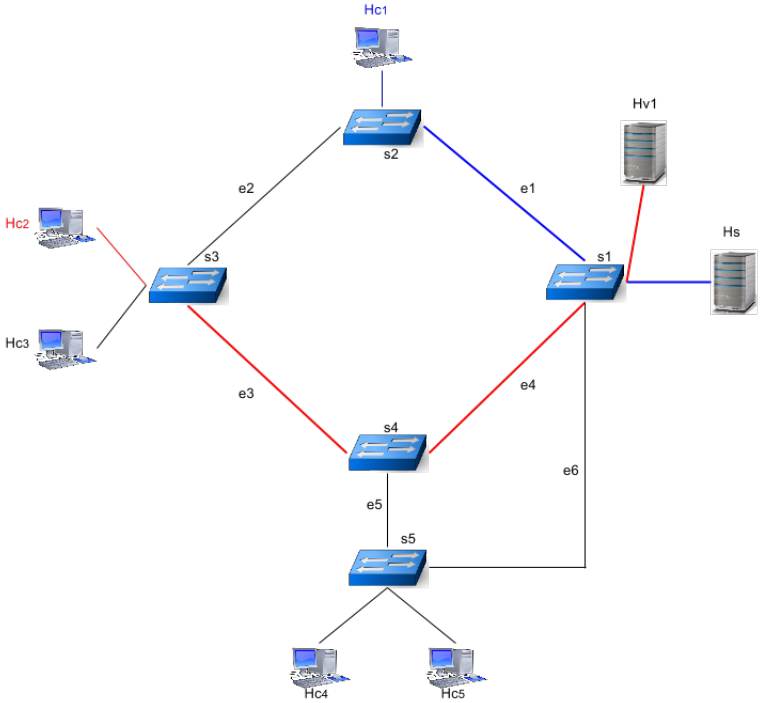
Fonte: O autor

meio do caminho $H_{c1} - P1$, nenhum link é sobrecarregado, não é verificado ataque, a ação vazia é tomada e assim a recompensa é positiva. Contudo, em um instante t_i ($i > 0$), o cliente Hc_2 também inicia conexão com o mesmo servidor, pelo caminho $H_{c2} - P1$, recebendo um arquivo enquanto a conexão do outro cliente não tenha terminado. O caminho sobrecarrega o link *e1* e também o link que conecta o servidor Hs ao switch *s1*, e dessa forma o agente recebe uma recompensa negativa.

O agente então adota a ação de replicar o servidor Hs. Neste caso, isso alivia a carga sobre o link de acesso de Hs à rede, mas o link *e1* segue sobrecarregado, ocasionando uma recompensa negativa ao chegar no estado 2. Então, o agente adota uma solução de balanceameto de tráfego, trocando o caminho $Hc_2 - P1$ para $Hc_2 - P2$. Agora sim a recompensa é positiva, pois nenhum link apresenta gargalo no *time step* t_j ($j > i$). A Figura 3.4 exibe a rede após as intervenções do agente. As tabelas 3.6 e 3.7 mostram, respectivamente, os estados visitados e os resultados das ações tomadas.

Suponha agora que, em um *time step* futuro t_k ($k > j$), após várias interações com o ambiente, o agente se encontre no estado *p* e a configuração da rede seja conforme a apresentada pela Figura 3.5: os clientes Hc_1 e Hc_2 e estabelecem conexão com o servidor Hs, e o cliente Hc_4 faça o mesmo, mas com o servidor Hv_1, todos por meio de seus caminhos mínimos. Suponha ainda que o tráfego gerado pelo cliente Hc_1 é baixo, mas combinado com o cliente Hc_2 ocasione um tráfego alto no links de acesso servidor. O agente explora a ação vazia e obtém recompensa negativa. Em seguida, no *time step* seguinte, o cliente Hc_5 também inicia

Figura 3.4: Resultado de réplica de servidor



Fonte: O autor.

Tabela 3.6: Instância de tabela de estados

Estado	*Caminho*	*Status caminho*	*Servidor*	*link servidor*
0	$Hc_1 - P1$	Baixo	Hs	Baixo
1	$Hc_1 - P1$	Alto	Hs	Alto
	$Hc_2 - P1$	Alto	Hs	Alto
2	$Hc_1 - P1$	Alto	Hs	Baixo
	$Hc_2 - P1$	Alto	Hv_1	Baixo
3	$Hc_1 - P1$	Baixo	Hs	Baixo
	$Hc_2 - P2$	Baixo	Hv_1	Baixo

Fonte: O autor

uma conexão com o servidor Hv_1 pelo seu caminho mínimo, e esta conexão ocasiona um tráfego de média intensidade no link $e6$, mas tráfego considerado alto para o link de acesso de Hv_1 ao switch *s1*.

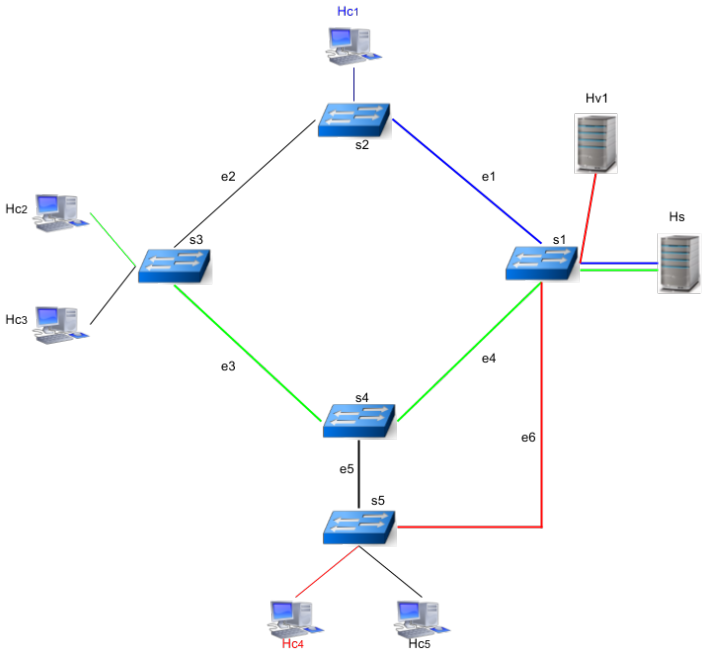
Este é o estado *q*, e o agente procura uma rota alternativa para acomodar este novo cliente, mas não importa a ação escolhida: a recompensa é sempre negativa, ações de balanceamento não alteram o destino dos fluxos, somente os caminhos até ele. Após tentativas infrutíferas, o agente se encontra no estado *r*, e a alternativa adotada é replicar o servidor. Ele assim o faz, conectando uma réplica ao switch *s4*. O cliente Hc_5 se conecta a esta réplica e o agente progride ao estado *s*. É necessário acrescentar à tabela de caminhos, obviamente, o caminho entre Hc_5 e a nova réplica Hv_2, que passa pelo switches *s5* e *s4*. Este caminho recebe o nome de $Hc_5 - P4$. A recompensa é negativa pois o link de acesso do servidor Hs ainda apresenta

Tabela 3.7: Instância de tabela estado-ação

Estado	Ação	Q-Value
0	Ação vazia	-0.1
1	Replicar Hs em *s1*	-0.1
2	Mudar rota de Hc_2 para $Hc_2 - P2$	0.1

Fonte: O autor

Figura 3.5: Situação da rede no momento da requisição de Hc_5



Fonte: O autor.

status alto. O agente então replica o servidor Hs no switch *s1*, desviando o destino de Hc_2 para ele e nenhum servidor apresenta gargalo em seu link de acesso, gerando uma recompensa positiva ao chegar no estado *t*.

