UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM REDES DE COMPUTADORES

JARDEL GONÇALVES FERREIRA

BEAVER VIEW: UMA FERRAMENTA INTERATIVA PARA O GERENCIAMENTO DE INFRAESTRUTURA VIRTUAL *DOCKER* BASEADA EM SDN

QUIXADÁ

2018

JARDEL GONÇALVES FERREIRA

BEAVER VIEW: UMA FERRAMENTA INTERATIVA PARA O GERENCIAMENTO DE INFRAESTRUTURA VIRTUAL *DOCKER* BASEADA EM SDN

Monografia apresentado no Curso de Graduação em Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Tecnólogo em Redes de Computadores.
Área de concentração: Computação.

Orientador: Prof. Dr. João Marcelo Uchôa de Alencar

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F441b Ferreira, Jardel Gonçalves.
Beaver View: Uma ferramenta interativa para o gerenciamento de infraestrutura virtual Docker baseada em SDN / Jardel Gonçalves Ferreira. – 2018.
55 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2018.
Orientação: Prof. Dr. João Marcelo Uchôa de Alencar.

1. Docker (Software). 2. Software defined networking (Computer network technology). I. Título.

CDD 004.6

JARDEL GONÇALVES FERREIRA

BEAVER VIEW: UMA FERRAMENTA INTERATIVA PARA O GERENCIAMENTO DE INFRAESTRUTURA VIRTUAL *DOCKER* BASEADA EM SDN

Monografia apresentado no Curso de Graduação em Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Tecnólogo em Redes de Computadores.
Área de concentração: Computação.

Aprovada em: ____/_____/_____.

BANCA EXAMINADORA

Prof. Dr. João Marcelo Uchôa de Alencar (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Marcos Dantas Ortiz
Universidade Federal do Ceará (UFC)

Prof. Me. Jeandro de Mesquita Bezerra
Universidade Federal do Ceará (UFC)

Dedico o presente trabalho aos meus pais, que sempre me apoiaram nos melhores e piores momentos e que fizeram de tudo para a faculdade se tornar um sonho possível. A minha luta sempre foi a de vocês. A minha vitória, será eternamente nossa.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde e forças para superar as dificuldades.

Aos meus pais Francisca Vilani Gonçalves e Antônio Inácio Ferreira e aos meus irmãos por todo o apoio não somente durante a graduação, mas por toda vida.

Meus agradecimentos aos meus amigos, em especial Mara Vitória, Flávio Júnior, Auricio Nogueira, Hugo Garcia, Aldair Silva, Antônio Carlos, Jefté Rufino, Matheus Cavalcante, Calebe Tavares e Esdras Emanuel, que fizeram parte da minha formação e que vão continuar presentes em minha vida.

Ao Professor Dr. João Marcelo Uchôa de Alencar pela oportunidade, apoio, paciência e dedicação na elaboração deste trabalho. Aos professores participantes da banca examinadora pelo tempo, pelas valiosas colaborações e sugestões.

A esta universidade, a todos os professores que se dedicaram a passar os seus conhecimentos necessários para que eu pudesse chegar até esse momento.

Aos colegas da turma de graduação, pelas reflexões, críticas e sugestões.

"People think focus means saying yes to the thing you've got to focus on. But that's not what it means at all. It means saying no to the hundred other good ideas that there are. You have to pick carefully."

(Steve Jobs)

RESUMO

A flexibilidade e programabilidade fornecidas pelo paradigma de Redes Definidas por Software (SDN) possibilitou novos serviços e aplicativos para controlar e gerenciar infraestruturas de comunicação digital, fornecendo novas oportunidades para construção de cenários robustos e extensíveis. Apesar dos benefícios oferecidos pela abordagem SDN, o gerenciamento de infraestrutura continua sendo um desafio devido a diferentes mecanismos integrado ao mesmo. Em infraestrutura virtualizada como o *Docker*, o controlador SDN não atua diretamente com as *bridges* do *Docker* dificultando a aplicação de políticas de QoS. Este trabalho teve como objetivo o desenvolvimento de uma ferramenta para auxiliar o administrador de rede a lidar com as complexidades de infraestrutura SDN. Optamos por uma abordagem de SDN Híbrida para ativar o controle de QoS e políticas de *Firewall*. Para validar a ferramenta proposta, dois experimentos foram executados, onde o objetivo foi bloquear e permitir o tráfego na rede através da funcionalidade de *firewall* e limitar a taxa de transmissão de um contêiner utilizando a disciplina de fila TBF. Os resultados obtidos demonstram que a solução proposta cumpriu os objetivos e é uma opção viável para desenvolvedores, administradores de redes e pesquisadores.

Palavras-chave: Docker. Redes Definidas por Software

ABSTRACT

The flexibility and programmability provided by the Software Defined Networks (SDN) paradigm have enabled new services and applications to control and manage digital communications infrastructures, giving new opportunities for building robust and extensible scenarios. Despite the benefits offered by the SDN approach, infrastructure management remains a challenge due to different mechanisms integrated with it. In a virtualized infrastructure such as Docker, the SDN controller does not act directly with the bridges of Docker, making it difficult to apply QoS policies. This work aims to provide a tool for the network administrator that will alleviate his task of dealing with the complexities of SDN infrastructure. We opted for a Hybrid SDN approach to enable QoS control and Firewall policies. To validate the proposed tool, we performed two experiments, where the goal was to block and allow network traffic through the firewall feature and limit the transmission rate of a container using the TBF row discipline. The results show that the proposed solution met the objectives and is a viable option for developers, network administrators, and researchers.

Keywords: Docker. Software Defined Networks.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Interface de Programação de Aplicativo*
GUI	*Interface Gráfica do Usuário*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LXC	*Linux Containers*
MAC	*Media Access Control*
NaN	*Not a Number*
NOS	*Sistema Operacional de Rede*
OVS	*Open vSwitch*
QoS	*Quality of Service*
REST	*Representational State Transfer*
SDN	*Software Defined Networking*
TC	*Traffic Control*
VXLAN	*Virtual Extensible Local Area Network*
WSGI	*Web Server Gateway Interface*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

Nos últimos anos, a área de redes tornou-se um sucesso devido a necessidade da sociedade utilizá-la para realizar atividades do dia a dia. Com isso, a Internet vem se demonstrando um mecanismo conhecido e acessado por uma parcela significativa da população, possibilitando iniciativas de inclusão digital com o intuito de expandir seu alcance para toda a população mundial (GUEDES *et al.*, 2012).

Essa característica trouxe problemas, pois novos protocolos e tecnologias não são mais viáveis na Internet tradicional, devido aos riscos de interrupção de serviços que tem a mesma como meio essencial para seu funcionamento. Tais problemas levaram diversos pesquisadores a afirmar que a arquitetura de redes em geral e a Internet tornou-se pouco flexível (GUEDES *et al.*, 2012).

Com o intuito de contornar esses problemas, surgiu o paradigma de *Software Defined Networking* (SDN) que traz como uma das suas abordagens a separação dos planos de controle e de dados permitindo que os elementos da rede sejam controlados por *software* trazendo flexibilidade em administrá-los, visto que em SDN possibilita a programabilidade da rede.

Além de fornecer flexibilidade através da programabilidade dos planos, novos mecanismos podem ser adicionados sem a necessidade de alterações nos elementos da rede, permitindo o rápido avanço tecnológico através da criação de protótipos que possibilitam a avaliação de novas estratégias avançadas de controle de *Quality of Service* (QoS), balanceamento de carga, mobilidade e segurança, entre outras (GOMES *et al.*, 2016).

Apesar das facilidades concedidas pelas tecnologias relacionadas a SDN que maximizam o uso de recursos dos elementos de rede através de novos serviços e recursos modernos. O gerenciamento de infraestruturas SDN é desafiador pois possibilita a integração de diferentes mecanismos como configurações, controle de QoS, balanceamento de carga e entre outros (GOMES *et al.*, 2016). Isso se aplica devido a falta de sistemas de controle que possam dar suporte aos administradores de rede quando confrontados com a complexidade da infraestrutura (SHARMA *et al.*, 2013).

Em ambientes de infraestrutura virtualizada como no caso da ferramenta *Docker*[1], uma rede com múltiplos hospedeiros *Docker* faz com que o controlador SDN atue somente nos

[1] https://www.docker.com/

switches virtualizados nos hospedeiros. Esse cenário apresenta um desafio de gestão, já que o controlador não se comunica com as *bridges* do *Docker* dificultando a aplicação de políticas de QoS.

Um dos fatores críticos encontrados no gerenciamento de serviços (e.g *firewall* e Qos) em SDN é a ausência de visualização e operação da topologia em tempo de execução, que são informações essenciais para administradores de rede com pouco ou nenhum conhecimento em desenvolvimento de *software*. Uma ferramenta de gerenciamento e orquestração SDN deve fornecer um conjunto de serviços ao administrador de rede para auxiliá-lo no gerenciamento de infraestrutura por completo (GOMES *et al.*, 2016).

Neste contexto, o presente estudo tem como objetivo desenvolver um solução de gerenciamento interativo de infraestrutura virtual *Docker* baseada em SDN fornecendo facilidades e auxiliando os administradores de redes a lidar com as complexidades de infraestrutura SDN. A infraestrutura virtual *Docker* foi escolhido devido ao crescente uso da tecnologia e a ausência de soluções maduras do seu uso em conjunto com a SDN. Questões relacionadas a área de segurança, não faz parte do escopo do nosso trabalho.

O restante do trabalho está organizado da seguinte maneira. Na Seção 1.2, são apresentados os objetivos geral e específicos. No Capítulo 2, é apresentada a fundamentação teórica. No Capítulo 3 apresenta os trabalhos relacionados. No Capítulo 4 detalha a arquitetura, solução proposta e explica como usá-la. No Capítulo 5 é apresentado a avaliação da ferramenta. No Capítulo 6, é apresentado a conclusão.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver uma solução de gerenciamento de infraestrutura virtual *Docker* baseada em SDN a fim de facilitar e auxiliar os administradores de redes a lidar com as complexidades ligadas a infraestrutura SDN de forma dinâmica e interativa.

1.2.2 Objetivos específicos

- Fornecer uma REST API para aplicação de políticas de QoS baseado em TBF em hospedeiros *Docker*.
- Definir arquitetura para integração dos componentes do ambiente de virtualização

Docker e uma ferramenta de alto nível para interação com a API.

- Desenvolver uma interface *web* para administração de infraestruturas de contêineres com suporte a políticas de QoS em redes definidas por *software*.

2 FUNDAMENTAÇÃO TEÓRICA

As seções a seguir apresentam conceitos importantes relacionados a este trabalho, incluindo Redes Definidas por *Software* (Seção 2.1) e contêineres *Docker* (Seção 2.2).

2.1 Redes Definidas por *Software*

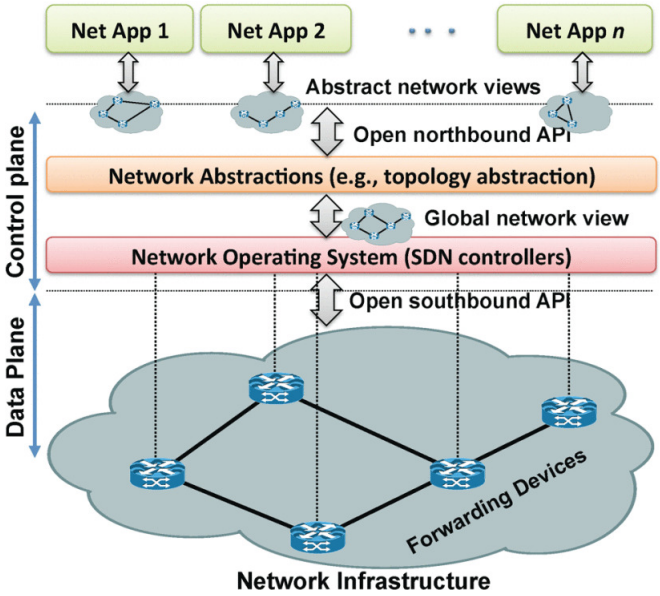
As Redes Definidas por *Software* (ou SDN) são um novo paradigma que possibilita alterar as limitações de infraestrutura de rede, flexibilizando a mesma através da separação entre plano de controle e o plano de dados. Segundo Kreutz *et al.* (2015), a SDN separa a lógica do plano de controle da rede dos roteadores e *switches* que encaminham o tráfego (plano de dados) tornando os *switches* dispositivos de encaminhamento simples e a lógica de controle é implementada em um controlador centralizado. Essa separação pode ser realizada por meio de uma interface de programação bem definida entre os *switches* e o controlador SDN.

