Emprego de NFV e Aprendizagem por Reforço para Detectar e Mitigar Anomalias em Redes Definidas por Software

Pedro H. A. Faustini[1], **Anderson S. Silva**[1], **Lisandro Z. Granville**[1], **Alberto E. Schaeffer-Filho**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{phafaustini,assilva,granville,alberto}@inf.ufrgs.br

Abstract. *A computer network is subject to several anomalies, and it is necessary to coordinate detection and mitigation techniques to keep the network operational. This paper proposes the use of reinforcement learning to promote resilience in software-defined networking (SDN). In particular, it is proposed collecting network metrics and their grouping into profiles, each one having a set of actions to handle problems using reinforcement learning, network functions virtualisation (NFV), and an SDN controller. Policies for dealing with anomalies are defined, based on the rewards for each action. Results show that the system obtains mostly positive rewards, but a small increment in the topology size leads to more than four times the number of entries in the state-action table.*

Resumo. *Uma rede de computadores está sujeita a diversas anomalias, e é necessário coordenar técnicas de detecção e mitigação para mantê-la operacional. Este artigo propõe aprendizagem por reforço para promover resiliência em redes definidas por software (SDN). Propõe-se que métricas de rede sejam coletadas e agrupadas em perfis, cada um com um conjunto de ações que trate problemas usando aprendizagem por reforço, virtualização de funções de rede (NFV) e um controlador SDN. Políticas para lidar com anomalias são definidas com base nas recompensas das ações. Os resultados mostram que o sistema obtém majoritariamente recompensas positivas, mas um pequeno aumento na topologia mais do que quadriplica o número de entradas na tabela estado-ação.*

1. Introdução

Redes de computadores podem apresentar diversas anomalias. Entende-se por anomalias não somente a presença de fluxos com conteúdo malicioso, como no caso de ataques de negação de serviço, mas também fluxos benignos que, combinados, podem deteriorar a experiência dos usuários. Surge portanto a necessidade de manter a resiliência da rede, isto é, a habilidade da rede em manter um nível aceitável de operação frente a várias falhas e desafios à sua operação normal [Sterbenz et al. 2010]. Diferentes anomalias requerem diferentes técnicas de mitigação, o que leva ao problema de como coordenar essas técnicas harmonicamente, ou seja, como acionar uma técnica sem que sua ação prejudique ações de outras técnicas usadas na promoção da resiliência.

Este artigo tem como objetivo investigar como redes definidas por software (*Software-Defined Networking* - SDN) podem se beneficiar de técnicas baseadas em

aprendizagem de máquina e virtualização de funções de rede (*Network Functions Virtualisation* - NFV) de modo a selecionar estratégias de mitigação para diferentes tipos de anomalias. Por exemplo, a sobrecarga de um servidor devido ao alto número, ainda que legítimo, de requisições, ou o gargalo em um enlace que transmite muitos pacotes de diferentes máquinas, são considerados anomalias e devem ser tratados.

Nesse contexto, as principais contribuições deste artigo são: **(i)** levantamento do estado-da-arte sobre o uso de aprendizagem por reforço em redes; **(ii)** modelagem de um agente que adota técnicas de aprendizagem por reforço e aprende a lidar com diferentes tipos de anomalias e **(iii)** implementação de um protótipo e avaliação de aspectos como crescimento das tabelas e desempenho na obtenção das recompensas.

Acreditamos que a combinação dos paradigmas de SDN e NFV tem o potencial de facilitar a construção de técnicas mais flexíveis para a promoção de resiliência. Em específico, SDN fornece um ambiente propício para manipular regras de roteamento de pacotes, além de facilitar a coleta de métricas em função da presença do controlador, que possui visão global da rede [McKeown et al. 2008]. NFV, por sua vez, fornece flexibilidade na instanciação de mecanismos a serem usados na detecção e mitigação de ameaças.

O restante deste artigo segue a seguinte estrutura: a Seção 2 apresenta o referencial teórico para os conceitos de SDN, NFV e aprendizagem por reforço. Na Seção 3 é proposto o modelo de trabalho conjunto de NFV e aprendizagem por reforço para promover a resiliência em SDN. Na Seção 4, são apresentados o protótipo implementado e resultados. A Seção 5 contém os trabalhos relacionados, e a Seção 6 apresenta as conclusões e prospecções de trabalhos futuros.