Aqui fica explícita a necessidade de se unir os perfis de réplica e balanceamento. As ações de replicar servidores podem auxiliar o agente a aliviar o estresse sobre um link instanciando uma réplica em outro ponto da topologia. Se os perfis não fossem unidos e o perfil de balanceamento tivesse maior prioridade em relação ao perfil de réplica, o agente poderia jamais encontrar a solução para uma anomalia deste tipo, pois seria ignorante em relação às recompensas do perfil de réplica e insistiria somente no balanceamento puro de tráfego.

As tabelas 3.8 e 3.9 resumem o resultado das suas ações neste momento crítico, bem como os estados percorridos pelo agente.

Além disso, a Figura 3.6 apresenta a configuração resultante da rede. Em seguida, em um *time step* t_l ($l > k$) o cliente Hc_3 estabelece conexão com o servidor Hs pelo caminho $Hc_3 - P2$. Essa conexão é seguida de várias outras iniciadas pelo mesmo host, e levam a uma sobrecarga nos links $e3$ e $e4$, bem como no link de acesso do servidor à rede em alguns *time*

Tabela 3.8: Instância de tabela estado-ação

Estado	**Ação**	**Q-Value**
...	...	...
p	Ação vazia	-0.1
q	Balancear $Hc_5 - P1$ para $Hc_5 - P2$	-0.1
r	Replicar Hs em *s4*	-0.1
s	Replicar Hs em *s1*	0.1

Fonte: O autor

Tabela 3.9: Instância de tabela de estados

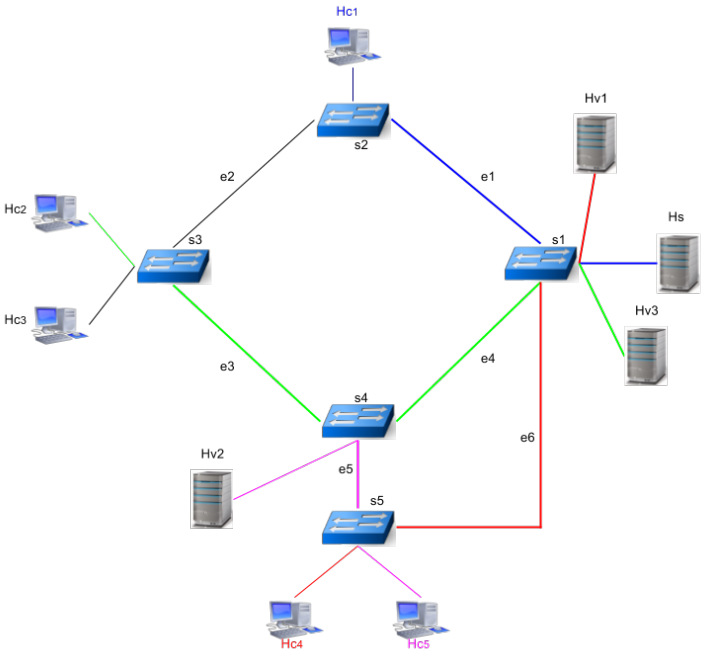
Estado	*Caminho*	*Status caminho*	*Servidor*	*link servidor*
...	...	Baixo	...	...
p	$Hc_1 - P1$	Baixo	Hs	Médio
	$Hc_2 - P2$	Baixo	Hs	Médio
	$Hc_4 - P1$	Baixo	Hv_1	Baixo
q	$Hc_1 - P1$	Baixo	Hs	Médio
	$Hc_2 - P2$	Baixo	Hs	Médio
	$Hc_4 - P1$	Alto	Hv_1	Alto
	$Hc_5 - P1$	Alto	Hv_1	Alto
r	$Hc_1 - P1$	Baixo	Hs	Médio
	$Hc_2 - P2$	Alto	Hs	Médio
	$Hc_4 - P1$	Baixo	Hv_1	Alto
	$Hc_5 - P2$	Alto	Hv_1	Alto
s	$Hc_1 - P1$	Baixo	Hs	Médio
	$Hc_2 - P2$	Baixo	Hs	Médio
	$Hc_4 - P1$	Baixo	Hv_1	Baixo
	$Hc_5 - P4$	Baixo	Hv_2	Baixo
t	$Hc_1 - P1$	Baixo	Hs	Baixo
	$Hc_2 - P2$	Baixo	Hv_3	Baixo
	$Hc_4 - P1$	Baixo	Hv_1	Baixo
	$Hc_5 - P4$	Baixo	Hv_2	Baixo

Fonte: O autor

steps: trata-se de um ataque que este host está aplicando. Portanto, como o perfil de ataque possui prioridade caso seu estado seja malicioso, o agente não buscará balancear o tráfego nem replicar servidores de forma a atender a entrada deste novo cliente, e sim barrar os fluxos maliciosos primeiro. Seguindo o discutido na Seção 3.2.3, uma VNF de *firewall* é instanciada junto ao switch *s5*, onde o cliente mal-intencionado está conectado. O agente então ordena que ele barre os tráfego envolvendo este cliente, e a anomalia é dessa forma eliminada.

Em seguida, há somente fluxos legítimos trafegando na rede, e o agente se encontra novamente no estado *t*. Os fluxos geram um tráfego baixo, e existe uma disposição de algumas VNFs pela rede, o que pode significar recursos desnecessários instanciados ou recursos necessários para manter o servidor em um estado baixo. Cabe ao agente encontrar a configura-

Figura 3.6: Situação da rede acomodando os clientes



Fonte: O autor.

ção mínima que mantenha os links de acesso dos servidores aos switches com baixa ou média vazão.

Aqui acontece o dilema *exploitation* vs *exploration*: o agente sabe que, ao optar pela ação vazia, deve ganhar uma recompensa alta, porque os servidores atuais dão conta da demanda, que é baixa. Entretanto, nada sabe a respeito da remoção de VNFs, que podem estar instanciadas em excesso (em função de algum host que cesse suas atividades, por exemplo). Conforme definido no começo da Seção 3.3, $\epsilon = 0.4$, ou seja, o agente buscará adotar um comportamento conservador, dando prioridade para executar ações conhecidas, mas permitindo que em um percentual não tão baixo de vezes, explore ações desconhecidas. Ele decide então remover a VNF Hv_2 e progride para o estado *u*. O host que a ela se conectava já não gerava o consumo de muitos recursos, e a remoção não causou grandes problemas nos demais servidores. Assim, a recompensa foi positiva.

Agora no estado *u*, o agente executa a remoção da VNF Hv_1 e ultrapassa a fronteira da configuração mínima necessária, sendo punido por isso e vai ao estado *v*. Porém, mais tarde ele consegue conecta uma VNF de réplica de servidor ao switch *s1*. Dessa forma, regressa ao estado *u* e obtém uma recompensa positiva, conforme mostram as tabelas 3.10 e 3.11, de estados e estado-ação, respectivamente.

Tabela 3.10: Instância de tabela de estados

Estado	*Caminho*	*Status*	*Servidor*	*link servidor*
...	...	...	...	...
t	$Hc_1 - P1$	Baixo	Hs	Baixo
	$Hc_2 - P2$	Baixo	Hv_3	Baixo
	$Hc_4 - P1$	Baixo	Hv_1	Baixo
	$Hc_5 - P4$	Baixo	Hv_2	Baixo
u	$Hc_1 - P1$	Baixo	Hs	Baixo
	$Hc_2 - P2$	Baixo	Hv_3	Baixo
	$Hc_3 - P2$	Baixo	Hs	Baixo
	$Hc_4 - P1$	Médio	Hv_1	Baixo
	$Hc_5 - P1$	Médio	Hv_1	Baixo
v	$Hc_1 - P1$	Baixo	Hs	Médio
	$Hc_2 - P2$	Baixo	Hv_3	Baixo
	$Hc_3 - P2$	Baixo	Hs	Médio
	$Hc_4 - P1$	Médio	Hs	Médio
	$Hc_5 - P1$	Médio	Hs	Médio

Fonte: O autor

Tabela 3.11: Instância de tabela estado-ação

Estado	**Ação**	**Q-Value**
...	...	...
t	Ação vazia	0.1
t	Remover Hv_2 em *s4*	0.1
u	Remover Hv_1 em *s1*	-0.1
v	Replicar Hv_1 em *s1*	0.1

Fonte: O autor

4 IMPLEMENTAÇÃO E ANÁLISE DE RESULTADOS

Este capítulo descreve a implementação de um protótipo do modelo apresentado no Capítulo 3, bem como exibe a análise de resultados para os testes realizados.