Na Figura 1 é demonstrado um modelo de arquitetura da tecnologia SDN, no qual podemos observar que os planos de dados e controle são separados, permanecendo apenas o plano de dados nos dispositivos. O **Plano de dados** é responsável pelo encaminhamento de tráfego de acordo com as instruções/lógica do plano de controle. Já o **Plano de controle** toma as decisões de como o plano de dados deve comportar. Essas decisões podem ser sobre roteamento, *firewall*, priorização de pacotes, dentre outras. Já os **Controladores** são dispositivos que se comunicam com equipamentos programáveis da rede obtendo uma visão global da rede e definem rotas que serão implantadas nos equipamentos de rede.

Em redes *Internet Protocol* (IP) tradicionais, os planos de controle e dados são implementadas nos mesmos dispositivos de rede, ou seja, todos os dispositivos são responsáveis pelo encaminhamento de pacotes, realizando as tomadas de decisão e gerando assim a tabela de roteamento/encaminhamento. Nesses equipamentos, um *software* proprietário realiza o controle de tomadas de decisões e por ser proprietário se torna impossível de executar qualquer tomada de decisão que não tenha sido estabelecida nos mesmos.

Na nova arquitetura de redes baseadas em *software*, segundo Hakiri *et al.* (2014), torna-se possível controlar, alterar e gerenciar dinamicamente o comportamento da rede por meio de interfaces de programação abertas, permitindo o controle centralizado dos dispositivos de encaminhamento de dados independentemente da tecnologia usada para interligar esses dispositivos e dos fornecedores. Esse controle centralizado fornece uma visão global da rede

Figura 1 – Arquitetura SDN.

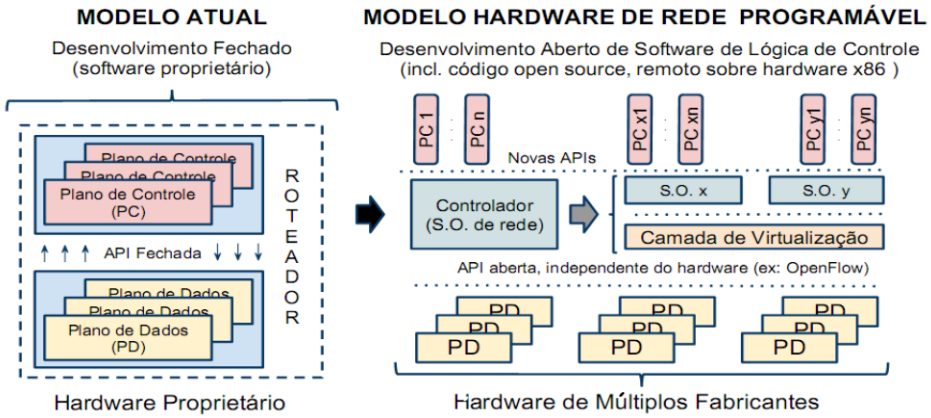


Fonte: Kreutz *et al.* (2015)

permitindo ao controlador executar funções de gerenciamento de rede.

Na Figura 2, é mostrada uma comparação entre as arquiteturas de roteadores. Podemos observar que no modelo atual, todas as decisões de roteamento são transferidas para o plano de controle via *Interface de Programação de Aplicativo* (API) proprietária, tendo suas funcionalidades limitadas pelo fabricante. Já modelo programável o desenvolvimento de aplicações é realizado via API abertas, o que permite aos administradores implementar a rede de acordo com suas necessidades.

Figura 2 – Modelo de rede atual *versus* Modelo de rede programável



Fonte: Rothenberg *et al.* (2010)

De acordo com Kreutz *et al.* (2015), as SDN possuem quatro pilares que ajudam a definir sua arquitetura de rede:

- Os planos de controle e de dados são separados. As responsabilidades atribuídas do plano de

controle são removidas dos dispositivos de rede tornando dispositivos simples de encaminhamento de tráfegos.

- As decisões de encaminhamento são baseadas em fluxo. Um fluxo é definido como um conjunto de valores de campo de pacotes agindo como um filtro e um conjunto de instruções. Em um fluxo de pacotes entre origem e destino, todos recebem políticas de serviços iguais nos equipamentos de rede como por exemplo em roteadores, *switches* e *firewalls*.
- A programação da rede é dada por meio de *software* executados sobre o controlador interagindo com os planos de dados dos dispositivos. Essa é a principal característica e proposta de valor da SDN.
- A lógica de controle se torna responsabilidade de um ou mais controladores SDN, na qual o mesmo é uma entidade externa.

A infraestrutura SDN diverge das redes tradicionais, ambas possuem os mesmos conjuntos de elementos como rotadores, *switches* e etc. Porém, o que difere é o comportamento, nas redes SDN esses elementos ficam apenas como simples encaminhadores de tráfegos, sem a existência de um *software* para realizar o controle das tomadas de decisões. Como citado anteriormente, o plano de controle é movido para uma entidade externa, denominada controlador de rede.

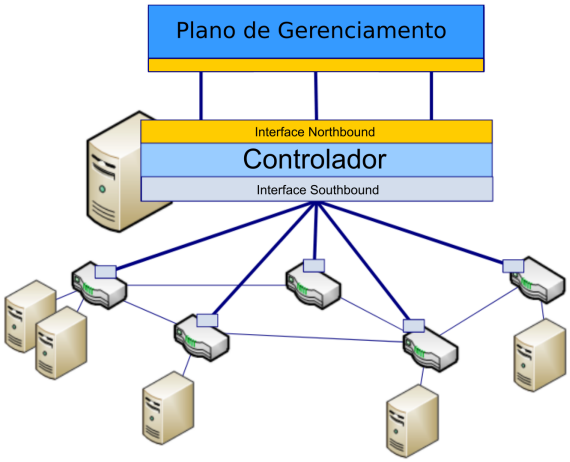
A SDN fornece novos recursos de gerenciamento e monitoramento que podem melhorar o desempenho da rede. É importante salientar os principais elementos da tecnologia SDN. Abaixo apresentamos os elementos junto a sua definição de acordo com Kreutz *et al.* (2015):

- **Interface *Southbound*:** API que define o conjunto de instruções nos dispositivos e os protocolos de comunicação entre os elementos de encaminhamento e os do plano de controle.
- **Interface *Northbound*:** o sistema operacional de rede fornece uma API para o desenvolvimento de aplicativos. Essa API representa uma interface para o *northbound*, abstraindo os conjuntos de instruções de baixo nível usadas pelas interfaces para o *southbound*, possibilitando a programação dos dispositivos de encaminhamento.
- **Plano de Gerenciamento:** faz uso de um conjunto de aplicativos que aproveitam as funções fornecidas pela interface *Northbound* para implementar a lógica de controle e operação da rede. Isso inclui aplicativos de roteamento, *firewall* e monitoramento, sendo que esses aplicativos de gerenciamento definem as políticas para a interface *Southbound*, de forma que a mesma programa o comportamento dos dispositivos de encaminhamento.

- **Dispositivos de encaminhamento:** dispositivos de planos de dados baseados em *hardware* ou em *software* com a responsabilidade apenas de encaminhamento de pacotes. Possuem conjuntos de instruções bem definidas usadas para executar determinadas ações nos pacotes recebidos. Essas instruções são definidas pela interface *Southbound* e instaladas nos dispositivos pelos controladores SDN.

Como podemos observar na Figura 3, os dispositivos de encaminhamento possuem o plano de dados onde a comunicação entre essa entidade e controlador é realizada através da interface *Southbound* e a comunicação entre o controlador e o plano de gerenciamento é realizada via interface *Northbound*.

Figura 3 – Elementos da tecnologia SDN.



Fonte: Modificada pelo autor.

Com o intuito de padronizar a comunicação entre controlador e dispositivos de rede, o protocolo de comunicação *OpenFlow* surgiu como o principal meio de comunicação entre controlador e *switches*, que será apresentado na subseção 2.1.1. Em nosso trabalho, utilizaremos a abordagem da SDN, como também da rede tradicional tornando assim uma solução de SDN híbrida, isso se dá devido a incapacidade de realizar a comunicação entre o controlador SDN e as *bridges* do *Docker*.

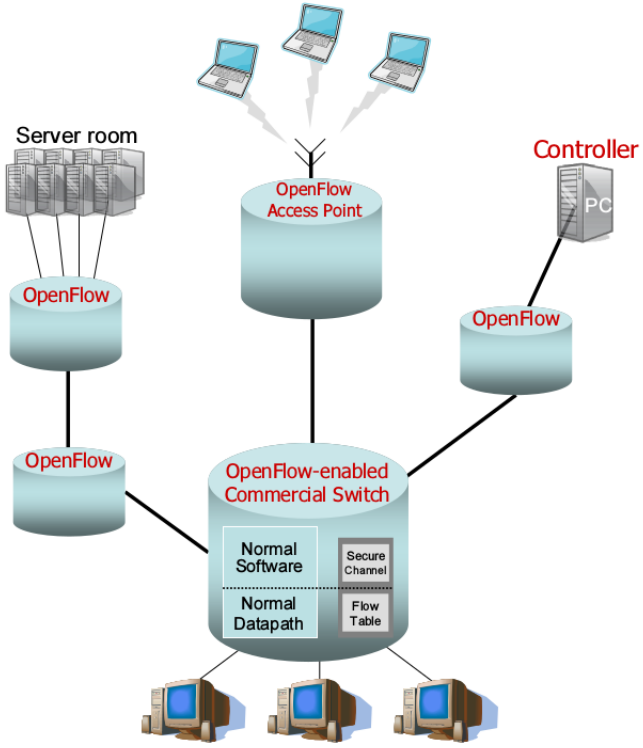
2.1.1 OpenFlow

Com o aumento de novas propostas de arquitetura e protocolos de rede sobre os equipamentos proprietários, surgiu a necessidade de atender a essa demanda de validação das mesmas. Para sanar essa limitação, a Universidade de *Stanford* propôs o *OpenFlow*, um protocolo que define um padrão controle dos dispositivos de rede, determinando as ações que

são responsabilidades de um elemento externo, chamado de controlador (ROTHENBERG *et al.*, 2010).

A Figura 4 apresenta uma rede que utiliza o protocolo *Openflow* para permitir que um dispositivo possa ser controlado por um ou mais controladores, esse protocolo funciona como uma interface de comunicação entre os dispositivos e os controladores. De acordo com Costa (2013), o *OpenFlow* tem como objetivo ser ágil atendendo os seguintes requisitos:

Figura 4 – Rede Openflow



Fonte: McKeown *et al.* (2008)

- Ser passível de implementação a baixo custo com alto desempenho.
- Proporcionar o suporte de uma vasta gama de pesquisas científicas.
- Assegurar o isolamento do tráfego de produção e o tráfego experimental.
- Não ser necessário expor o projeto dos fabricantes em suas plataformas, permitindo o acesso a soluções proprietárias através de interfaces padronizadas.

Para que uma rede se torne programável com *OpenFlow*, há a necessidade que sua arquitetura seja constituída por quatro componentes básicos. A seguir, apresentamos os quatro componentes e suas funcionalidades essenciais de acordo com Rothenberg *et al.* (2010):

- **Tabela de Fluxos:** todas as entradas na tabela de fluxos do *hardware* de rede são constituídas por regras, ações e controles de métricas estatísticas. A regra está associada a um valor de um ou mais campos do cabeçalho do pacote, através do mesmo que é determinado o

fluxo de tráfego. São as ações que determinam o tratamento do pacote, se será descartado, encaminhado, para onde será encaminhado ou como deve ser processado. Os controles de estatísticas fazem uso de contadores com o objetivo de armazenar estatísticas de utilização e remover fluxos inativos.

- **Protocolo *OpenFlow*:** protocolo aberto para comunicação que utiliza uma interface externa para a troca de mensagens entre os dispositivos e os controladores habilitando o gerenciamento dos mesmos.
- **Controlador**: *software* que realiza as tomadas de decisões e também operações como adicionar e remover as entradas na tabela de fluxo, atuando como um *Sistema Operacional de Rede* (NOS).
- **Canal Seguro**: a fim de evitar que os dispositivos sofra ataques mal-intencionados, o canal seguro garante segurança na troca de informações entre o *switch* e o controlador.

De acordo com McKeown *et al.* (2008), cada fluxo entrante possui uma ação associada. As três ações básicas em que todo dispositivo *OpenFlow* deve apoiar são:

1. Encaminhar pacotes deste fluxo para uma ou mais portas, permitindo que os pacotes possam trafegar através da rede.
2. Encapsular e encaminhar pacotes para um controlador, isso normalmente acontece para os primeiros pacotes de um fluxo, pois as decisões são tomadas em um controlador, pois o mesmo pode decidir se o fluxo deve ser adicionado à tabela de fluxo.
3. Descartar pacotes deste fluxo, usado por questões de segurança por exemplo em casos de ataques de negação de serviço ou para reduzir o tráfego de descoberta.