2. Fundamentação

Nesta seção serão apresentados importantes conceitos que formam a fundamentação deste trabalho. Primeiro será visto SDN, seguido de NFV e, por fim, aprendizagem por reforço.

2.1. Redes Definidas por Software

O conceito de SDN 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, desacoplando o plano de controle do plano de dados [ONF 2014]. O plano de dados é responsável por trafegar pacotes, e é mais dependente de componentes de hardware, como switches. O plano de controle possui a capacidade de executar funções via software a fim de flexibilizar a inserção e remoção de funcionalidades de controle. Um elemento fundamental no plano de controle é o controlador, que 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.

O protocolo OpenFlow é o padrão *de facto* para comunicação entre plano de controle e plano de dados [B. A. A. Nunes et al. 2014]. O protocolo serve como um canal entre o plano de dados e o plano de controle, tornando possível adicionar ou remover entradas nas tabelas de fluxos dos switches. Além disso, redes programáveis podem diminuir barreiras para a concepção de novas ideias, facilitando a inovação em redes [McKeown et al. 2008].

2.2. Virtualização de Funções de Rede

Redes de computadores costumam possuir muitas funções em sua infraestrutura. Exemplos de funções de rede incluem *firewalls*, sistema de detecção de intrusões, 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 e Botero 2016]. Outro problema é a falta de flexibilidade no gerenciamento desses dispositivos. NFV busca portar tais funções para hardware de propósito geral, executando-as 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 o desempenho em ambientes virtualizados, resiliência frente a falhas tanto de hardware como de software, entre outros [ETSI 2012].

Há três componentes principais na arquitetura NFV: infraestrutura, serviço e orquestração & gerenciamento [Herrera e Botero 2016]. A Infraestrutura, conhecida como NFVI (*NFV Infrastructure*), abrange os recursos de hardware e software, bem como o ambiente de virtualização. O componente de Serviço abrange as funções de rede virtualizadas, ou VNFs (*Virtualised Network Functions*). Elas são executadas em ambientes virtualizados, em vez de hardware dedicado (*middleboxes*) [King et al. 2015]. Orquestração e gerenciamento, conhecida como NFV-Mano (*management and orchestration*), foca em tarefas mais específicas da virtualização. Por exemplo, se entre as VNFs estão *firewall* e um detector de intrusões (IDS), NFV-Mano pode determinar que um pacote siga, após o resultado da inspeção do detector, para um *firewall*. Apesar dos conceitos de NFV e SDN serem independentes, as tecnologias podem ser complementares. É possível que, no futuro, ambas tecnologias se tornem menos distinguíveis até se unirem como um único paradigma em redes baseadas em software [ETSI 2014].

2.3. 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* o agente muda de estado e recebe uma recompensa [Sutton e Barto 2012]. Uma maneira de definir problemas de aprendizagem por reforço é através dos processos de decisão de Markov (*Markov decision process*, ou MDP). MDP fornece um *framework* matemático muito usado na modelagem das dinâmicas de um ambiente sob diferentes ações, e pode ser definido por uma 4-upla [Malialis 2014] $\langle S, A, R, P \rangle$, em que ***S*** é o conjunto de estados, ***A*** é o conjunto de ações, ***R*** é função de recompensa (retorna a recompensa no estado s_{t+1}, quando a ação a é executada no estado s), e ***P*** é a 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.

Na maioria dos problemas, a dinâmica do ambiente não está disponível [Malialis 2014], e os problemas de aprendizagem por reforço costumam envolver a estimação de funções valor. Uma tabela estado-ação, ou Q-Table $Q(s, a)$, procura definir o quão bom é executar uma ação a em um estado s. Esta tabela é atualizada a partir da recompensa obtida pelo agente. Entre as formas de atualização da recompensa está o algoritmo Sarsa [Sutton e Barto 2012]. 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. O algoritmo, em pseudo-código, é apresentado a seguir:

Algoritmo 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*)
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r - Q(s,a)]$
        $S \leftarrow S', A \leftarrow A'$

A recompensa é denotada por *r* e α é a taxa de aprendizagem. Quanto maior o seu valor, maior será a variação dos valores atualizados na tabela a partir da recompensa. Enquanto aprende, o agente se depara com um dilema entre adotar uma ação cuja recompensa é conhecida ou explorar novas possibilidades e adquirir mais conhecimento sobre as consequências de suas ações. Uma técnica popular e efetiva que lida com este dilema é ϵ-*greedy* [Sutton e 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.