4.1 Implementação do protótipo

Esta seção mostra como o protótipo foi desenvolvido, abordando quais tecnologias, ferramentas e linguagens foram adotadas e como foram integradas.

4.1.1 Integração do ambiente

Para a realização de testes sobre o sistema proposto, foi utilizada a plataforma Mininet[1], que simula uma SDN. Nela, hosts são processos e trocam pacotes entre si por meio de switches virtuais que fornecem suporte ao protocolo OpenFlow. Em conjunto com a plataforma, foi utilizado o framework Pox[2] (que implementa a versão 1.0 do OpenFlow) para o desenvolvimento do controlador.

Contudo, Mininet não provê suporte nativo para NFV. Tipicamente a tecnologia NFV adota máquinas virtuais para o desenvolvimento de funções de rede, mas tais contêineres têm se mostrado uma alternativa competitiva (RAHO et al., 2015). As funções de rede portanto foram implementadas em contêineres Docker[3], tecnologia surgida em 2013. Seus contêineres são baseados em Linux, compartilham o kernel do sistema hospedeiro, mas proveem isolamento entre as aplicações como é esperado no caso de máquinas virtuais e apresentam boa performance (FELTER et al., 2015).

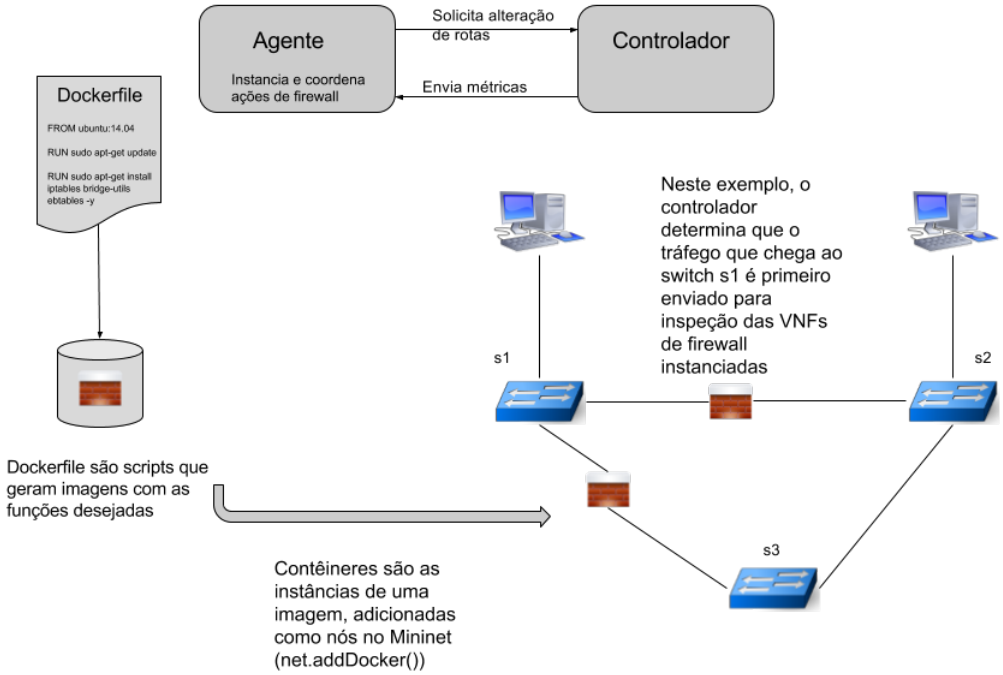
Além disso, hosts usuais no Mininet compartilham o sistema de arquivos, o que pode ser perigoso: uma vez que um host deleta um arquivo, outro host perde acesso a ele também. Contêineres Docker integrados ao Mininet[4] resolvem esse problema, pois cada um possui sua própria estrutura de diretórios e arquivos. Contêineres podem desempenhar tarefas de alto nível, como servidores ou réplicas, bem como funções de rede, como *firewalls*.

Inicialmente é criada uma imagem para cada tipo de contêiner por meio de Dockerfiles.

[1] http://mininet.org/
[2] http://sdnhub.org/tutorials/pox/
[3] https://www.docker.com/what-docker
[4] https://github.com/mpeuster/containernet

No caso, foram implementados Dockerfiles que geram uma imagem de *firewall* que barra fluxos de um dado host e outra imagem de servidor que envia vídeos a clientes. Um contêiner é uma instância de uma imagem. Dessa forma, contêineres originados da imagem de um *firewall* são instanciados no Mininet diversas vezes, dando origem a múltiplos *firewalls*, por exemplo. A Figura 4.1 exemplifica este processo.

Figura 4.1: Integração Docker + Mininet



Fonte: O autor.

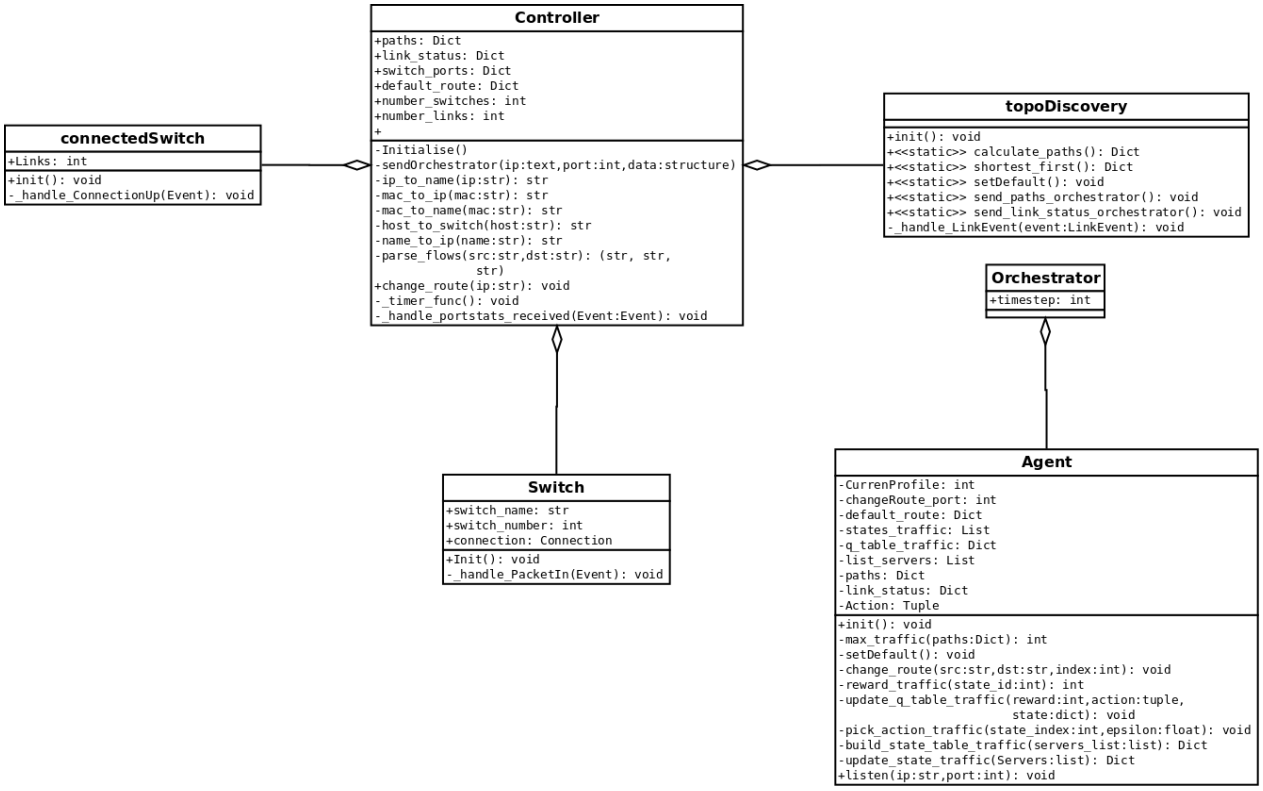
Docker provê a infraestrutura básica para hospedar funções de rede, mas não a orquestração das mesmas. Na implementação feita, esta tarefa é desempenhada conjuntamente por controlador e agente de RL. O controlador é quem determina que pacotes passem por *firewalls* instanciados, por exemplo, antes de chegar ao seu destino. O controlador também é quem coleta métricas da rede e as envia por mensagens TCP para o agente construir os estados da rede. É o agente, por outro lado, quem realiza as chamadas necessárias para instanciar as VNFs definidas por Dockerfiles e ordena que um *firewall* barre determinados fluxos. O agente também envia ao controlador, por mensagens TCP, solicitação de ações como a alteração da rota entre um par de hosts.