Em nosso projeto, utilizaremos o padrão *OpenFlow* através do comutador *Open vSwitch* (OVS) usado para ambientes virtuais. Segundo Pfaff *et al.* (2009), a abordagem do OVS difere das tradicionais. Ele possui uma interface para controle aprimorado do encaminhamento que pode ser usado para suportar regras de QoS, encapsulamento e filtragem, também suporta uma interface remota permitindo a migração do estado de configuração.

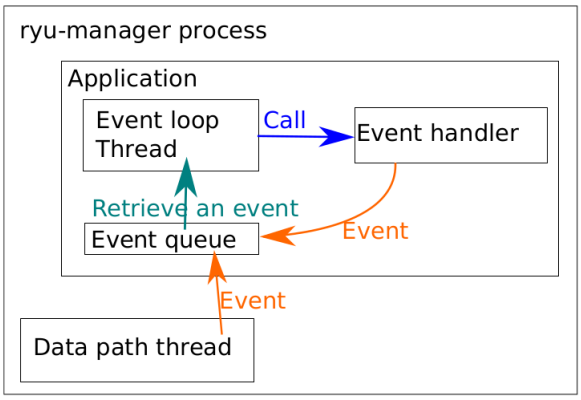
Além disso, a implementação do OVS tem um mecanismo de encaminhamento flexível, baseado em tabela, que pode ser usado para particionar logicamente o plano de encaminhamento, também utilizaremos o controlador Ryu que será apresentado na subseção 2.1.2, ele foi escolhido por ser desenvolvido na linguagem de programação *Python* que de certo modo ela facilita o desenvolvimento da ferramenta (linguagem de *script*), possui uma boa documentação e a sua comunidade está ativa.

2.1.2 Controlador Ryu

O Ryu é um controlador SDN desenvolvido com a linguagem de programação *Python* e sua estrutura SDN é baseada em componentes, ou seja, ele fornece componentes de *software* através de uma API que visa facilitar o desenvolvimento de aplicações de gerenciamento e controle de rede. Essa solução suporta vários protocolos para gerenciar os dispositivos como por exemplo o *OpenFlow*, que será utilizado no presente estudo. Para mais informações sobre o Ryu ou o processo de instalação, o leitor pode consultar o site[1] ou a sua documentação[2].

A Figura 5 apresenta o modelo de programação usado nas aplicações Ryu, onde em *Application* toda a lógica usada pelo usuário é descrita como aplicativo. A comunicação entre aplicações é realizada através da transmissão e do recebimento de eventos (RYU, 2015).

Figura 5 – Modelo de programação de aplicações Ryu



Fonte: Ryu (2015)

Toda aplicação Ryu possui uma fila de eventos (*Event queue*) que armazena os eventos recebidos, uma *thread* que é criada automaticamente e executada em *loop* (*Event loop Thread*). Quando um evento é recebido na fila de eventos a *thread* se responsabiliza em carregar o evento para o manipulador de eventos (*Event Handler*), além disso, o usuário pode criar *threads* adicionais para realizar processamentos específicos (RYU, 2015).

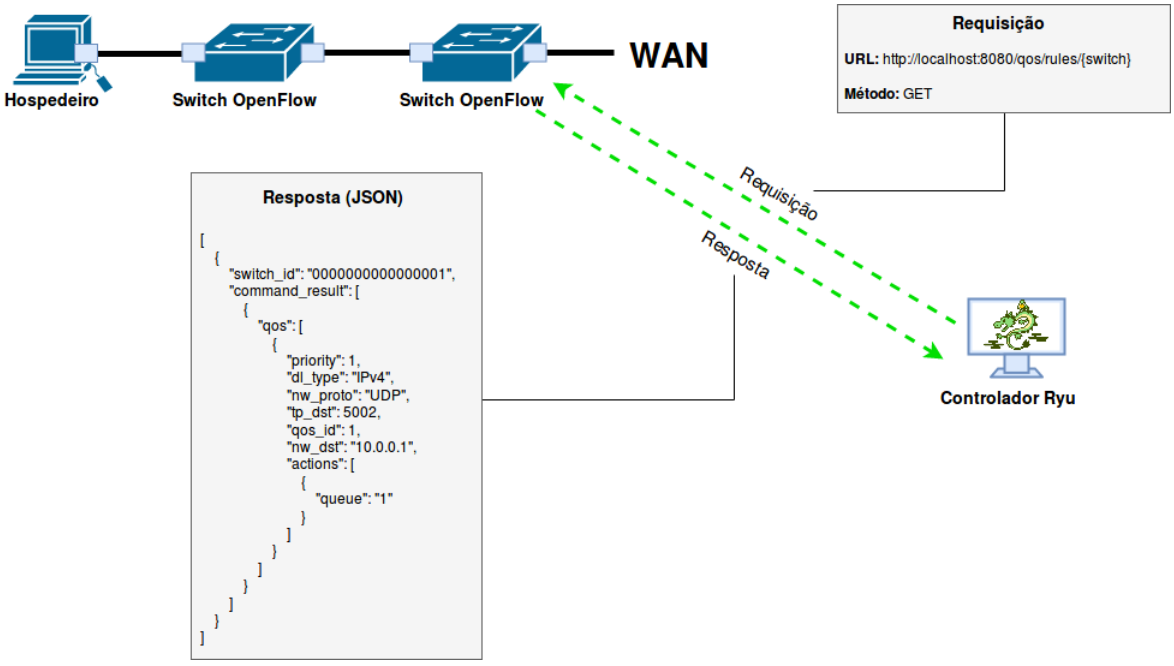
O Ryu implementa *Web Server Gateway Interface* (WSGI)[3] que é uma interface de servidor *web* que especifica como o servidor se comunica com a aplicação da *web*, através disso, o Ryu fornece uma *Representational State Transfer* (REST) API que com ela podemos facilmente aplicar políticas de QoS, regras de roteamento e *firewall* assim como podemos observar nas

[1] http://osrg.github.io/ryu/
[2] https://ryu.readthedocs.io/en/latest/index.html
[3] https://wsgi.readthedocs.io/en/latest/

figuras 6 e 7.

Na Figura 6 é demonstrado um exemplo de consumo da REST API disponibilizada pelo controlador que está sendo utilizado um método *GET* a um determinado recurso localizado por um endereço, como está sendo executado no próprio controlador, ou seja, localmente e como resposta é retornado um objeto *JavaScript Object Notation* (JSON)[1] contendo o conteúdo solicitado.

Figura 6 – Obtendo as políticas de QoS através da REST API



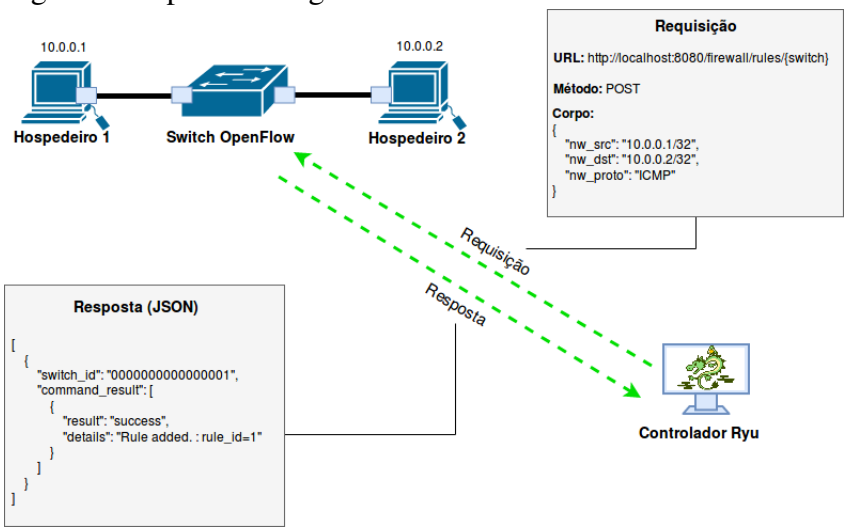
Fonte: O próprio autor.

Já na Figura 7, ao invés de utilizar o método *GET*, é executado o método *POST*. Este método submete um dado a um recurso e como ilustrado podemos observar que no corpo da requisição é enviado um objeto JSON especificando o endereço de origem, endereço de destino e protocolo respectivamente. Isso informa ao *switch OpenFlow* para adicionar uma política de *firewall* para permitir o tráfego tendo como endereço de origem o IP 10.0.0.1, endereço de destino o IP 10.0.0.2 e o tipo de tráfego pertencente ao protocolo *Internet Control Message Protocol* (ICMP). Como resposta da requisição é retornado um objeto JSON informando o *status* da solicitação que no caso foi aplicado ou um relatório de erro caso alguma informação esteja inconsistente.

Além dos métodos *GET* e *POST* existem outros métodos que caso o leitor ainda não conheça, pode ser consultada a RFC[2] do HTTP e para consultar quais recursos a REST API do

[1] https://www.json.org/
[2] https://tools.ietf.org/html/rfc2616

Figura 7 – Aplicando regras de *Firewall* utilizando a REST API



Fonte: O próprio autor.

Ryu fornece, acesse a documentação do mesmo.

2.2 *Docker*

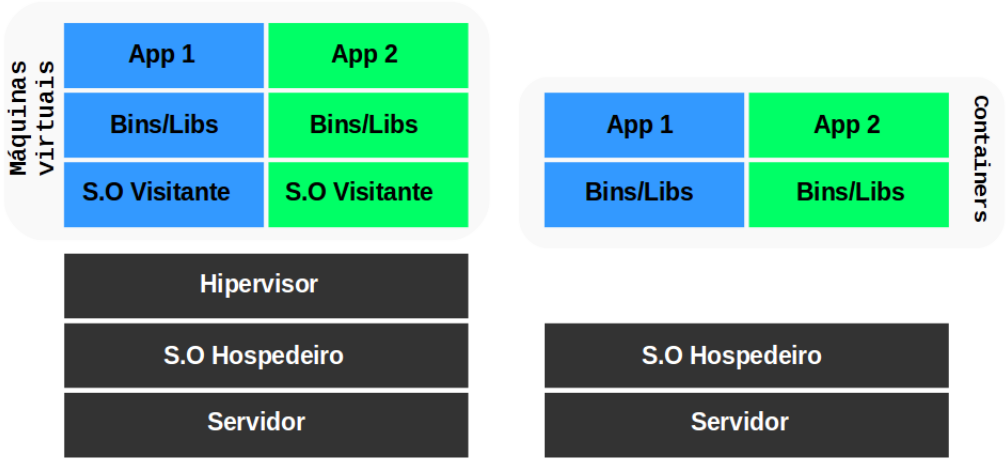
Os contêineres possuem uma longa história e notável em computação, sendo um tipo de virtualização que difere da virtualização tradicional como a de hipervisor na qual uma ou mais máquinas independentes executam virtualmente o *hardware* físico por meio de uma camada intermediária. Já os contêineres permitem executar várias instâncias de espaço de usuário isoladas em um único hospedeiro, fazendo com que sejam geralmente chamados de virtualização no nível do sistema operacional (TURNBULL, 2014).

Algumas tecnologias de contêineres incluem OpenVZ, *Solaris Zone*, *Linux Containers* (LXC) e *Docker* que usa o LXC. O *Docker* possui recursos modernos de *kernel* do Linux, como grupos de controle e *namespace*, o que possibilita o isolamento dos contêineres, sua própria rede e pilhas de armazenamento.

Não é o nosso foco discorrer sobre os tipos de virtualização, porém, para entendermos a virtualização no nível do sistema operacional utilizada pelos contêineres. Na Figura, 8 apresenta um comparativo entre a virtualização de máquinas virtuais e a virtualização de contêineres. Como podemos observar na virtualização de máquinas virtuais, é virtualizado a aplicação, os binários e/ou bibliotecas e o sistema operacional convidado através de uma camada intermediária denominada hipervisor que executa o *hardware* físico virtualmente, isso tudo executado em cima do sistema operacional do hospedeiro. Na virtualização de contêineres as aplicações, os binários e/ou bibliotecas não necessitam de camadas intermediárias tornando a virtualização leve

e podendo executar diretamente no sistema operacional hospedeiro.

Figura 8 – Virtualização de máquinas virtuais *versus* virtualização de containers



Fonte: O próprio autor

Apesar das vantagens que uma virtualização leve traz, os contêineres de início não foram aceitos em grande escala, motivo esse dado pela sua complexidade, a dificuldade de configuração e de gerenciamento. Por esses motivos o *Docker* surgiu para facilitar o uso de virtualização de contêineres minimizando a complexidade (TURNBULL, 2014).

O *Docker* é uma ferramenta de código aberto que adiciona um mecanismo de implementação de aplicativos e ambientes de contêineres virtualizados, fornecendo uma plataforma leve e rápida para executar aplicações e implantar fluxos de trabalhos levando código para ambientes de testes e para produção. O *Docker* fornece uma maneira fácil e leve de modelar a realidade, ele depende do modelo *copy-on-write* o que permite realizar alterações em aplicativos de forma rápida, permitindo aos desenvolvedores se preocuparem apenas com a aplicação.