3. Aprendizagem por Reforço em Infraestruturas Baseadas em SDN/NFV

Neste artigo, propomos um modelo de aprendizagem por reforço integrado à arquitetura de NFV por meio de um agente que reside no orquestrador. O agente recebe métricas de rede e constrói perfis de rede. Cada perfil busca tratar uma anomalia. Cada perfil é formado por um conjunto de ações, bem como uma tabela de estados e uma tabela estado-ação. A cada ciclo de aprendizagem (um intervalo de tempo, ou *time step*, arbitrário), o agente seleciona um perfil, executa uma ação disponível e recebe uma recompensa para atualizar a tabela estado-ação (Q-Table) daquele perfil, conforme exibe a Figura 1.

A cada ciclo, o agente executa somente uma ação de um perfil. 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 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 adotada apresente alto tráfego por causa 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 (p.ex. em uma CDN). Entretanto, replicar um conteúdo implica em desviar a rota de certos hosts

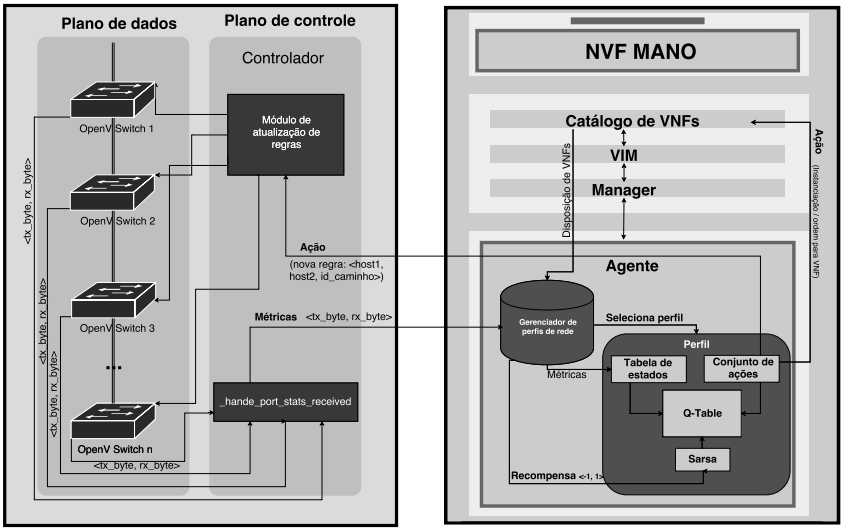


Figura 1. Arquitetura para aprendizagem por reforço em infraestruturas baseadas em SDN/NFV

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. Em específico, neste trabalho são estudados três perfis: **(i)** balanceamento de tráfego, **(ii)** replicação de servidores e **(iii)** ataques de negação de serviço.

3.1. Balanceamento de Tráfego

Dispositivos de encaminhamento, ou switches, possuem portas por onde links responsáveis por transmitir pacotes se conectam. Se os links incidentes a um switch apresentam alta vazão, a fila de espera (*buffer*) do switch pode encher, ocasionando o descarte de pacotes e prejudicando a experiência de usuários. 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. As ações deste perfil consistem em **(i)** ação vazia, **(ii)** aplicar o caminho mínimo entre dois hosts, e **(iii)** alterar um caminho aplicado entre dois hosts.

Um caminho é dito congestionado (tem status alto) se algum de seus links apresenta tráfego igual ou superior a um determinado percentual da capacidade máxima, por exemplo 80%. Senão, o status é considerado médio se algum de seus links apresenta tráfego em uma faixa inferior, por exemplo, a partir de 30% da capacidade máxima. Senão, é considerado baixo. Um link pode pertencer a vários caminhos. Se o tráfego gerado por cada fluxo for pequeno, mas somados deixarem o link sobrecarregado, ambos caminhos apresentarão status alto.