4.1.2 Protótipo

O protótipo desenvolvido abrange os perfis de mitigação de ataques e balanceamento de carga descritos na Seção 3.2.1 e 3.2.3. Portanto, foram implementados Dockerfiles que

implementam VNFs de *firewall* e servidor que envia vídeos requeridos por clientes. A Figura 4.2 mostra o diagrama de classes dos componentes implementados. As relações de componentes prontos, como as classes Mininet ou módulos do Pox já implementados por padrão, foram omitidas.

Figura 4.2: Diagrama de classes.



Fonte: O autor.

O controlador monitora as tabelas de fluxos dos switches e envia as métricas necessárias para a formação dos estados em cada *time step* à classe Orchestrator, onde reside o agente de RL. Devido à visão global da rede, o controlador também calcula os caminhos possíveis entre os hosts e envia essa relação ao agente. As trocas de mensagens entre agente e controlador ocorrem por meio de conexões TCP.

O agente é responsável pela tomada de decisões acerca de qual ação tomar com base no estado observado, o que resulta em uma recompensa. Ele mantém uma tabela estado-ação, chamada de Q-Table, que é atualizada a cada iteração.

A Q-Table é um dicionário em Python, cuja chave é uma tupla (*id_estado*, ação), e o valor é a recompensa para aquela ação naquele estado, atualizado conforme o algoritmo Sarsa, descrito em maiores detalhes na Seção 3.2.4. O *id_estado* é um número inteiro, e a ação é ou o valor *None*, para ação vazia, ou então uma tupla, sendo o primeiro valor outra tupla (contendo um par de hosts) e o outro valor o novo caminho a ser adotado na troca de pacotes entre eles.

A seguir serão vistos resultados de dois casos de teste. O primeiro, na Seção 4.2.1.1, é uma prova de conceito que realiza mitigação de ataque e balanceamento de carga. O segundo, na Seção 4.2.1.2, vê com pouco mais de detalhes o balanceamento de carga.

4.2 Experimentos e análise de resultados

Para realizar os experimentos, foi utilizado um computador com processador Intel i7 de 3,4 Ghz, 16 GB de Ram e uma máquina virtual executando o sistema operacional Ubuntu 14.04 (dos quais 10 GB de Ram foram alocados para ela).

4.2.1 Descrição do ambiente e experimentos

Servidores são contêineres Docker. Eles enviam arquivos de vídeo a clientes para gerar tráfego por meio de conexões TCP. *Firewalls* também são abrigados por contêineres. Clientes são hosts usuais do Mininet. Os links que conectam clientes e servidores a switches têm vazão máxima de 5 Mbps. Os demais possuem capacidade para até 11 Mbps.

Foram realizados dois tipos de experimentos. No primeiro o agente procura balancear o tráfego em direção a dois servidores e precisa também mitigar um ataque. Dessa forma trabalha-se a prioridade entre perfis. O segundo caso aborda somente o balanceamento, mas em uma topologia maior e que portanto permite maiores combinações de rotas alternativas.

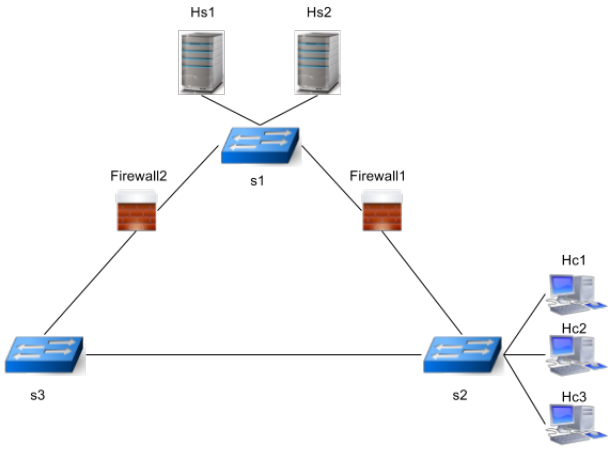
Foi adotada uma taxa de aprendizagem $\alpha = 0.1$, taxa de exploração $\epsilon = 0.4$ (ou seja, em 60% dos casos o agente escolheria a melhor ação já conhecida para o estado em que se encontra e em 40% ele explora uma ação de consequências desconhecidas) e dois segundos de duração para cada *time step*. Foi conferida alta prioridade ao perfil de mitigação e baixa prioridade ao perfil de balanceamento.

4.2.1.1 Mitigação e balanceamento

Para este cenário foi gerada uma topologia conforme a exibida na Figura 4.3. Ela contém três switches que formam um clique. Dois servidores estão conectados ao switch *s1*. Duas VNFs que implementam um *firewall* têm a função de proteger os servidores de um eventual ataque vindo através dos switches *s2* e *s3* e foram deixadas instanciadas por padrão para fins de simplificação de implementação. As VNFs foram implementadas em contêineres Docker por

meio das ferramentas *bridge-utils*[5] e *ebtables*[6].

Figura 4.3: Topologia



Fonte: O autor.

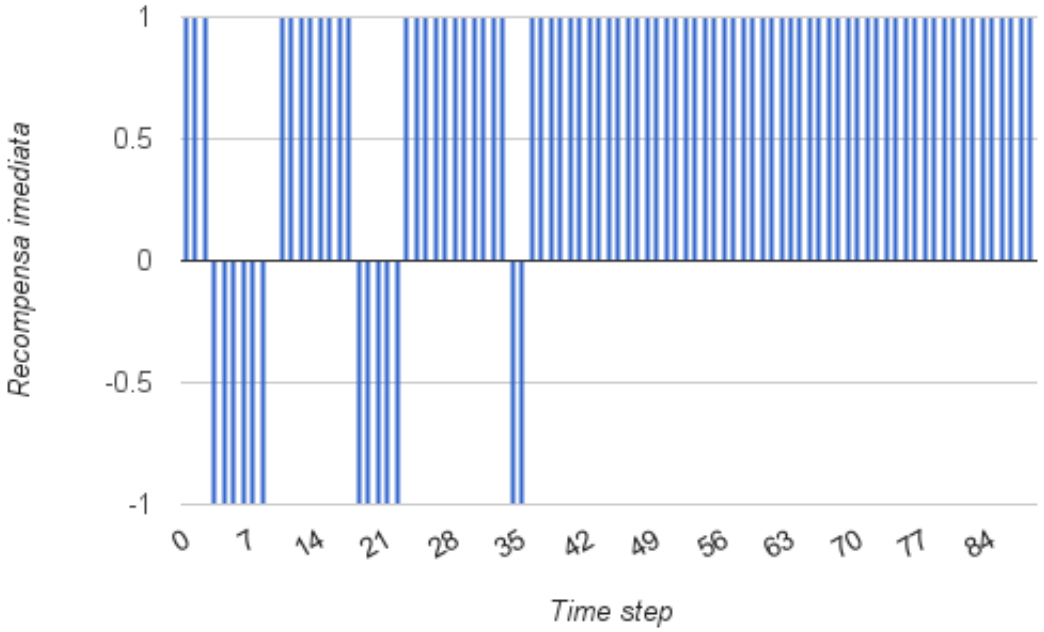
Também foi estipulado que após 40 *time steps* a taxa de exploração cairia para $\epsilon = 0.1$. No teste realizado, os clientes Hc_1 e Hc_3 estabelecem conexão com o servidor Hs_1, e o cliente Hc_2 faz o mesmo com o servidor Hs_2. A rota adotada pelo controlador em um primeiro momento para todos os hosts é o caminho mínimo. Neste cenário, portanto, inicialmente a troca de pacotes ocorre por meio do link que conecta os switches *s1* e *s2*. Em todos os casos, o arquivo sendo trocado em cada conexão possui 150 Mb.