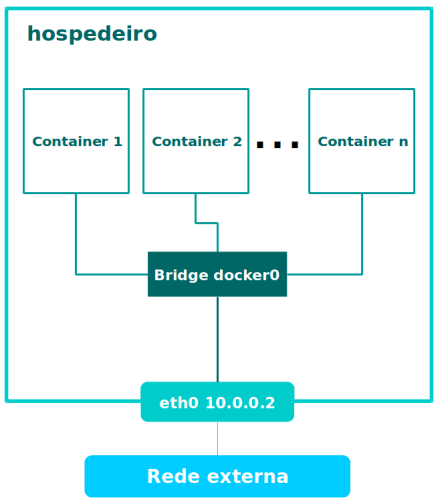
Existem quatro componentes essenciais que compõem o *Docker*, que serão apresentados de acordo com Turnbull (2014).

- **Cliente e Servidor do *Docker*:** a arquitetura do *Docker* é a cliente servidor, onde um cliente se comunica com o servidor ou *daemon* do *Docker* responsável pela realização de todo o trabalho. Tanto o *daemon* como o cliente pode ser executado no mesmo hospedeiro ou conectar o cliente *Docker* local a um *daemon* remoto.
- **Imagens do *Docker*:** contêineres podem ser lançados a partir de imagens, essas imagens são blocos de construção do mundo *Docker*, seu formato é em camadas, possui sistema de arquivos. As imagens podem ser considerada como o código-fonte dos contêineres, podem ser compartilhados e atualizados.

- **Registros:** registros são os locais onde é armazenado as imagens criadas, existem dois tipos de registros: os registros públicos conhecido como *Docker Hub* que quem cuida é a empresa que desenvolve o *Docker*, porém qualquer pessoa pode utilizar desde que crie uma conta na *Docker Hub* o que lhe permite armazenar e compartilhar suas imagens. Os registros privados permitem que você armazene imagens atrás do seu *firewall*, o que pode ser um requisito para algumas organizações.
- **Contêineres:** os contêineres são iniciados a partir de uma imagem que pode conter um ou mais processos em execução. Ele é um formato de imagem, um ambiente de execução e um conjunto de ações padrão como por exemplo ele pode ser iniciado, interrompido, reiniciado e destruído. Eles são isolados a nível de disco, memória, processamento e rede.

Na instalação padrão do *Docker*, uma *bridge* chamada de *docker0* é criada no hospedeiro. Esta *bridge* recebe um endereço IP privado e uma sub-rede associado a ela. Como podemos ver na Figura 9, todos os contêineres lançados são vinculados a *docker0*, onde a mesma faz o uso da interface do hospedeiro para se comunicar com a rede externa (DUA *et al.*, 2016).

Figura 9 – Redes de contêineres do Docker



Fonte: O próprio autor

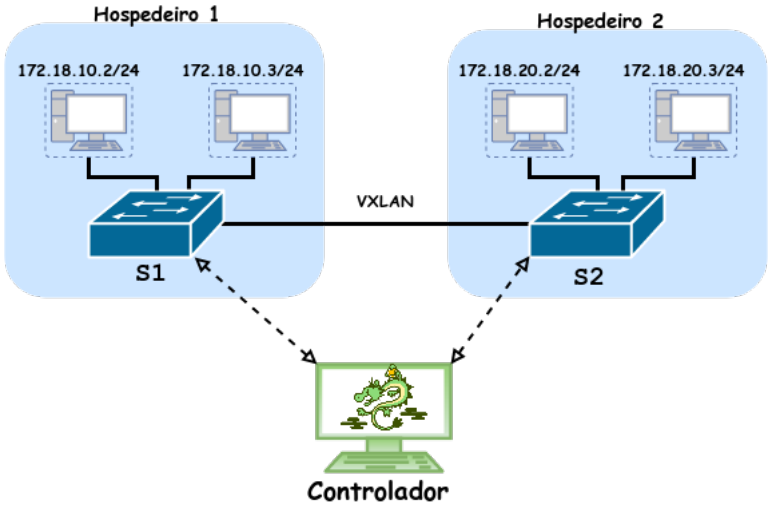
Novas *bridges* podem serem lançadas no *Docker* e configuradas de quatro modos, esses modos são apresentados de acordo com Dua *et al.* (2016).

- **Modo *Default***: a *bridge* permite que todos os contêineres possam se comunicarem uns com os outros.
- **Modo *None***: contêineres são isolados, ou seja, são impedidos de se conectarem à rede.
- **Modo *Container:$Container2***: o contêiner criado compartilha seu *namespace* de rede, ou

seja, os recursos globais são isolados impedindo que os processos de um contêiner vejam ou interajam com os do contêiner chamado de *$Container2*.

- **Modo *Host***: O contêiner criado compartilha seu *namespace* de rede com o hospedeiro.

Figura 10 – Cenário com dois hospedeiro Docker conectados



Fonte: O próprio autor.

Como apresentado nos modos de configuração da *bridge*, existe apenas comunicações dos contêineres dentro do mesmo hospedeiro, porém, para este trabalho é necessário a comunicação entre vários hospedeiros *Docker* e que possibilite utilizarmos a tecnologia SDN, para isso, usaremos o OVS que não só fornecerá a comunicação entre os hospedeiros *Docker* como também a implantação de um controlador de rede.

Para a comunicação entre os hospedeiros *Docker*, utilizaremos túneis *Virtual Extensible Local Area Network* (VXLAN) que fornecem a capacidade de criar uma rede lógica utilizando os hospedeiros *Docker* possibilitando assim a comunicação de contêineres hospedados em diferentes hospedeiros. Na Figura 10 temos o cenário com dois hospedeiros, os contêineres são representados pelos computadores com a borda pontilhada, cada hospedeiro possui o OVS instalado conectado a um outro hospedeiro através de túneis VXLAN.

3 TRABALHOS RELACIONADOS

No trabalho de Adami *et al.* (2015), os autores buscaram implementar uma solução que fosse capaz de controlar e gerenciar a rede conforme requisitos dinâmicos, então optaram por explorar SDN com o intuito de habilitar o controle de QoS a partir de uma ferramenta construída sobre um controlador que já fornece um algoritmo nativo para o cálculo de roteamento. Tal algoritmo desconsidera métricas de roteamento como largura de banda, atraso e entre outras, portanto os autores decidiram modificar este algoritmo para considerar tais métricas. O sistema implementado pelos autores, seguem etapas que iniciam com a instalação manual pelo usuário das filas nos *switches* e finaliza com o envio de regras do controlador para os *switches* de forma automatizada utilizando os dados processados na coleta.

Nosso trabalho possui alguns pontos semelhantes com o projeto mencionado acima, tal como a ideia de fornecer uma ferramenta para o gerenciamento e controle de uma infraestrutura SDN, porém difere no aspecto da solução apresentada no artigo ser uma ferramenta automatizada que não interage com o administrador da rede.

Em Isolani *et al.* (2015), os autores relatam a falta de pesquisa que visa empregar as informações de monitoramento para auxiliar o administrador a entender o comportamento da rede e interagir com ela. Em outras palavras, existem poucos projetos que visa utilizar as mensagens de monitoramento do *OpenFlow* para criar visualizações sobre a rede e abordando a configuração interativa da SDN. Além disso, os autores ressaltam que muitas soluções que lidam com monitoramento, visualização ou configuração de SDN para tarefas automatizadas não ajuda o administrador da rede a entender o comportamento da rede e a interagir com ela. Portanto, apresentam uma aplicação *Web* que permite o administrador entender e controlar o comportamento da rede através de uma *Interface Gráfica do Usuário* (GUI).

Semelhante ao trabalho de Isolani *et al.* (2015), este trabalho também busca desenvolver uma ferramenta para o monitoramento, gerenciamento e configuração de SDN, mas especificamente para redes *Docker*.

O trabalho de Gomes *et al.* (2016), atua na questão de auxiliar os administradores de rede reduzindo a complexidade no gerenciamento de infraestrutura SDN. Esse projeto habilita a interação do administrador com os dispositivos da rede de forma dinâmica, eliminando a necessidade de ter conhecimento avançado em desenvolvimento de *software*. Para isso, os autores propõem uma ferramenta de gerenciamento interativo modular para infraestrutura SDN que fornece maior escalabilidade, robustez e reutilização de código.

Assim como o trabalho de Gomes *et al.* (2016), o nosso também busca implementar uma ferramenta de gerenciamento oferecendo ao administrador a interação com os dispositivos da rede através de uma aplicação *web*, auxiliando através da visualização dos dados coletados sobre a rede com o enfoque na infraestrutura virtual *Docker*.

Baseado nos trabalhos acima relatados, o presente estudo visa a implementação de uma aplicação Web que gerencie, monitore e configure infraestruturas *Docker* baseada em SDN auxiliando os administradores de rede na tomada de decisão e reduzindo a complexidade e forneça a interação com os dispositivos da rede, assim como relatado nos trabalhos de Isolani *et al.* (2015) e de Gomes *et al.* (2016), mas utilizando uma rede com hospedeiros *Docker*.

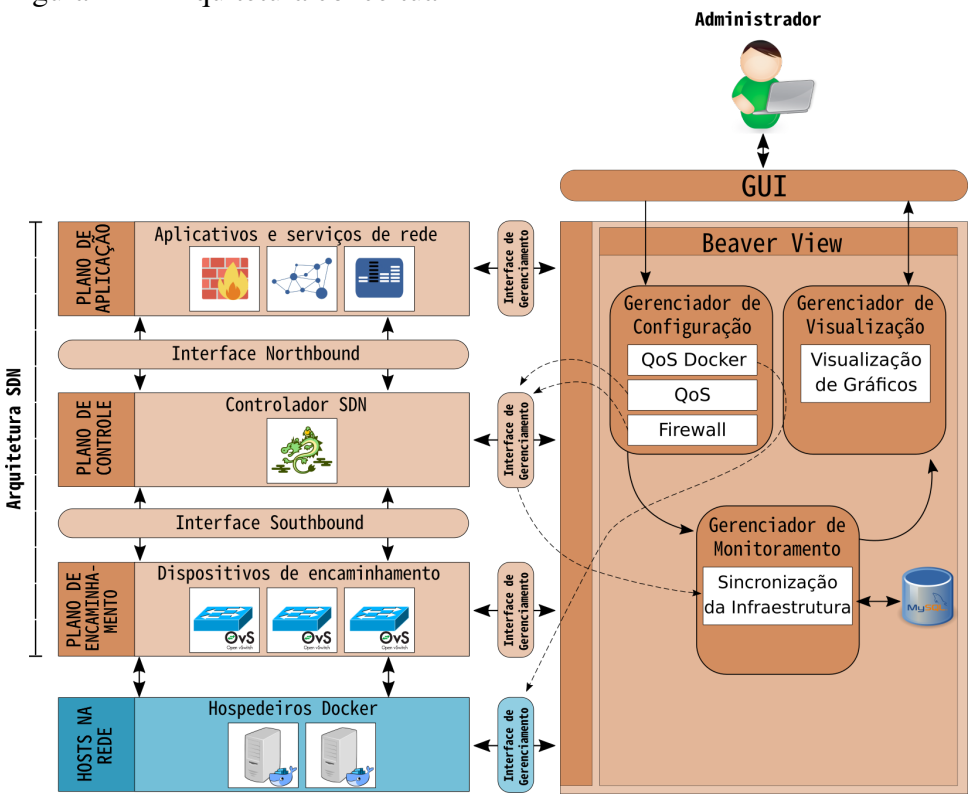
4 SOLUÇÃO PROPOSTA

Neste Capítulo é apresentado a arquitetura e componentes da ferramenta, as funcionalidades implementadas, bem como os requisitos funcionais e não funcionais da solução proposta.

4.1 Arquitetura e componentes

Uma visão geral da arquitetura da ferramenta pode ser observada na Figura 11, nela o administrador de rede tem acesso a uma interface web, na qual é possível gerenciar configurações como de *firewall* e de QoS da infraestrutura, coletar informações tanto dos *switches* quanto dos hospedeiros *Docker* e salvá-las em um banco de dados, além de visualizar essas informações em tempo real.