O agente, 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*.

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

Os campos *tx_byte* e *rx_byte* das tabelas de fluxo são cumulativos, isto é, não possuem a informação de quantos bytes trafegaram por uma porta em um determinado tempo. Contudo, tal restrição é facilmente contornada calculando a diferença entre os valores encontrados nos ciclos atual e anterior. Em cada *time step*, o agente tem a sua disposição o estado atual e o conjunto de caminhos possíveis calculado pelo controlador para cada par de hosts. De posse dessas informações, ele pode solicitar que o controlador modifique regras nas tabelas de fluxo dos switches de forma a trocar um caminho por outro.

O agente somente poderá alterar um caminho aplicado entre dois hosts caso um ou mais links em um caminho esteja no status alto. Existem dois objetivos: *(i)* evitar que um link esteja com status alto da sua 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 status alto, ou -1 caso contrário. Ou, formalmente:

$$R = \begin{cases} -1, & \text{se há link com tráfico em status alto} \\ 1, & \text{caso contrário} \end{cases} \tag{1}$$

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 retornará 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. Replicação de Servidores

Servidores fornecem serviços para clientes. 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 link de acesso ao servidor. A construção do perfil de carga sobre servidores ocorre por meio de três métricas: **(i)** o tráfego no link de acesso entre servidor e switch, **(ii)** a relação de servidores ou VNFs de réplicas, e **(iii)** os caminhos incidentes ao servidor ou réplicas.

Como o tráfego em um link possui muitos valores possíveis, ele é dividido em três faixas: *baixo* (por exemplo, $[0\%, 30\%)$ de uso), *médio* (por exemplo, $[30\%, 80\%)$) e *alto* (por exemplo, $[80\%, 100]$). 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. 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 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-calcular todas as combinações possíveis, e sim utilizar estruturas de dados que dinamicamente acrescentem os novos estados à medida em que eles forem verificados pela primeira vez, como listas. Para o perfil de carga nos servidores,

as ações possíveis consistem em: **(i)** instanciar uma réplica de servidor, **(ii)** remover uma réplica instanciada, e **(iii)** 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 uma necessidade maior de instanciações do que em uma situação de média vazão, por exemplo. 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. O agente tem dois objetivos: *(i)* manter o link de acesso do servidor ao switch em um estado de baixo tráfego, *(ii)* fazer isso 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:

$$R = \begin{cases} 1, & \text{se tráfego em link de acesso for baixo} \\ -1, & \text{caso contrário} \end{cases} \tag{2}$$

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 de forma a mantê-los com status de baixa vazão. É possível que o agente execute uma ação que leve um link 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 com uma recompensa negativa. No longo prazo, aprenderá a como reagir frente a cada configuração de tráfego.

3.3. Mitigação de Ataques

Ao contrário de balanceamento de carga ou replicação 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 execute ações dos demais perfis somente para os pacotes legítimos. Este perfil não adota aprendizagem de máquina em sua solução, e assim não há uma tabela estado-ação. Este perfil possui somente dois estados, portanto: **(i)** malicioso e **(ii)** benigno.

Quando este perfil se encontra no estado malicioso, a única ação possível é mitigar o ataque. Enquanto um ataque não for detectado, o agente pode executar ações de perfis de prioridade mais baixa. O controlador mantém registro de média e desvio padrão de requisições que cada servidor recebe. 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 de SDN 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. 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. O próximo passo da aplicação é mitigar o ataque.

Para a mitigação, foi adotada uma heurística simples, para fins de prova de conceito: 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 mais três desvios são barrados. Para isso, 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. O agente então instancia e ordena que VNFs que implementam *firewalls* barrem pacotes provenientes dos endereços IPs considerados suspeitos.

4. Protótipo e Resultados

Esta seção descreve o protótipo implementado, o qual está disponível publicamente[3]. Em específico, na Subseção 4.1 são vistos detalhes do protótipo desenvolvido, e na Subseção 4.2 são apresentados experimentos realizados e resultados obtidos.