O cliente Hc_3 é um host malicioso que executa diversas requisições TCP a fim de exaurir o servidor Hs_1. Rapidamente os links ficam sobrecarregados em função dos muitos arquivos sendo transmitidos pelos poucos links existentes, e o agente não encontra nenhuma rota alternativa que lhe confira uma recompensa positiva. O ataque é posteriormente detectado, e o agente ordena que as VNFs de *firewall* barrem todo o fluxo envolvendo o cliente Hc_3. A Figura 4.4 mostra as recompensas imediatas obtidas pelo agente para o perfil de balanceamento de carga. Note que durante a identificação e eliminação do ataque, no *time step* 9, não houve recompensa porque o perfil de rede ativo era o de mitigação, que possui prioridade mais alta em relação ao de balanceamento.

A Figura 4.5 mostra a quantidade de requisições feita pelo cliente malicioso Hc_3 (em vermelho) e o número de requisições que de fato chegou ao servidor Hs_1 (em azul). Até o *time step* 9 havia paridade entre requisições do cliente e atendimentos por parte do servidor. Contudo, nota-se a atuação dos *firewalls*, que neste instante bloquearam o fluxo do atacante. Assim, a vítima não mais recebeu as requisições, apesar das tentativas persistentes do atacante.

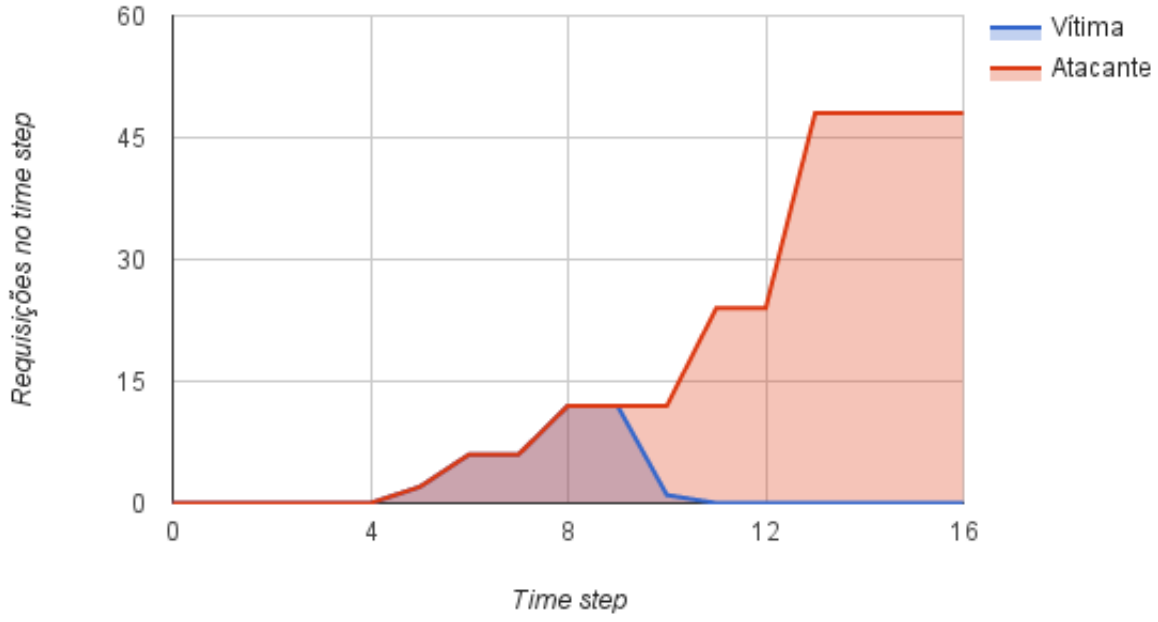
[5]https://wiki.linuxfoundation.org/networking/bridge
[6]http://ebtables.netfilter.org

Figura 4.4: Recompensas imediatas obtidas.



Fonte: O autor.

Figura 4.5: Requisições enviadas e recebidas.

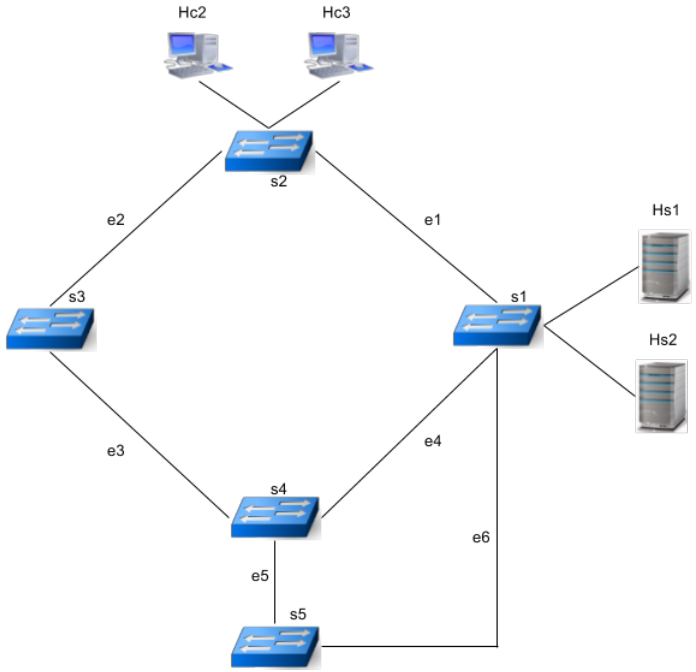


Fonte: O autor.

4.2.1.2 Balanceamento em topologia maior

Para este caso de teste foi gerada uma topologia bastante semelhante à usada no passo a passo da Seção 3.3.1, mas com diferença na colocação dos hosts, conforme mostra a Figura 4.6. Os hosts Hs_1 e Hs_2 estão conectados ao switch *s1* e agem como servidores, ao passo que os hosts Hc_1 e Hc_2 estão conectados no switch *s2* e são configurados como clientes.

Figura 4.6: Topologia de teste.

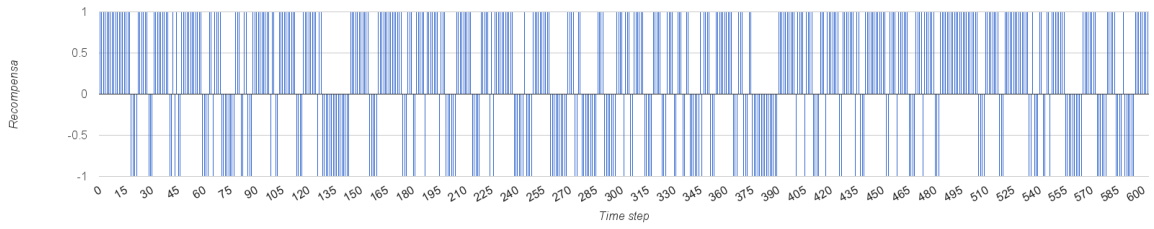


Fonte: O autor.