Figura 11 – Arquitetura conceitual



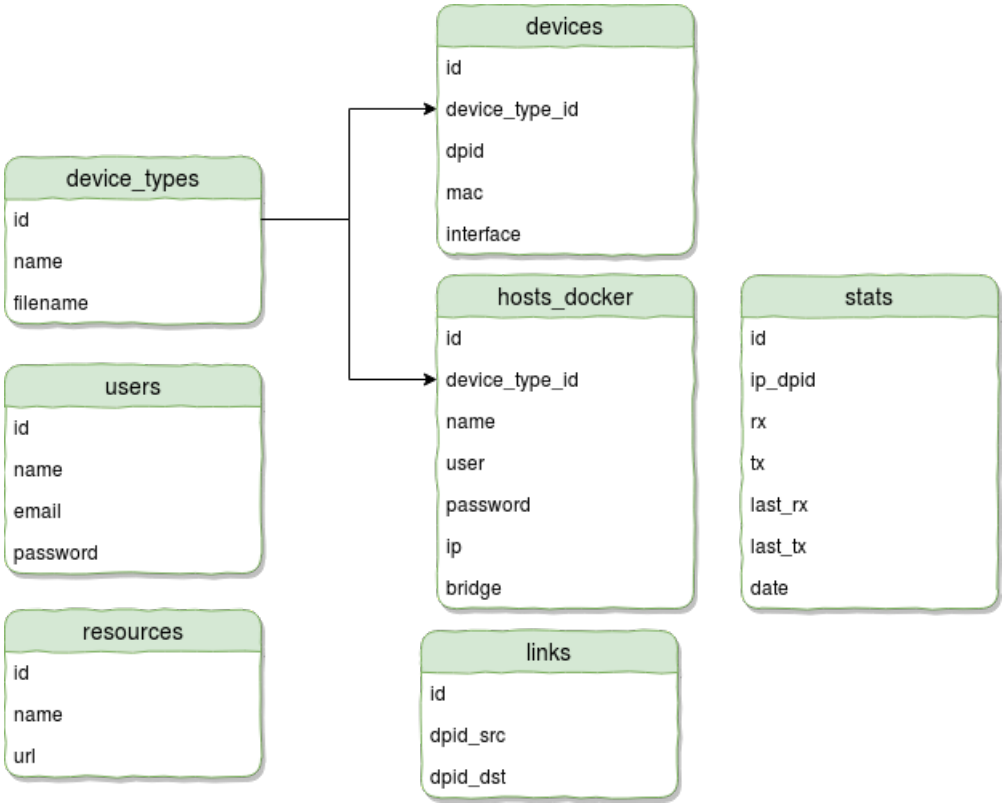
Fonte: O próprio autor.

Os componentes Gerenciador de Configuração, Gerenciador de Visualização e Gerenciador de Monitoramento se comunicam ou com o controlador ou com o hospedeiro *Docker* através de uma interface de gerenciamento. Abaixo são descrito todos os componentes da ferramenta:

- **Interface Web:** aplicação desenvolvida utilizando a linguagem de programação *Python* na versão 2.7 em conjunto com o *microframework Flask*. Esta aplicação integra os componentes de gerenciamento de configuração e visualização, além de acessar o banco de dados, fazendo a leitura dos dados armazenados e exibindo para o usuário.
- **Gerenciador de Configuração:** este componente fornece a capacidade do administrador através da interface web aplicar, visualizar ou remover políticas de QoS e *Firewall*. Ele é responsável por fornecer tais funcionalidades através do consumo da REST API do controlador e REST API desenvolvida neste trabalho.
  .
- **Gerenciador de Visualização:** componente que também faz parte da aplicação *web* sendo responsável pelo acesso ao banco de dados e apresentar as informações coletadas ao administrador de rede.
- **Gerenciador de Monitoramento:** este componente deve estar presente em aplicações do Ryu que implementa os eventos `EventSwitchEnter` e `EventLinkAdd` e um arquivo `agent.py` é disponibilizado para ser executado utilizando o *crontab*, no qual o mesmo coleta dados relacionados a quantidade de *bytes* recebido por cada *switches*.
- **Banco de dados:** é um servidor de banco de dados *MySQL*. Na figura 12 apresentamos um diagrama com o esquema utilizado no banco de dados e nela as seguintes tabelas: (1) *device_types*, que guarda todos os tipos de dispositivos que podem ser adicionados a rede com as seguintes colunas: identificador (*id*), *name* e *filename*. (2) *users*, que guarda as informações de autenticação do administrador da rede à ferramenta, como: identificador (*id*), *name*, *email* e *password*. (3) *resources*, armazena as URLs (endpoints) dos recursos que são usados pela ferramenta para que as funcionalidades como QoS e *firewall* funcione corretamente e possui as seguintes colunas: identificador (*id*), *name* e url. (4) *devices*, guarda os dados dos *switches*, esse procedimento é realizado através do evento `EventSwitchEnter` e possui as seguintes colunas: identificador (*id*), *device_type_id*, dpid, mac e interface. (5) *hosts_docker*, é responsável por armazenar informações do hospedeiro *Docker* que são utilizadas por algumas funcionalidades da ferramenta, que contém as colunas: identificador (*id*),

device_type_id, *name*, *user*, *password*, *ip* e *bridge*. (6) *links* essa tabela guarda todas as conexões entre os *switches* para a criação da topologia, esse procedimento é realizado na detecção de eventos do tipo `EventSwitchEnter` e possui as seguintes colunas: identificador (*id*), *dpid_src* e *dpid_dst*. (7) *stats*, essa tabela armazena informações como a quantidade de *bytes* recebidos e transmitidos dos dispositivos da rede, os dados não são acumulativos, ou seja, se durante o monitoramento a ferramenta coletar 500 *bytes* recebidos por um determinado *switch* e na próxima coletar 600 *bytes* recebidos, será armazenado a diferença que é 100 *bytes*, a tabela possui as seguintes colunas: identificador (*id*), *ip_pdid*, rx, tx, *last*_rx, *last*_tx e *date*.

Figura 12 – Esquema do banco de dados



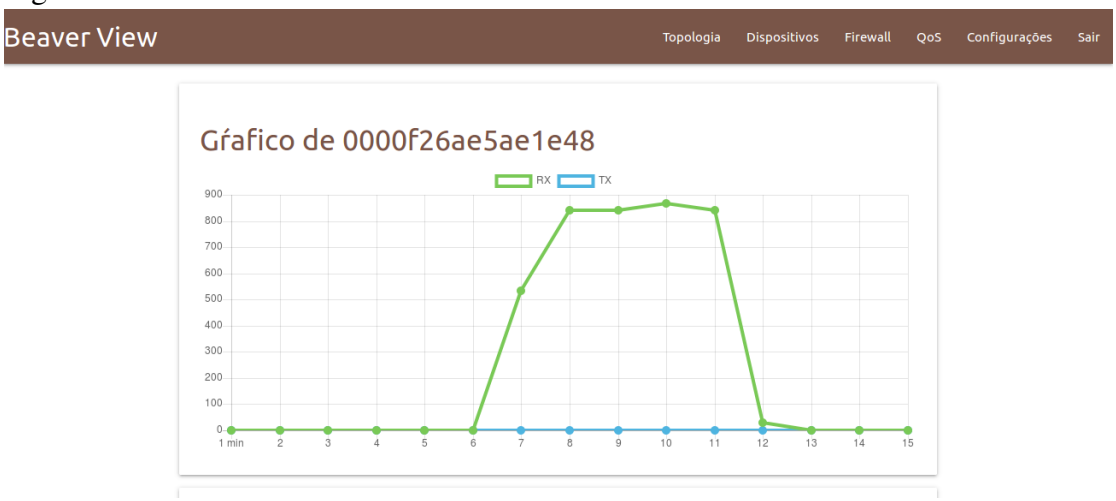
Fonte: O próprio autor.

4.2 Funcionalidades

A Figura 13, apresenta na página inicial disponível ao administrador da rede após o processo de autenticação com gráficos referentes às últimas 15 coletas da quantidade de *bytes* recebidos e transmitidos de todos os *switches* na rede e as opções presentes , entre elas estão:

Topologia, Dispositivos, *Firewall*, QoS e Configurações. Discutimos em detalhes cada opção nas seções seguintes.

Figura 13 – Painel de controle da ferramenta



Fonte: O próprio autor.

4.2.1 Topologia

Para esta função ficar disponível na ferramenta, é necessário implementar dois eventos em uma aplicação Ryu a ser executada. A Figura 14, apresenta a implementação dos eventos `EventSwitchEnter` e `EventLinkAdd`. É importante observar os métodos `register_device` e `register_link`, os mesmos são responsáveis por adicionar todos os *switches* na base de dados e cadastrar os links entre os *switches* respectivamente.

Figura 14 – Implementação do `EventSwitchEnter` e `EventLinkAdd`

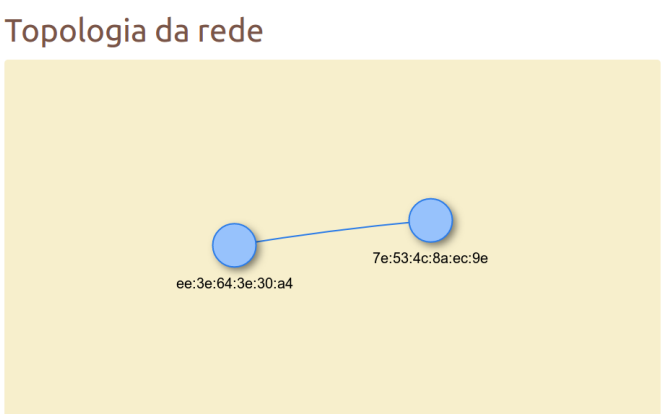
```
@set_ev_cls(event.EventSwitchEnter)
def handler_switch_enter(self, ev):
    print "Switches"
    msg = ev.switch.to_dict()
    register_device(msg, "Switch")

@set_ev_cls(event.EventLinkAdd)
def event_link_add_handler(self, ev):
    print "Links"
    msg = ev.link.to_dict()
    register_link(msg)
```

Fonte: O próprio autor.

O código fonte dos dois métodos citados acima, estão disponíveis no *Github*[1] na pasta `my_app/func_lib` e o arquivo `switch_L2.py` dentro da pasta `my_app` é um exemplo de aplicação Ryu implementando os eventos citado anteriormente.

Com os eventos devidamente implementados, a aplicação deve ser executada passando a opção `--observe-links` habilitando assim a capacidade de observação de eventos de *links*. Com a aplicação em execução, a Figura 15 apresenta o acesso a opção Topologia disponível através do painel de controle da ferramenta. Nela podemos observar o endereço *Media Access Control* (MAC) de cada *switch* na rede e a conexão entre eles.

Figura 15 – Apresentação da topologia da rede pela ferramenta.



Fonte: O próprio autor.

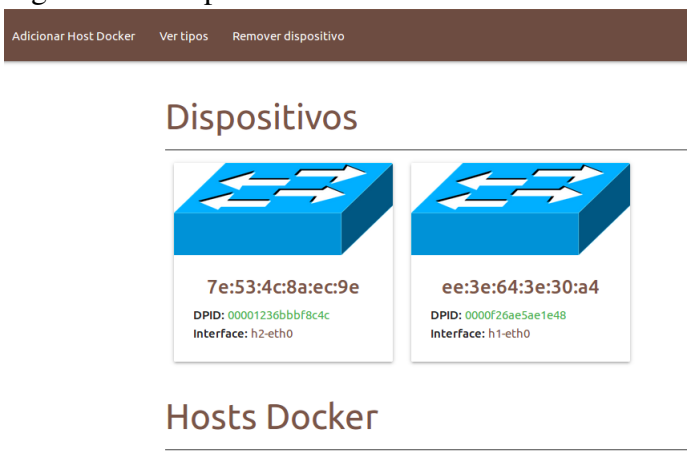
4.2.2 Dispositivos

Nessa função da aplicação, todos os *switches* da rede e todos os hospedeiros *Docker* cadastrados são listados, exibindo as seguintes informações: MAC ou nome do hospedeiro *Docker*, DPID ou IP do hospedeiro *Docker* e a interface do *switch* ou a *bridge* do *Docker*. Na Figura 16, podemos observar a página divida em ***Hosts Docker*** para listar todos os hospedeiros *Docker* cadastrados e **Dispositivos** para listar todos os *switches* da rede, que no caso são listados dois *switches*. Além disso, outro menu secundário é exibido contendo três funções: Adicionar *Hosts Docker*, Ver tipos, Remover dispositivos.

Na função **Adicionar *Hosts Docker***, o administrador de rede pode cadastrar os hospedeiros *Docker* da rede através de um formulário como podemos observar na Figura 17, o formulário possui os campos:

[1] https://github.com/JardelGoncalves/ryu

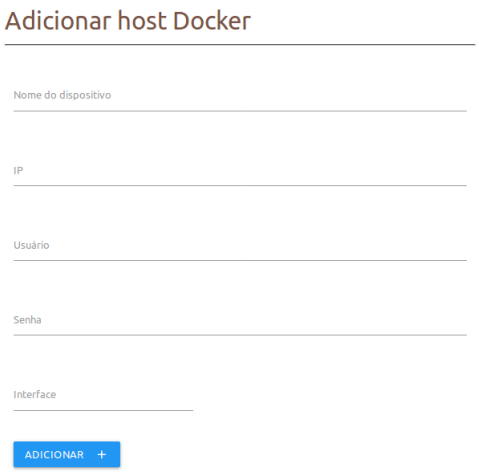
Figura 16 – Dispositivos da rede.