4.1. Implementação do Protótipo

O protótipo desenvolvido abrange os perfis de mitigação de ataques e balanceamento de tráfego. Para a realização de experimentos sobre o sistema proposto, foi utilizada a plataforma Mininet[4], que emula uma SDN, e o framework Pox[5], que implementa a versão 1.0 do OpenFlow, para o desenvolvimento do controlador. Contudo, Mininet não provê suporte nativo para NFV. As funções de rede portanto foram implementadas em contêineres Docker[6]. Seus contêineres são baseados em Linux, compartilham o kernel do sistema hospedeiro, mas proveem isolamento entre as aplicações como no caso de máquinas virtuais.

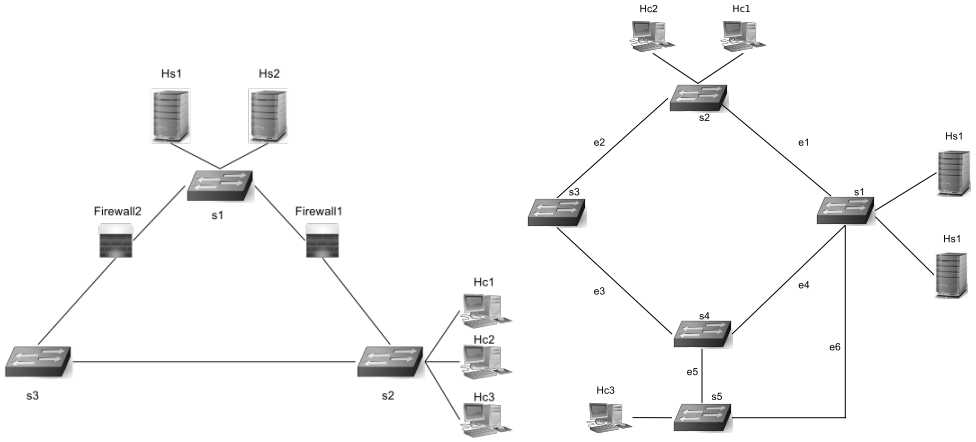
Docker provê a infraestrutura básica para hospedar funções de rede, mas não a orquestração das mesmas. Na implementação feita, a partir do projeto Containernet[7], que fornece ao Mininet suporte a contêineres Docker, esta tarefa é desempenhada conjuntamente por controlador e agente. O controlador é quem determina quais pacotes passem por *firewalls* instanciados, por exemplo, antes de chegar ao seu destino. Também é ele quem coleta métricas e as envia para o agente construir os perfis e 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. A instanciação é feita por meio de uma chamada especial, *net.addDocker*. O agente também envia ao controlador solicitação de ações como a alteração da rota entre um par de hosts. O controlador monitora as tabelas de fluxos dos switches e envia as métricas necessárias para a formação dos estados em cada *time step* ao agente. O controlador também calcula os caminhos possíveis entre os hosts e envia essa relação ao agente.

4.2. Análise Experimental

Foram realizados dois experimentos. Em ambos, servidores estão em contêineres Docker e enviam arquivos de vídeo a clientes para gerar tráfego através de conexão TCP. Clientes

[3]https://github.com/phfaustini/containernet
[4]http://mininet.org/
[5]http://sdnhub.org/tutorials/pox/
[6]https://www.docker.com/what-docker
[7]https://github.com/containernet/containernet

são hosts usuais do Mininet. Links que conectam clientes e servidores a switches têm vazão máxima de 5 Mbps. Os demais trafegam até 11 Mbps. A banda consumida pelos clientes é 5 Mbps, que é o limite para os links que conectam hosts a switches. Para realizar os experimentos, foi utilizado um computador com processador Intel i7 de 3,4 Ghz e 16 GB de Ram, em uma máquina virtual executando o sistema operacional Ubuntu 14.04 (dos quais 10 GB Ram foram alocados para ela). Também em ambos os casos, foi adotada uma taxa de aprendizagem $\alpha = 0.1$ (para evitar que uma ação isolada possa impactar fortemente a política de decisões do agente), taxa de exploração iniciando em $\epsilon = 0.4$ (pois espera-se explorar prioritariamente uma ação conhecida, mas não raramente descobrir as consequências de outras ações) e dois segundos para cada *time step*.