Um arquivo é transmitido entre os hosts Hs_1 e Hc_1 e outro entre os hosts Hs_2 e Hc_2 ao mesmo tempo. O tráfego total gerado é de 1Gb. O caminho inicial adotado pelo controlador para o encaminhamento dos pacotes é o caminho mínimo.

Foi fixada uma taxa de aprendizagem $\alpha = 0.1$ e taxa de exploração $\epsilon = 0.4$. Após a conclusão do envio do arquivo por parte dos dois fluxos, as recompensas imediatas obtidas em cada *time step* foram as seguintes, conforme mostra a Figura 4.7.

Figura 4.7: Recompensas imediatas obtidas ao longo do tempo

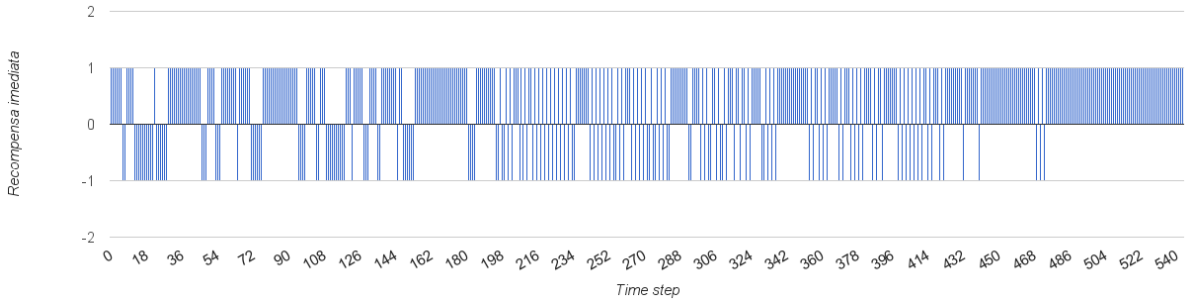


Fonte: O autor.

Observa-se que o agente explora diferentes ações, recebendo recompensas positivas em 375 oportunidades e negativas em outras 228. Isso é esperado, uma vez que a topologia é simples, há poucos hosts presentes. Em 60% ele adota um comportamento conservador, adotando uma ação já conhecida para o visitado (se houver), e em 40% dos casos explora uma ação desconhecida, que pode vir se provar boa ou ruim.

Em seguida foi feito um novo teste, desta vez com os mesmos servidores, mas com dois clientes conectados ao switch *s5* e outro conectado a *s3*. Os dois primeiros clientes estabeleciam conexão com um servidor cada. Após 100 *time steps*, o terceiro cliente fazia o mesmo com um dos servidores. A Figura 4.8 mostra as recompensas imediatas obtidas ao longo do tempo pelo agente, durante o balanceamento de carga. Após 400 *time steps*, a taxa de exploração foi reduzida para $\epsilon = 0.1$.

Figura 4.8: Recompensas imediatas obtidas ao longo do tempo



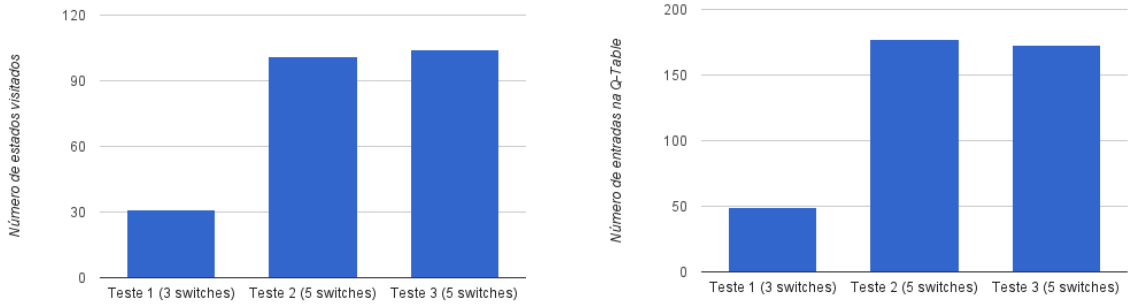
Fonte: O autor.

Nota-se, ao minimizar a taxa de exploração e adotar preponderantemente a melhor política encontrada, que o agente de fato conseguiu encontrar um caminho que evite gargalos.

Contudo, registra-se o baixo desempenho espacial da técnica tradicional em se armazenar estados em tabelas. O primeiro teste apresentava uma topologia com três switches, e os outros dois testes acrescentavam dois, totalizando cinco switches. Apesar do pequeno aumento da topologia, o número de combinações possível para encaminhar pacotes cresceu de tal forma que o agente percorreu muito mais estados, conforme mostra a Figura 4.9 (a). Isso se refletiu ainda no tamanho da tabela estado-ação, que também registrou aumento expressivo no número de entradas adicionadas, conforme mostra a Figura 4.9 (b).

Quanto mais métricas forem coletadas para a formação de um estado, com mais intensidade esse fenômeno tende a ocorrer. No caso de perfil de balanceamento de carga, por exemplo, ainda que os valores que compõem os estados sejam discretizados em faixas do tipo "alto", "médio" e "baixo", cada estado é representado por uma matriz, sendo que cada linha da matriz possui dois campos (caminho e status do caminho). Quanto maior for o número de switches, mais haverão caminhos possíveis entre os hosts, levando a uma grande combinação

Figura 4.9: Crescimento da tabela estado-ação e estados visitados



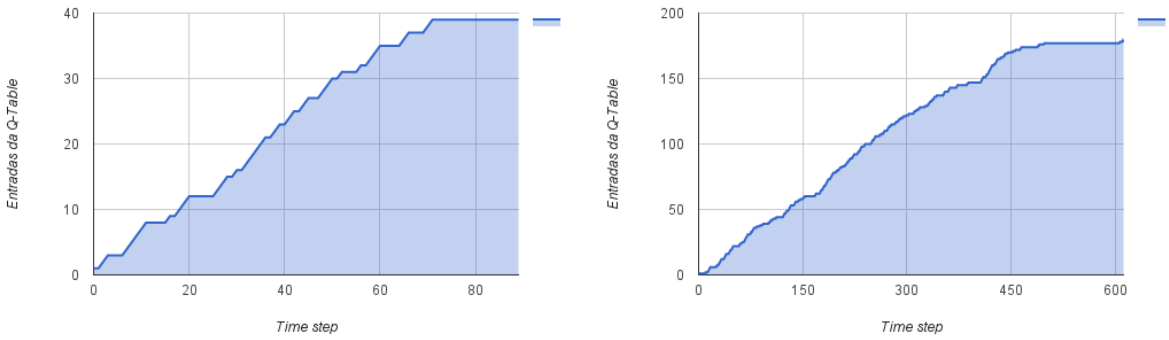
(a) Estados visitados

(b) Tamanho da tabela estado-ação

de estados possíveis. Em uma rede grande, pode ser custoso tanto o armazenamento dessas informações como o tempo necessário para o treinamento do agente.

As figuras 4.10 (a) e 4.10 (b) mostram o quanto o tamanho da Q-Table varia conforme o progresso dos *time steps* para os testes de mitigação e balanceamento com redução na taxa de exploração.

Figura 4.10: Crescimento da tabela estado-ação e estados visitados



(a) Estados visitados

(b) Tamanho da tabela estado-ação

Nota-se uma curva de crescimento acelerado, o que evidencia a necessidade de se pensar alternativas para o armazenamento de dados em tabelas a fim de evitar, ou pelo menos atenuar, esse fenômeno.

5 TRABALHOS RELACIONADOS

Diversos pesquisadores têm se empenhado em tornar redes de computadores mais seguras. Técnicas de inteligência artificial têm sido adotadas para tornar redes mais resilientes, sejam elas definidas por software (e consequentemente aproveitando a flexibilidade de um controlador), ou não. O mesmo é válido para trabalhos que exploram e mesclam características de NFV e SDN.