Fonte: O próprio autor.

- **Nome do dispositivo**: É o nome que será exibido quando listado na página Dispositivo.
- **IP**: Referente a máquina que possui o *Docker* instalado.
- **Usuário**: Deve ser informado um usuário da máquina alvo que possui privilégios de super usuário.
- **Senha**: Referente ao usuário informado anteriormente.
- **Interface**: A mesma *bridge* do *Docker* conectada ao OVS.

Figura 17 – Adicionar hospedeiro *Docker*.

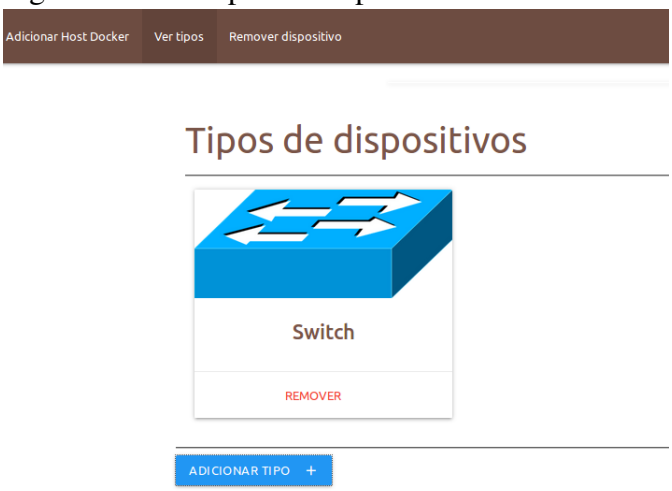


Fonte: O próprio autor.

Na função **Ver Tipos** são listados todos os tipos de dispositivos cadastrados pelo administrador de rede contendo a imagem (com o intuito de facilitar a identificação do tipo de dispositivo na página Dispositivos) e o nome do tipo de dispositivo. Na Figura 18, podemos observar todos os tipos de dispositivos cadastrados, por padrão, o tipo *Switch* é cadastrado durante

a instalação da ferramenta e um botão REMOVER abaixo de cada tipo que tem a funcionalidade de remover aquele determinado tipo de dispositivo.

Figura 18 – Ver tipos de dispositivos.

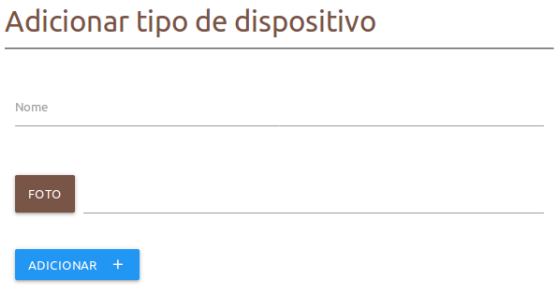


Fonte: O próprio autor.

Ainda na Figura 18, podemos observar que na página também possui um botão ADICIONAR TIPO que redireciona para outra página e permite cadastrar tipos de dispositivos. A sequência dessa ação está na Figura 19, o formulário para cadastro de tipos de dispositivos com todos os campos obrigatórios, são eles:

- **Nome**: Referente ao nome do tipo de dispositivo (e.g. Roteador)
- **FOTO**: Campo do formulário que permite adicionar um arquivo de foto.

Figura 19 – Adicionar Tipo de dispositivo.



Fonte: O próprio autor.

Já a função **Remover Dispositivo**, disponível no menu secundário da página Dispositivos, todos os dispositivos cadastrados na base de dados são listados na página com seu MAC ou nome do hospedeiro *Docker* e com um botão REMOVER abaixo que permite ao administrador

de rede remova qualquer dispositivo. Na Figura 20, podemos observar todos os dispositivos com um botão REMOVER que ao ser clicado o dispositivo é removido da base de dados.

Figura 20 – Remover dispositivo.

Fonte: O próprio autor.

4.2.3 Firewall

Nessa função da aplicação, permite que o administrador interaja com os *switches* da rede, adicionando ou removendo politicas de *firewall*, seja ela de bloqueio ou de permissão. Porém, ao executar uma aplicação de *firewall* com o controlador Ryu, por padrão todos os *switches* são desabilitados interrompendo a comunicação entre eles, a Figura 21 apresenta tal situação.

Figura 21 – Página referente ao *Firewall*.

Fonte: O próprio autor.

Ao clicar no botão HABILITAR *SWITCHES* como ilustrado na figura acima, todos

os *switches* desabilitados alteram seus *status* para habilitado e a página começa a exibir uma tabela. Como pode ser observado na Figura 22, essa tabela se refere a todas as políticas aplicadas em qualquer *switch* da rede que pode ser identificado pela coluna DPID. A tabela possui também as seguintes colunas: Origem, Destino, Protocolo e Ação. Quando uma regra é aplicada, um botão vermelho com um ícone de lixeira é exibido em cada política e ao ser clicado o mesmo remove a política selecionada.

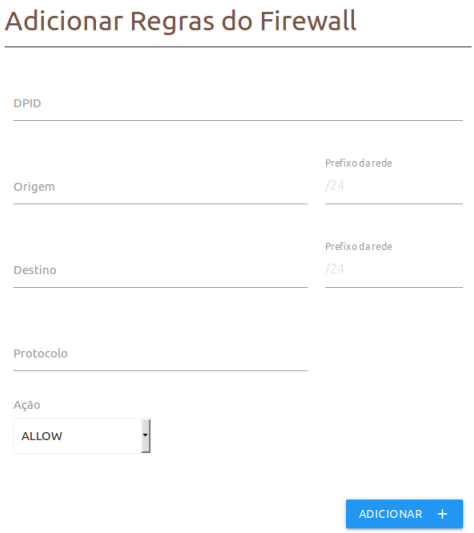
Figura 22 – Tabela com as regras de *firewall*.

Fonte: O próprio autor.

Além disso, para adicionar regras, a página de *Firewall* possui a função **Adicionar Regras** em um menu secundário que redireciona para uma página contendo um formulário para adicionar políticas. Esse formulário pode ser observado na Figura 23, que contém os seguintes campos:

- **DPID**: Referente a identificação do *switch* (este dado pode ser obtido na página de Dispositivos) onde a regra será aplicada.
- **Origem**: Este campo não é obrigatório e é referente a um IP da rede de origem (caso não seja informado, é considerado um IP qualquer).
- **Destino**: Este campo não é obrigatório e é referente a um IP da rede de destino (caso não seja informado, é considerado um IP qualquer).
- **Prefixo de rede**: Esses dois campos são referentes ao prefixo da rede de Origem e Destino respectivamente e também não é um campo obrigatório.
- **Protocolo**: Neste campo é necessário que o administrador informe um protocolo da camada de aplicação ou da camada de transporte.
- **Ação**: Este campo refere-se o comportamento da regra a ser aplicada, o administrador possui a opção *ALLOW* que permite o fluxo de tráfego ou *DENY* que nega a passagem daquele tipo de fluxo.

Figura 23 – Formulário para adicionar regras de *firewall*.



Fonte: O próprio autor.

4.2.4 QoS

Nesta seção, serão apresentadas duas abordagens de QoS implementadas durante o desenvolvimento da ferramenta, porém, não foi possível validar a funcionalidade que utiliza os recursos do controlador durante os testes. Mesmo assim apresentamos os erros encontrados, pois tais dados possuem um valor agregado para novos trabalhos que venham surgir sobre a área.

Na primeira funcionalidade de QoS, o administrador de rede pode por meio do controlador da rede adicionar configurações e políticas de filas para reservar largura de banda da rede através de operações de QoS por par de fluxo. O administrador da rede irá criar uma política individual para cada conexão entre dois contêineres na rede.

Na Figura 24, podemos observar a página quando acessamos a funcionalidade de QoS que contém um menu secundário com as funções de Configurar Filas, Adicionar Regras, QoS *Hosts Docker* e OVSDB. Na figura também observamos uma mensagem de aviso informando que é necessário definir uma determinada variável no controlador (que será discutido posteriormente) e uma tabela usada para exibir todas as políticas definidas com as seguintes informações:

- **DPID**: identificador do *switch* e em qual *switch* foi aplicado a política.
- **Prio**: Prioridade da política (política com o menor valor, possui a maior prioridade).
- **Proto**: É o tipo do protocolo do determinado fluxo que o administrador pretende dar prioridade ou não.
- **IP dst** e **Port dst**: Essas informações são referentes ao contêiner destino, cujo a

combinação de certas informações o fluxo terá prioridade ou não no *switch*.

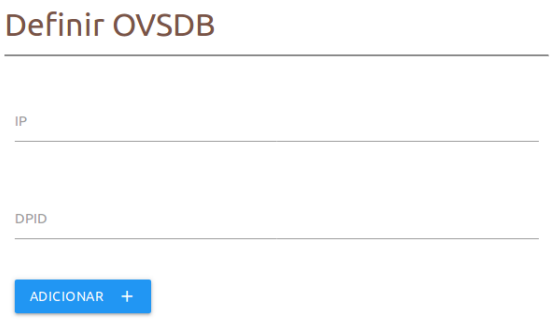
- **IP src** e **Port src**: É o contêiner fonte, essa informação não necessariamente precisa ser preenchida, ou seja, qualquer IP e Porta pode ser a fonte (transmite os dados).

Figura 24 – Página QoS.

Fonte: O próprio autor.

A função **OVSDB** é o primeiro passo para usar a funcionalidade de QoS por par de fluxo na qual precisamos configurar no controlador a variável **ovsdb_addr** para cada *switch*. Ao clicar, o administrador de rede irá ser redirecionado para uma nova página que contém um formulário com dois campos, um para o identificador do *switch* (DPID) e outro para o IP do máquina que possui aquele *switch* instalado, como podemos observar na Figura 25.

Figura 25 – Formulário para configurar ovsdb dos *switches*



Fonte: O próprio autor.

Feita tal configuração, o próximo passo é acessar a função de **Configurar Filas**, que ao clicar, uma nova página é exibida ao administrador de rede que serão listadas em uma tabela todos os *switches* e suas respectivas portas configuradas. Por padrão nenhum *switch* é configurado. Essa informações podem ser observadas na Figura 26.

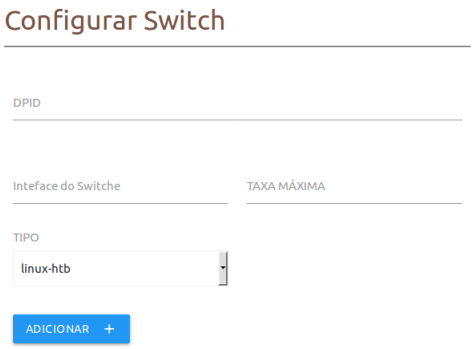
Para configurar um *switch*, na figura anterior é apresentado um botão para adicionar tal configuração, ao clicar uma nova página é exibida com um formulário com campos para

Figura 26 – *Switches* configurados.

Fonte: O próprio autor.

informar o *switch*, qual porta do *switch*, uma taxa máxima para a classe pai que serão usadas pelas classes filhas para controlar a taxa de transmissão e um campo para o tipo de algoritmo com a única opção *linux-htb*, como podemos observar na Figura 27.

Figura 27 – Formulário para configurar *switches*.



Fonte: O próprio autor.

Após adicionar a configuração de um *switch*, ela estará disponível na página da função de Configurar Filas. Dois botões aparecem para cada configuração adicionada, um para remover a configuração aplicada ao *switch* e outro botão para visualizar todas as filas (classes filhas) definidas na configuração realizada no *switch* como podemos observar na Figura 28.

Ao clicarmos no botão verde ilustrado na figura anterior, o administrador será redirecionado a uma nova página apresentada na Figura 29, que lista todas as filas definidas (caso haja) para aquela classe pai com as seguintes informações:

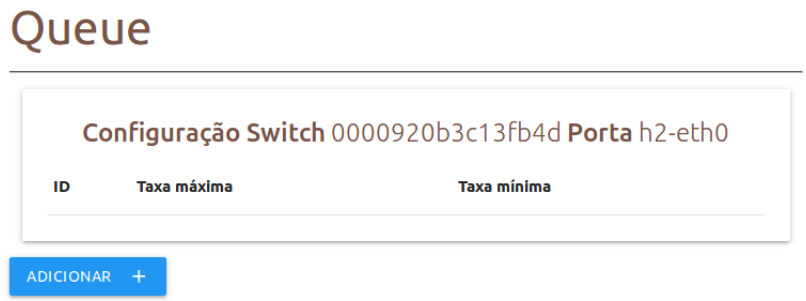
- **DPID**: identifica o *switch* em que aquelas filas foram definidas.
- **Taxa máxima**: É a taxa máxima de transmissão para aquela classe, essa informação pode ser omitida para casos que não necessite limitar a taxa máxima para uma fila.

Figura 28 – Listando *switches* configurados.

Fonte: O próprio autor.