(a) Topologia usada no experimento #1. (b) Topologia usada no experimento #2

Figura 2. Topologias usadas nos experimentos

No primeiro experimento, há dois servidores (Hs_1 e Hs_2) e três clientes (Hc_1, Hc_2 e Hc_3), conforme mostra a Figura 2 (a). O agente procura balancear o tráfego em direção aos servidores e precisa também mitigar um ataque. Neste caso, o perfil de mitigação possui prioridade mais alta em relação ao perfil de balanceamento. Duas VNFs que implementam um *firewall* têm a função de proteger os servidores de um eventual ataque vindo através dos switches *s2* e *s3*. Elas foram implementadas em contêineres por meio das ferramentas *bridge-utils*[8] e *ebtables*[9]. Também foi estipulado que após 40 *time steps* a taxa de exploração cairia para $\epsilon = 0.1$.

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. Inicialmente a troca de pacotes ocorre por meio do link que conecta os switches *s1* e *s2*. O arquivo trocado em cada conexão tem 150 Mb. O cliente Hc_3 é malicioso e executa diversas requisições TCP a fim de exaurir o servidor Hs_1. Rapidamente os links ficam sobrecarregados em função dos muitos arquivos sendo transmitidos, e o agente não encontra uma rota alternativa que ocasione 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.

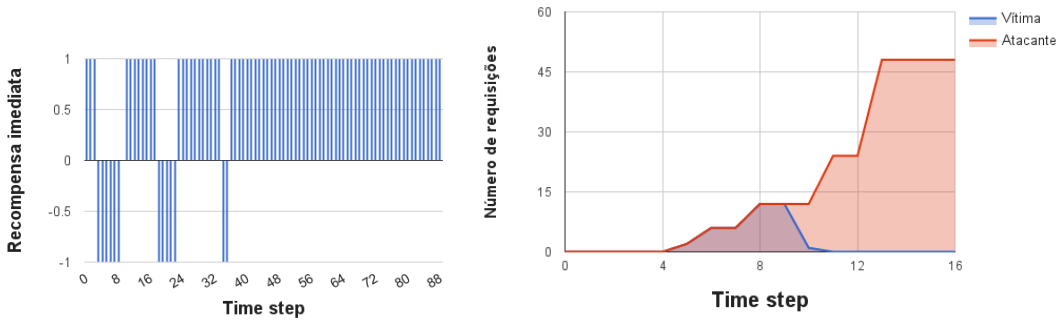
A Figura 3 (a) mostra as recompensas imediatas obtidas pelo agente para o perfil

[8]https://wiki.linuxfoundation.org/networking/bridge

[9]http://ebtables.netfilter.org

de balanceamento de carga. 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. Depois que a taxa de aprendizagem foi reduzida para 0.1, no *time step* 40, conforme o estipulado, observa-se que o agente somente adquiriu recompensas positivas. Isso evidencia que de fato encontrou ações que evitem o congestionamento dos links, caso contrário receberia recompensas negativas. Sobre os ataques, a Figura 3 (b) mostra a quantidade de requisições feitas pelo cliente malicioso Hc_3 (em vermelho) e o número de requisições que de fato chegaram 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.

O segundo experimento aborda somente balanceamento de tráfego, mas em uma topologia maior, permitindo mais combinações de rotas alternativas, conforme mostra a Figura 2 (b). Os hosts Hs_1 e Hs_2 estão conectados ao switch *s1* e agem como servidores, ao passo que Hc_1 e Hc_2 estão conectados no switch *s2* e são configurados como clientes.



(a) Recompensas imediatas obtidas. (b) Requisições enviadas e recebidas.

Figura 3. Recompensas e requisições

Um arquivo de 500 Mb é transmitido entre os hosts Hs_1 e Hc_1 e outro idêntico entre os hosts Hs_2 e Hc_2 ao mesmo tempo. O caminho inicial adotado pelo controlador para o encaminhamento dos pacotes é o caminho mínimo. Após 100 *time steps*, o terceiro cliente fazia o mesmo com o servidor Hs_1. A Figura 4 mostra as recompensas imediatas obtidas ao longo do tempo pelo agente, durante o balanceamento de carga.