Este capítulo é organizado da seguinte forma: primeiro, são apresentados trabalhos que estudam detectar e mitigar ataques em SDN. Em seguida, são discutidos trabalhos que adotam aprendizagem por reforço para lidar com anomalias em redes, sejam elas definidas por software ou não. O universo de trabalhos que exploram detecção e mitigação de anomalias em SDN é extenso, e em vez que apresentar uma lista exaustiva, este capítulo pincela um subconjunto com maior afinidade com as ideias propostas neste documento. Ao final, é feita uma discussão.

5.1 Anomalias em SDN

Machado, Granville e Schaeffer-Filho (2016) propõem uma arquitetura que combina SDN e NFV para tornar redes definidas por software mais resilientes. VNFs e controlador monitoram e mitigam ataques com base em um *control-loop* que identifica a melhor estratégia para resolver um tipo específico de anomalia na rede. O *control-loop*, que exerce a função de "cérebro" do sistema, não adota aprendizagem de máquina. Esta possibilidade é, contudo, citada para um trabalho futuro a fim de melhorar o *feedack* do *control-loop*.

Oktian, Lee e Lee (2014) propõem mitigar ataques ao monitorar todos os nós da rede. Sua aplicação reside no controlador e possui seis características: (*i*) manutenção da relação de endereços MAC e IP de hosts e switches, MAC e portas, portas ativas nos switches e uma lista de MACs conectados; (*ii*), manutenção da informação de quais hosts estão conectados a quais switches; (*iii*), quando o controlador adiciona uma regra, compara o par MAC/IP com a lista de MAC/IP citada anteriormente para identificar pacotes falsos; (*iv*), o controlador busca por estatísticas nos switches como o tamanho do tráfego: uma grande alteração é considerada uma porta de entrada para ataques; (*v*), periodicamente busca por entradas maliciosas em tabelas de fluxo para deletá-las e (*vi*), monitora status das portas de cada switch. Sua aplicação ajuda a mitigar sobretudo ataques de endereços IPs e MACs forjados, mas não utiliza NFV, tecnologia que poderia pelo menos aliviar o aumento da latência do controlador (que sobe 21% segundo os pesquisadores), por meio de, por exemplo, a instanciação de uma VNF de *firewall*, como

propõe nosso trabalho.

Belyaev e Gaivoronski (2014) utilizam balanceamento de carga para lidar com ataques DDoS. Em sua abordagem, fluxos maliciosos não são descartados; em vez disso são redirecionados na rede. Seus experimentos mostraram que a solução ajuda a aumentar o tempo de sobrevivência a um ataque, ao tentar absorvê-lo, aproveitando a visão global que o controlador SDN tem para realizar a distribuição do tráfego. Nosso trabalho, por outro lado, optou por uma abordagem diferente, de separar a rede em perfis, e só balancear o tráfego de fluxos legítimos, eliminando aqueles considerados maliciosos primeiro com o auxílio da tecnologia NFV.

Mousavi e St-Hilaire (2015) propõem um sistema para detectar ataques DDoS rapidamente. Sua ideia é usar entropia para calcular a aleatoriedade dos pacotes na rede. Em uma situação de ataque, a entropia seria baixa, pois o tráfego seria pouco aleatório e mais direcionado a uma vítima. O controlador SDN é a figura responsável por coletar as estatísticas necessárias para determinar se um ataque está em curso. A janela de monitoramento selecionada pelos pesquisadores tem um tamanho de 50 pacotes, pois alegam ser este o menor tamanho que efetivamente detecta ataques. Em vez de usar entropia, nosso trabalho optou por uma abordagem mais simples, que adota limiares, mas uma alternativa mais refinada, que use entropia, não é descartada, conforme é abordado no Capítulo 6, na discussão de trabalhos futuros.

5.2 Aprendizagem por reforço em redes

Malialis (2014) propõe uma abordagem para lidar com ataques de negação de serviço. Seu trabalho emprega aprendizagem por reforço instalando múltiplos agentes em roteadores. Sua arquitetura é descentralizada para ser mais resiliente a ataques e os agentes trabalham em conjunto descartando pacotes para manter um servidor operacional. Os agentes são dispostos em pontos fixos na topologia seguindo a proposta de Yau (2005). Sua tese não reside no universo de SDN nem NFV, mas não seria infactível, em um trabalho futuro, instalar os agentes em VNFs, em vez de roteadores. Isso conferiria uma flexibilidade maior para instanciá-los sob demanda, algo que nosso agente de RL procura aprender ao interagir com o ambiente.

Hu e Chen (2013) adotaram aprendizagem por reforço supervisionada para balancear o tráfego de uma rede. A premissa é que mesmo que se adote como política de encaminhamento o menor caminho entre dois hosts, se muitos pacotes trafegarem pela mesma rota, esta pode ficar congestionada. Dessa forma, outro caminho alternativo, embora mais longo, poderia ser uma alternativa mais adequada. Eles usaram o algoritmo Q-Learning para encaminhar pacotes por caminhos diferentes da rede, com a figura de um supervisor presente para conferir recompensas

extras ao agente e otimizar sua aprendizagem, além daquelas provenientes do ambiente, em um cenário de aprendizagem por reforço supervisionado, portanto. Para o nosso trabalho, buscou-se adotar aprendizagem de máquina para balancear o tráfego de maneira *online*, sem a necessidade de um supervisor.

Em um trabalho semelhante, Kim et al. (2016) também mesclam SDN e aprendizagem por reforço para tratar congestionamento de fluxo. Os pesquisadores então definiram um limiar em cada link. Quando o tráfego o supera, o controlador procura por uma nova rota a partir do algoritmo Q-Learning. O controlador SDN é figura central no reordenamento de rotas. O foco do balanceamento de carga é diminuir o estresse somente sobre os links, sem levar em consideração o host que eventualmente seja o destino de todo o tráfego. Nosso trabalho buscou resolver essa questão também ao propor um agente que não somente distribui o tráfego, mas também se necessário instancia réplicas de servidores ao realizar o balanceamento.

Outra proposta de balancear tráfego usando aprendizagem por reforço é feita por Desai e Patil (2014). Switches recebem informações de seus vizinhos a respeito do *delay* ocorrido ao transmitir pacotes. Com base nisso, caminhos alternativos são adotados em momentos de gargalo em uma rota. Entretanto, seu trabalho não reside no universo de SDN, e portanto a lógica de encaminhamento é descentralizada, pois fica diluída entre os switches, que precisam comunicar-se entre si. Nossa proposta se vale da visão global da rede que possui o controlador para calcular os caminhos possíveis entre os hosts bem como efetuar as medições necessárias nos links.

5.3 Considerações

Os diferentes trabalhos brevemente apresentados acima procuram explorar técnicas para promover resiliência em redes focando somente em uma anomalia por vez. Nosso trabalho é pioneiro, ou um dos pioneiros, na área de SDN, a procurar mesclar diferentes técnicas de resiliência concorrentemente, usando aprendizagem por reforço e a tecnologia NFV.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi proposto um sistema para detectar e mitigar anomalias em redes definidas por software com auxílio da tecnologia NFV. Após uma introdução, foram revisados conceitos de SDN, NFV e aprendizagem por reforço. A modelagem do sistema foi apresentada na sequência, seguida do protótipo desenvolvido e análise dos resultados. Finalmente, foi discutido como o presente trabalho se diferencia de outros trabalhos publicados que são relacionados.

Em específico, propôs-se a adoção de aprendizagem por reforço para coordenar os trabalhos de um agente que reside no orquestrador da arquitetura NFV e coleta evidências de anomalias a partir de métricas de rede. As métricas de rede, além de fornecerem a base para a evidência de anomalias, são utilizadas para a construção de perfis de rede. Os perfis de rede procuram hierarquizar o tratamento a diferentes ameaças, fazendo com que o agente se concentre em eliminar primeiro determinadas anomalias mais urgentes. Esta hierarquização também serve para evitar que uma ação de um perfil destinada a eliminar uma ameaça acabe anulando os efeitos de uma outra ação de outro perfil.