- **Taxa mínima**: É a taxa mínima garantida para uma classe, essa informação pode ser omitida para casos que não necessite garantir uma taxa mínima para uma fila.

Figura 29 – Filas definidas a um determinado *switch*.



Fonte: O próprio autor.

Para adicionar ou atualizar as filas definidas em uma determinada classe configurada em um *switch*, o botão adicionar ilustrado na figura anterior redireciona para outra página que permite definir varias filas como podemos observar na Figura 30, um formulário com os campo para definir a taxa máxima e a taxa miníma, o botão verde permite o administrador de rede definir varias filas e podem ser visualizadas em uma tabela abaixo do botão de finalizar.

Com todas as configurações realizadas anteriormente, a funcionalidade **Adicionar Regras** permite o administrador aplicar políticas de QoS em um determinado *switch*. Na Figura 31, é apresentado o formulário da página para aplicar políticas de QoS com os seguintes campos:

- **Endereço de Origem** e **Porta de Origem**: São informações de um contêiner que inicia a comunicação e que podem serem filtrados.
- **Endereço de Destino** e **Porta de destino**: São informações do contêiner destino, ou seja, para onde os fluxos foram endereçados.
- **DPID**: identifica o *switch* em que a regra será aplicada.

Figura 30 – Formulário para definir filas.

Fonte: O próprio autor.

- **Fila**: Em qual fila deverá ser encaminhado o fluxo que combine com as informações inseridas no formulário, essas filas foram cadastrada anteriormente.
- **Protocolo**: Podem ser escolhido um entre três tipos de protocolos, UDP, TCP ou ICMP.

Figura 31 – Formulário para aplicar políticas de QoS.

Fonte: O próprio autor.

Como citado anteriormente, foi desenvolvido duas abordagens de QoS, a primeira utilizando operações de QoS por par de fluxo, aplicando políticas por meio do controlador e a segunda abordagem que apresentaremos agora é aplicando disciplinas de fila TBF, não é o nosso foco aprofundar sobre TBF, mas apresentaremos como funciona o algoritmo **Token Bucket**.

O algoritmo de *Token Bucket* é usado para controlar a transmissão de fluxo de dados

em uma rede. Segundo Wagner (2001), os *tokens* são gerados em uma determinada taxa e o balde possui uma capacidade máxima. Cada *token* representa permissão para enviar um determinado número de *bits* à rede. Para transmitir um pacote, um número de *tokens* correspondente ao tamanho do pacote deve ser removido do balde, caso não haja *tokens* suficientes o pacote é descartado.

Para implementar essa abordagem TBF, desenvolvemos uma *REST API* que estar disponível no *Github*[1] para facilitar a aplicação de políticas através do *Traffic Control* (TC) do *linux* e é necessário que a mesma esteja em execução para que esta funcionalidade possa funcionar corretamente.

A função **QoS *Host Docker***, permite que o administrador de rede aplique por meio de um formulário disciplinas de filas TBF nos contêineres. A Figura 32, apresenta a página contendo uma tabela que lista todas as políticas aplicadas nos contêineres (caso haja) com as seguintes informações:

- **IP Host**: Endereço IP do hospedeiro *Docker*.
- **IP contêiner**: IP do contêiner em que a política foi aplicada.
- **Taxa**: Taxa média atribuída a classe.
- ***Burst***: Tamanho do balde.
- **Latência**: Tempo máximo que um pacote pode ficar na fila esperando uma ficha.
- ***Peakrate***: Taxa de pico máxima das rajadas.
- ***Minburst***: Quantidade mínimas de *bytes* contabilizada por pacote (em geral, é comum utiliza o MTU de um quadro *Ethernet*).

Figura 32 – Lista de políticas aplicadas nos contêineres.

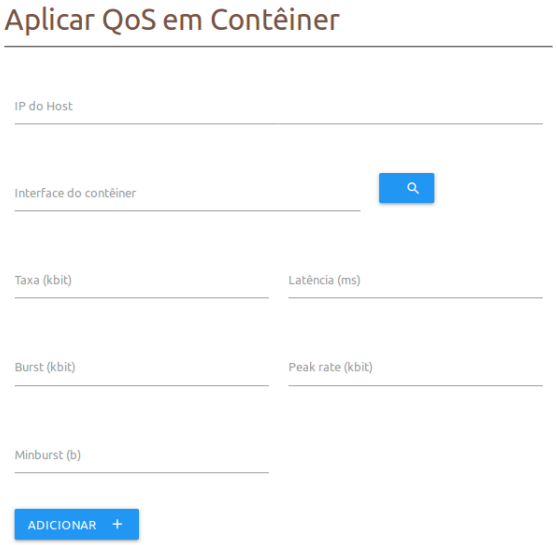
Fonte: O próprio autor.

Ao clicar no botão ADICIONAR, ilustrado na figura anteriormente, o administrador de rede será redirecionado a outra página contendo o formulário para aplicação das políticas

[1] https://github.com/JardelGoncalves/TBF-Docker-RESTFul-API

que pode ser observado na Figura 33, quase todos os campos do formulário foram explicados anteriormente, exceto o campo Interface do contêiner, que deve ser inserido a interface criada ao lançar um contêiner.

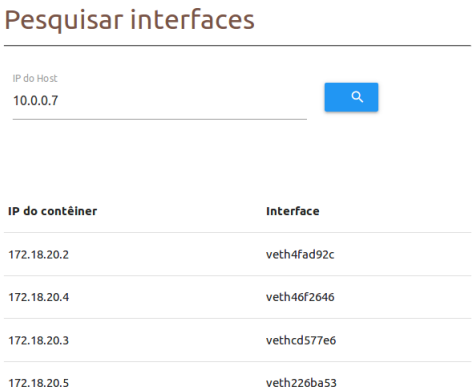
Figura 33 – Formulário para aplicar políticas de QoS.



Fonte: O próprio autor.

Porém, existe uma grande dificuldade de associar uma interface (com o prefixo *veth*) a um contêiner quando se tem mais de um contêiner em execução no hospedeiro *Docker*, para resolver esse problema o administrador de rede pode clicar no botão azul que contém um ícone de lupa que abrirá uma nova aba no navegador onde pode ser consultado todos os contêineres e suas respectivas interface de um determinado hospedeiro *Docker*, como ilustrado na Figura 34.

Figura 34 – Consultar interfaces.



Fonte: O próprio autor.

Tendo todos os campos devidamente preenchidos do formulário apresentado na Figura 33, ao clicar em ADICIONAR, a política será aplicada na interface do contêiner escolhido.

4.2.5 Configurações

Nessa funcionalidade, permite que o administrador de rede possa configurar recursos da *REST API* do controlador como QoS, *Firewall* e o recurso que permite coletar estatística dos *switches* ou adicionar recursos caso o administrador desenvolva novas funcionalidades.

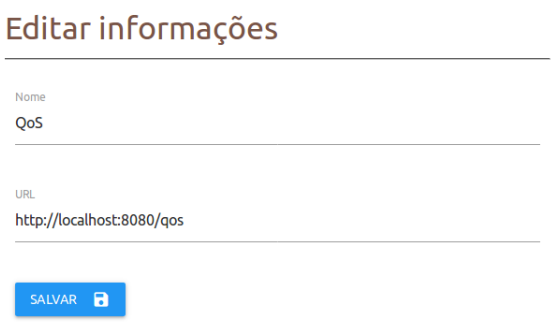
A Figura 35, apresenta uma tabela com todos os recursos cadastrados, tais recursos são cadastrados durante a instalação. Além disso, para cada recurso cadastrado, é possível remove-lo (botão vermelho) ou editá-lo (botão amarelo).

Figura 35 – Listando os recursos cadastrados.

Fonte: O próprio autor.

Caso o administrador queira modificar a URL do recurso, ao clicar no botão amarelo do recurso, uma nova página será exibida contendo as informações daquele do mesmo, permitindo a alteração das informações como podemos observar na Figura 36.

Figura 36 – Editar informações de um recurso.

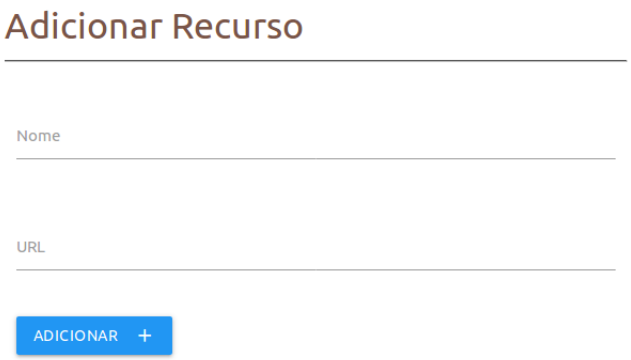


Fonte: O próprio autor.

Para caso em que o administrador desenvolve uma nova funcionalidade e queria adicionar a URL do recurso, na página de Configurações aparece um menu secundário contendo

a função de **Adicionar Recurso**. Essa função permite ao administrador adicionar um novo recurso como podemos observar na Figura 37, contendo um simples formulário com os campos **Nome** e **URL**.

Figura 37 – Formulário para adicionar recurso.



Fonte: O próprio autor.

4.3 Requisitos

Buscando demonstrar as tarefas e serviços oferecidos ou não pela ferramenta, foram definidos os requisitos funcionais e não funcionais da solução desenvolvida, que são:

- Requisitos funcionais:
  a) O administrador da rede deve poder visualizar todas as políticas de *firewall* aplicadas na rede.
  b) O administrador da rede deve poder adicionar ou remover qualquer política de *firewall* em qualquer comutador virtual.
  c) O administrador da rede deve poder adicionar ou remover informações dos hospedeiros *Docker*.
  d) O administrador da rede deve poder adicionar ou remover tipos de dispositivos.
  e) O administrador da rede deve poder visualizar todas as políticas de QoS aplicadas na rede ou nos contêineres.
  f) O administrador da rede deve poder adicionar ou remover políticas de QoS aplicadas na rede ou nos contêineres.
  g) O administrador da rede deve poder visualizar todos os recursos cadastrados na base de dados.

h) O administrador da rede deve poder adicionar, editar ou remover recursos da base de dados.

- Requisitos não funcionais:
  a) A ferramenta deve fornecer um sistema de autenticação visando assegurar que o usuário de fato é o administrador da rede.
  b) A ferramenta deve apresentar todas as exceções que possam ocorrer durante a execução de uma funcionalidade.
  c) A ferramenta deve atualizar os dados de *bytes* recebidos e transmitidos na página inicial a cada 30 segundos.

5 AVALIAÇÃO DA FERRAMENTA

Este capítulo descreve como foram executados os testes para avaliação da ferramenta desenvolvida e os resultados obtidos com eles.

5.1 Metodologia de avaliação

Para a realização dos testes, utilizamos o *Virtualbox*[1] para instanciar 3 maquinas com 1 CPU virtual, 1 GB de memória RAM e 8 GB de disco com sistema operacional *ubuntu 16.04.3 LTS*. Em duas das três máquinas (os hospedeiros *Docker*) foram instalados o *Open vSwitch*[2] na versão 2.5.5, Docker[3] na versão 18.06.1-CE e o utilitário brctl na versão 1.5.

Foram utilizados dois cenários de testes, em ambos a quantidade de hospedeiros *Docker* eram equivalentes, em todos os cenários também tinha uma máquina executando o controlador RYU e a ferramenta, a única diferença foi a funcionalidade testada.

No primeiro cenário, um contêiner foi instanciado em cada hospedeiro *Docker*, em seguida no controlador foi executado uma aplicação de *firewall*. O contêiner instanciado no hospedeiro 2 executou um *ping* com destino contêiner do hospedeiro 1, por padrão qualquer tipo de tráfego é bloqueado entre os comutadores virtuais que interligam os hospedeiros *Docker*, depois de uns segundos aplicamos políticas que permitissem o tráfego ICMP na rede.

No segundo cenário, foi instanciado um contêiner que realizou o *download* de um arquivo da internet, durante o processo de obtenção do arquivo aplicamos uma disciplina de fila a fim de observar a mudança de comportamento da taxa de transmissão.

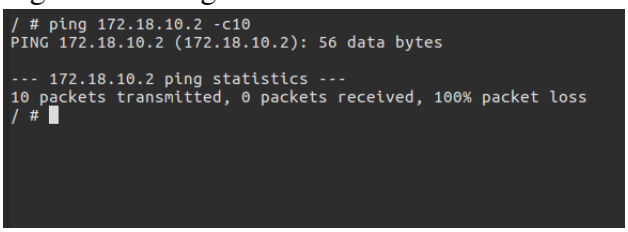
5.2 Resultados

No primeiro cenário, como citado anteriormente, após executar a aplicação de *Firewall* no controlador, toda a comunicação entre os comutadores são desabilitadas e interrompidas, habilitamos a comunicação e executamos um *ping* do contêiner do hospedeiro 2 (IP 172.18.20.2/24) com destino o contêiner do hospedeiro 2 (IP 172.18.10.2/24) como podemos observar na Figura 38.