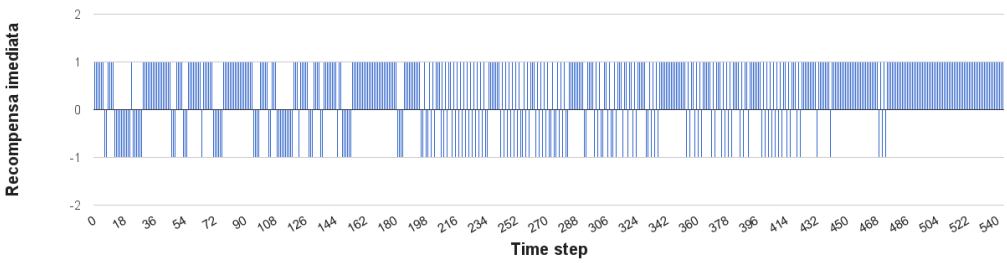
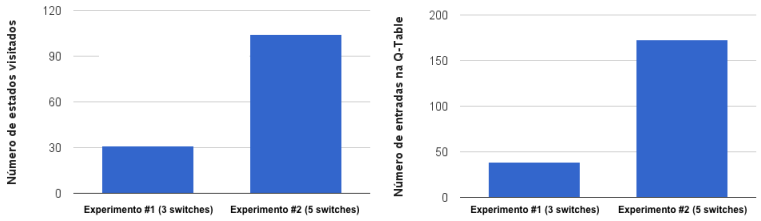


Figura 4. Recompensas imediatas obtidas ao longo do tempo

Após 400 *time steps*, a taxa de exploração foi reduzida para $\epsilon = 0.1$, e o agente

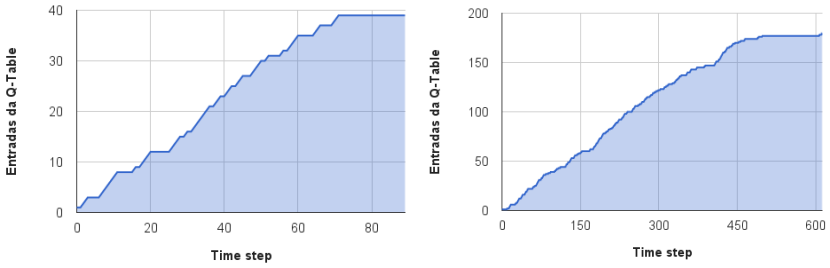
conseguiu encontrar um caminho que evite gargalos, obtendo recompensas positivas, conforme mostra a Figura 4. Em 10% dos casos, ele explorava ações desconhecidas, o que o levava a recompensas negativas, mas em proporção muito menor em relação ao início do experimento.

Analisando-se os dois experimentos, registra-se o baixo desempenho espacial da técnica de se armazenar estados em tabelas. O primeiro experimento apresentava uma topologia com três switches, e o segundo acrescentava 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 5 (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 5 (b).



(a) Número de estados visitados (b) Tamanho da tabela estado-ação

Figura 5. Tamanho da tabela estado-ação e número de estados visitados



(a) Q-Table no experimento #1 (b) Q-Table no experimento #2

Figura 6. Crescimento da tabela estado-ação

Quanto mais métricas forem coletadas para a formação de um estado, com mais intensidade esse fenômeno tende a ocorrer. No caso do balanceamento de tráfego, 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 o número de switches, mais caminhos possíveis entre os hosts existirão, levando a uma grande combinação de estados possíveis. O armazenamento dessas informações e o tempo necessário para o treinamento do agente pode ser custoso em uma rede de maior porte.

As figuras 6 (a) e 6 (b) mostram o quanto o número de entradas na Q-Table aumenta conforme o progresso dos *time steps* para os dois experimentos. 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 atenuar, esse fenômeno. Como a taxa de exploração foi reduzida ao final dos dois experimentos, o agente parou de explorar ações desconhecidas. Consequentemente, novas combinações de tráfego/ação pararam de ser acrescentadas à Q-Table, e houve um estancamento no seu crescimento.