Foi verificado que, tipicamente, estudos publicados na literatura procuram tratar, ainda que com profundidade, soluções para apenas um tipo específico de anomalia de rede. Este trabalho tem, como uma das principais contribuições, portanto, procurar combinar diferentes técnicas de mitigação para diferentes anomalias harmonicamente, e para os testes realizados, ainda que em ambientes virtualizados, houve resultados promissores.

Por outro lado, também por meio da análise de resultados foi possível constatar que a representação dos estados por meio de tabela apresenta um elevado custo de armazenamento: um pequeno incremento no número de switches provoca um grande aumento na quantidade de estados que podem ser visitados.

Futuramente, vislumbra-se, a fim de se tentar obter melhores resultados, uma série de aprimoramentos no sistema aqui apresentado.

- Ampliar o universo de perfis modelados de forma a abranger outras funções de rede. Por exemplo, incluir monitoramento de escaneamento de portas (*port scan*), sistema detector de intrusos (IDS), antivírus, entre outros.
- Refinar perfis propostos. Por exemplo, adotar técnicas mais refinadas para detectar e mitigar ataques de negação de serviço. No modelo proposto, um número pequeno de requisições é o suficiente para ser considerado um ataque, caso o aumento seja brusco. Uma alternativa seria a adoção de entropia (SILVA et al., 2016). Poderia-se também

monitorar a relação tráfego / horas do dia, de forma a guardar registro do comportamento sazonal da rede a fim de distinguir com maior precisão fluxos maliciosos de legítimos.

- Adotar outras técnicas para representação dos estados dos perfis de rede. Em vez de usar a abordagem de discretização de valores e posterior armazenamento em tabelas, poderia-se adotar redes neurais, redes bayesianas (ASHRAF; LATIF, 2014) ou Tile-Coding (SUTTON; BARTO, 2012).
- Em uma abordagem mais refinada para a representação de estados, poderia-se coletar outras métricas para a representação dos mesmos. Por exemplo, o perfil de réplica de servidor poderia levar em consideração não somente a vazão no link de acesso do servidor à rede, mas também seu consumo de CPU, uso de memória, percentual de disponibilidade ao longo do tempo, entre outras.

REFERÊNCIAS

AHMAD, I. et al. Security in software defined networks: A survey. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2317–2346, Fourthquarter 2015. ISSN 1553-877X.

ASHRAF, J.; LATIF, S. Handling intrusion and DDoS attacks in Software Defined Networks using machine learning techniques. In: **Software Engineering Conference (NSEC), 2014 National**. [S.l.: s.n.], 2014. p. 55–60.

BELYAEV, M.; GAIVORONSKI, S. Towards load balancing in sdn-networks during ddos-attacks. In: **Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International**. [S.l.: s.n.], 2014. p. 1–6.

DESAI, R.; PATIL, B. P. Reinforcement learning for adaptive network routing. In: **Computing for Sustainable Global Development (INDIACom), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 815–818.

ETSI. **Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action**. 2012. Available from Internet: <https://portal.etsi.org/nfv/nfv_white_paper.pdf>.

ETSI. **Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress**. 2015. Available from Internet: <https://portal.etsi.org/nfv/nfv_white_paper3.pdf>.

FELTER, W. et al. An updated performance comparison of virtual machines and Linux containers. In: **Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on**. [S.l.: s.n.], 2015. p. 171–172.

FONSECA, P. et al. A replication component for resilient openflow-based networking. In: **2012 IEEE Network Operations and Management Symposium**. [S.l.: s.n.], 2012. p. 933–939. ISSN 1542-1201.

FONSECA, P. et al. Resilience of SDNs based On active and passive replication mechanisms. In: **2013 IEEE Global Communications Conference (GLOBECOM)**. [S.l.: s.n.], 2013. p. 2188–2193. ISSN 1930-529X.

HERRERA, J. G.; BOTERO, J. F. Resource Allocation in NFV: A Comprehensive Survey. **IEEE Transactions on Network and Service Management**, PP, n. 99, p. 1–1, 2016. ISSN 1932-4537.

HU, Z.; CHEN, H. Network load balancing strategy based on supervised reinforcement learning with shaping rewards. In: **Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on**. [S.l.: s.n.], 2013. p. 393–397.

JMAL, R.; FOURATI, L. C. Implementing shortest path routing mechanism using openflow pox controller. In: **Networks, Computers and Communications, The 2014 International Symposium on**. [S.l.: s.n.], 2014. p. 1–6.

JONES, A. K.; SIELKEN, R. S. **Computer System Intrusion Detection: A Survey**. "http://www.cs.virginia.edu/ jones/IDS-research/Documents/jones-sielken-survey-v11.pdf", 2000.

KANDOI, R.; ANTIKAINEN, M. Denial-of-service attacks in openflow sdn networks. In: **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2015. p. 1322–1326. ISSN 1573-0077.

KIM, S. et al. Congestion prevention mechanism based on Q-learning for efficient routing in SDN. In: **2016 International Conference on Information Networking (ICOIN)**. [S.l.: s.n.], 2016. p. 124–128.

KING, D.; FARREL, A.; GEORGALAS, N. The role of SDN and NFV for flexible optical networks: Current status, challenges and opportunities. In: **2015 17th International Conference on Transparent Optical Networks (ICTON)**. [S.l.: s.n.], 2015. p. 1–6. ISSN 2162-7339.

MACHADO, C. C.; GRANVILLE, L. Z.; SCHAEFFER-FILHO, A. ANSwer: Combining NFV and SDN features for network resilience strategies. In: **2016 IEEE Symposium on Computers and Communication (ISCC)**. [S.l.: s.n.], 2016. p. 391–396.

MALIALIS, K. **Distributed Reinforcement Learning for Network Intrusion Response**. Thesis (PhD), United Kingdom, 9 2014.

MATIAS, J. et al. Toward an SDN-enabled NFV architecture. **IEEE Communications Magazine**, v. 53, n. 4, p. 187–193, April 2015. ISSN 0163-6804.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1355734.1355746>.

MITCHELL, T. M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

MOUSAVI, S. M.; ST-HILAIRE, M. Early detection of ddos attacks against sdn controllers. In: **Computing, Networking and Communications (ICNC), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 77–81.

NUNES, M. M. B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X.

OKTIAN, Y. E.; LEE, S.; LEE, H. Mitigating denial of service (dos) attacks in openflow networks. In: **2014 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.: s.n.], 2014. p. 325–330. ISSN 2162-1233.

ONF. **Software-Defined Networking: The New Norm for Networks**. 2012. Available from Internet: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

ONF. **SDN Architecture Overview**. 2014. Available from Internet: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf>.

RAHO, M. et al. KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing. In: **Information, Electronic and Electrical Engineering (AIEEE), 2015 IEEE 3rd Workshop on Advances in**. [S.l.: s.n.], 2015. p. 1–8.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2. ed. New Jersey, EUA: Prentice Hall, 2003.

SEZER, S. et al. Are we ready for SDN? Implementation challenges for software-defined networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804.

SILVA, A. S. da et al. ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 27–35.

STERBENZ, J. P. et al. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. **Computer Networks**, v. 54, n. 8, p. 1245 – 1265, 2010. ISSN 1389-1286. Resilient and Survivable networks. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1389128610000824>.

SUTTON, R. S.; BARTO, A. G. **Introduction to Reinforcement Learning**. 2. ed. Massachusetts, EUA: MIT Press, 2012.

SZEPESVáRI, C. **Algorithms for Reinforcement Learning**. Edmonton, Alberta, Canada: Morgan & Claypool, 2013.

YAU, D. K. Y. et al. Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles. **IEEE/ACM Trans. Netw.**, IEEE Press, Piscataway, NJ, USA, v. 13, n. 1, p. 29–42, feb. 2005. ISSN 1063-6692. Available from Internet: <http://dx.doi.org/10.1109/TNET.2004.842221>.