Mesmo com a comunicação habilitada todo o tráfego na rede é bloqueado, para contornar tal situação, é necessário aplicar políticas para permitir o tráfego na rede. Na Figura 39,

[1] https://www.virtualbox.org/
[2] https://www.openvswitch.org/
[3] https://www.docker.com/

Figura 38 – Ping entre os contêineres



Fonte: O próprio autor.

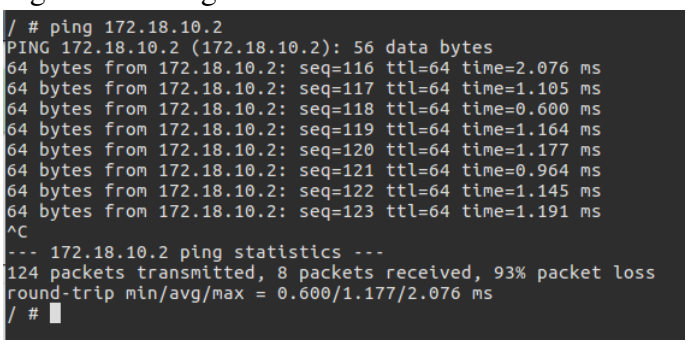
apresenta as políticas aplicadas em nosso experimento. Nela, é possível observar que para ambos dos comutadores virtuais na rede, permite tráfego ICMP para qualquer endereço de Origem e Destino.

Figura 39 – Políticas de *Firewall* aplicadas nos comutadores.

Fonte: O próprio autor.

Após aplicar as políticas, a Figura 40 apresenta o *ping* funcionando entre os contêineres, além disso, podemos observar que a quantidade de *ping* entre as máquinas não foram limitadas e que a política foi aplicada enquanto o *ping* também executava.

Figura 40 – Ping funcionando entre os contêineres



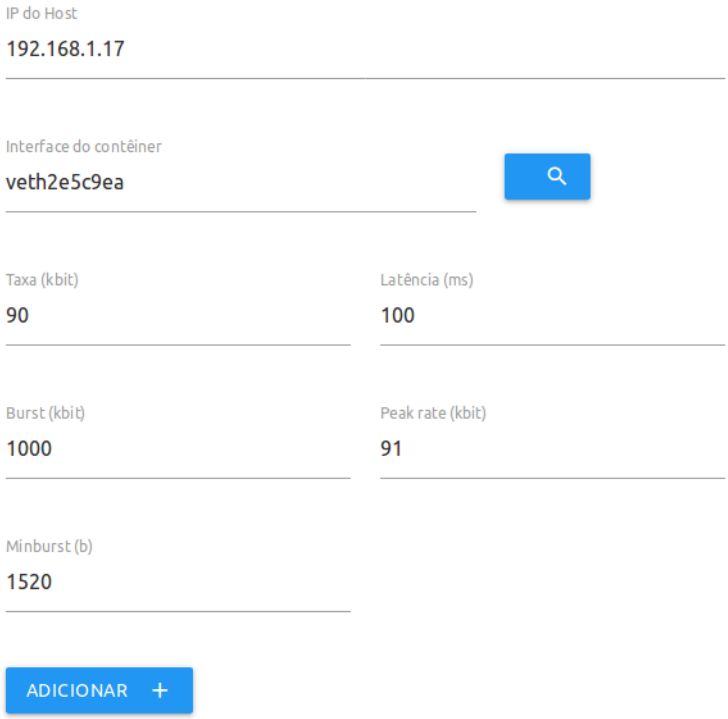
Fonte: O próprio autor.

No segundo cenário, inicialmente propomos uma funcionalidade que usa operações de QoS por par de fluxo, um recurso fornecido pela REST do controlador. Porém, a mesma

apresentou alguns problemas durante a validação, usamos o *iperf*[1] para tentar validar a ferramenta, por algum motivo desconhecido, as estatística da rede era exibidas como *Not a Number* (NaN). Tentamos também executar *wget* para obter um arquivo hospedado em outro contêiner, entretanto, as requisições chegavam ao contêiner, mas nenhuma resposta era retornada.

Devido a esses problemas, nós modificamos a nossa abordagem e decidimos desenvolver uma solução para limitar a taxa de transmissão dos contêineres. Para isso os hospedeiros *Docker* devem estar executando uma *REST API* (citada na subseção 4.2.4). A Figura 41, apresenta a política aplicada no contêiner no experimento realizado em que limitamos a taxa de transmissão para 90*kbit/s* (cerca de 11.25*KB/s*), limitamos a taxa de pico máxima das rajadas para 91*kbit/s*, o tempo de espera dos pacotes em 100*ms*, o tamanho do balde para 1000*kbit* e a quantidade miníma de 1520*bytes* contabilizada por pacote.

Figura 41 – Política de QoS aplicada no contêiner



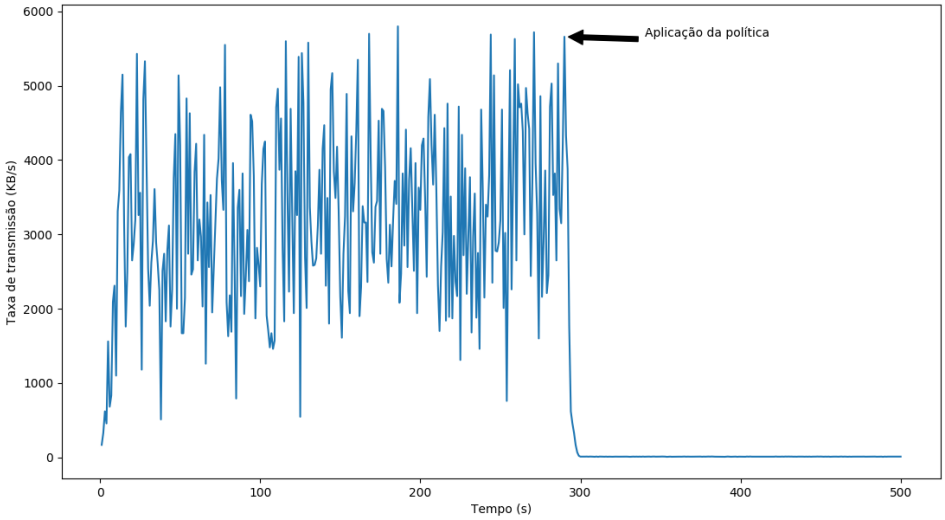
Fonte: O próprio autor.

Na Figura 42, apresenta os resultados do testes do segundo cenário. Nela, podemos observar que antes de executar a política (momento esse indicado pela seta) a taxa de transmissão varia entre um pouco mais de 165 KB/s á quase 6000 KB/s.

A Figura 43, é a mesma citada anteriormente, porém apresenta os últimos 200 segundos do teste realizado. A fim de facilitar o entendimento, a linha vermelha representa o

[1] https://iperf.fr/

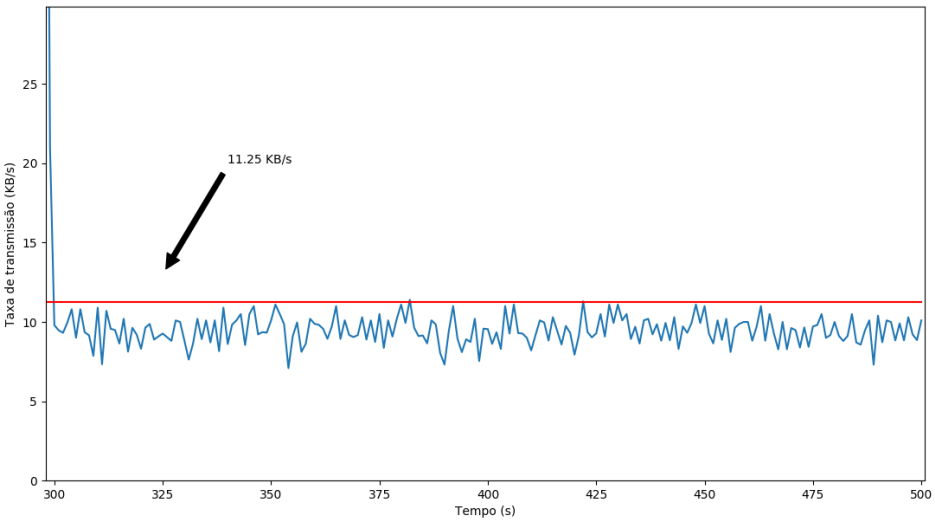
Figura 42 – Taxa de transmissão durante 500 segundos



Fonte: O próprio autor.

limite da taxa de transmissão aplicada no contêiner. Podemos observar que apenas em alguns momentos a taxa de transmissão ultrapassou o limite, mas logo em seguida a taxa de transmissão caiu respeitando assim o limite aplicado a mesma.

Figura 43 – Taxa de transmissão entre 300 e 500 segundos.



Fonte: O próprio autor.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve por objetivo o desenvolvimento de uma ferramenta que auxilia os administradores de redes no gerenciamento de infraestrutura virtual *Docker* baseada em SDN. Para isso, primeiro foi realizado um levantamento das soluções de gerenciamento de infraestruturas virtuais e percebemos que as soluções não abordavam o gerenciamento de infraestruturas *Docker* baseado em SDN, em seguida foi elaborado uma lista com as funcionalidades para assim iniciar o processo de implementação da solução. O código da ferramenta desenvolvida está disponível para a comunidade em geral no repositório no *Github*[1].

Através dos experimentos feitos nos cenários criados e os resultados obtidos, foi possível validar as funcionalidades de *Firewall* e QoS baseado em disciplinas TBF. Apesar dos testes terem ocorrido com sucesso, a versão atual da ferramenta implementa apenas os módulos/funcionalidades citados anteriormente, tornando-a simplificada.

Como trabalhos futuros, fica a sugestão da implementação de novos módulos na ferramenta, como módulo de roteamento, gráficos sobre outros dados da rede como taxa de erros, detecção de gargalos e sistema de notificação através de *e-mails* ou redes sociais que irão ajudar o administrador de rede em outras tomadas de decisões.

[1] https://github.com/JardelGoncalves/Beaver-View

REFERÊNCIAS

ADAMI, D.; DONATINI, L.; GIORDANO, S.; PAGANO, M. A network control application enabling software-defined quality of service. In: IEEE. **Communications (ICC), 2015 IEEE International Conference on**. [S.l.], 2015. p. 6074–6079.

COSTA, L. R. **Openflow e o paradigma de redes definidas por software**. [S.l.: s.n.], 2013.

DUA, R.; KOHLI, V.; KONDURI, S. K. **Learning Docker Networking**. [S.l.]: Packt Publishing Ltd, 2016.

GOMES, C. S.; SILVA, F. S. D.; NETO, E. P.; COSTA, K. B.; SILVA, J. B. da. Towards a modular interactive management approach for sdn infrastructure orchestration. In: IEEE. **Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on**. [S.l.], 2016. p. 1–6.

GUEDES, D.; VIEIRA, L. F. M.; VIEIRA, M.; RODRIGUES, H.; NUNES, R. V. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC**, v. 30, n. 4, p. 160–210, 2012.

HAKIRI, A.; GOKHALE, A.; BERTHOU, P.; SCHMIDT, D. C.; GAYRAUD, T. Software-defined networking: Challenges and research opportunities for future internet. **Computer Networks**, Elsevier, v. 75, p. 453–471, 2014.

ISOLANI, P. H.; WICKBOLDT, J. A.; BOTH, C. B.; ROCHOL, J.; GRANVILLE, L. Z. Interactive monitoring, visualization, and configuration of openflow-based sdn. In: IEEE. **Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on**. [S.l.], 2015. p. 207–215.

KREUTZ, D.; RAMOS, F. M.; VERISSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, Ieee, v. 103, n. 1, p. 14–76, 2015.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 2, p. 69–74, 2008.

PFAFF, B.; PETTIT, J.; AMIDON, K.; CASADO, M.; KOPONEN, T.; SHENKER, S. Extending networking into the virtualization layer. In: **Hotnets**. [S.l.: s.n.], 2009.

ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; MAGALHÃES, M. F. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia, Campinas**, v. 7, n. 1, p. 65–76, 2010.

RYU, S. **Framework Community: Ryu SDN Framework**. [S.l.: s.n.], 2015.

SHARMA, P.; BANERJEE, S.; TANDEL, S.; AGUIAR, R.; AMORIM, R.; PINHEIRO, D. Enhancing network management frameworks with sdn-like control. In: IEEE. **Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on**. [S.l.], 2013. p. 688–691.

TURNBULL, J. **The docker book**. [S.l.]: Lulu. com, 2014.

WAGNER, K. Short evaluation of linux's token-bucket-filter (tbf) queuing discipline. **http://www. docum. org/stef. coene/qos/docs/other/tbf02_kw. ps**, 2001.