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 SDN ou não. O mesmo vale para trabalhos que mesclam características de NFV e SDN. Esta seção seleciona publicações que abordam ataques de negação de serviço, que adotam aprendizagem por reforço para realizar balanceamento de tráfego e que procuram identificar anomalias em redes baseadas em infraestruturas SDN/NFV.

Machado, Granville e Schaeffer-Filho (2016) propõem uma arquitetura que combina SDN e NFV para promover resiliência. 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 *feedback* do *control-loop*.

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.

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 procura aprender ao interagir com o ambiente.

Hu e Chen (2013) adotaram aprendizagem por reforço supervisionada para balancear o tráfego. 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. 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 definiram um limiar em cada link, e 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 propor um agente que não somente distribui o tráfego, mas também se necessário instancia réplicas de servidores ao realizar o balanceamento.

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 e de técnicas baseadas em aprendizagem por reforço. Propôs-se um agente que reside no orquestrador da arquitetura NFV e coleta evidências de anomalias a partir de métricas de rede. As métricas, 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. Tal 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 experimentos 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 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.

Por fim, vislumbra-se para o futuro aprimoramentos no sistema aqui apresentado a ampliação do universo de perfis modelados. Por exemplo, incluir monitoramento de escaneamento de portas, sistema detector de intrusões (IDS), entre outros. Também propõe-se o emprego de mecanismos mais sofisticados para detecção de ataques, uma vez que esse não foi o foco do artigo, onde optamos por adotar uma abordagem mais simplista. No entanto, uma alternativa para o futuro seria a adoção de técnicas, por exemplo, baseadas em entropia [da Silva et al. 2016]. Além disso, a adoção de outras técnicas para representação dos estados dos perfis de rede é um tópico a ser investigado. Em vez de usar a abordagem de discretização de valores e posterior armazenamento em tabelas, poderia-se adotar redes neurais ou Tile-Coding [Sutton e Barto 2012]. Por fim, citamos a coleta de outras métricas para a representação dos estados. Por exemplo, o perfil de réplica de servidor poderia levar em consideração não somente a vazão no enlace de acesso do servidor à rede, mas também consumo de CPU e uso de memória.

Referências

B. A. A. Nunes, M. M., Nguyen, X. N., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634.

Belyaev, M. and Gaivoronski, S. (2014). Towards load balancing in sdn-networks during ddos-attacks. In *Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014 International*, pages 1–6.

da Silva, A. S., Wickboldt, J. A., Granville, L. Z., and Schaeffer-Filho, A. (2016). ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 27–35.

ETSI (2012). Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. V. 1, 22-24/10/2012.

ETSI (2014). Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress. V. 3, 14-17/10/2014.

Herrera, J. G. and Botero, J. F. (2016). Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management*, PP(99):1–1.

Hu, Z. and Chen, H. (2013). Network load balancing strategy based on supervised reinforcement learning with shaping rewards. In *Intelligent Control and Information Processing (ICICIP), 2013 Fourth International Conference on*, pages 393–397.

Kim, S., Son, J., Talukder, A., and Hong, C. S. (2016). Congestion prevention mechanism based on Q-learning for efficient routing in SDN. In *2016 International Conference on Information Networking (ICOIN)*, pages 124–128.

King, D., Farrel, A., and Georgalas, N. (2015). 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)*, pages 1–6.

Machado, C. C., Granville, L. Z., and Schaeffer-Filho, A. (2016). ANSwer: Combining NFV and SDN features for network resilience strategies. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 391–396.

Malialis, K. (2014). *Distributed Reinforcement Learning for Network Intrusion Response. University of York*. PhD thesis, United Kingdom.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

ONF (2014). Sdn architecture overview.

Sterbenz, J. P. G., Hutchison, D., Çetinkaya, E. K., Jabbar, A., Rohrer, J. P., Schöller, M., and Smith, P. (2010). Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines. *Comput. Netw.*, 54(8):1245–1265.

Sutton, R. S. and Barto, A. G. (2012). *Introduction to Reinforcement Learning*. MIT Press, Massachusetts, EUA, 2 edition.

Yau, D. K. Y., Lui, J. C. S., Liang, F., and Yam, Y. (2005). Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles. *IEEE/ACM Trans. Netw.*, 13(1):29–42.