Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

Uma ferramenta para controle centralizado de *switches*: Utilizando redes definidas por software em escritórios

Discente: João Paulo Silva de Luna (jpsl2@cin.ufpe.br)

Orientador: José Augusto Suruagy Monteiro (suruagy@cin.ufpe.br)

Recife, Setembro de 2018

Universidade Federal de Pernambuco

Centro de Informática

Uma ferramenta para controle centralizado de *switches*: Utilizando redes definidas por software em escritórios

Trabalho de graduação apresentado ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em em Ciência da Computação

Discente: João Paulo Silva de Luna (jpsl2@cin.ufpe.br)

Orientador: José Augusto Suruagy Monteiro (suruagy@cin.ufpe.br)

Recife, Setembro de 2018

RESUMO

O conceito de redes locais virtuais (VLANs) traz uma solução para o problema de segmentar redes, reduzindo assim o domínio de *broadcast* e evitando quedas de desempenho. A vantagem sobre uma segmentação física se dá ao não trazer novos problemas como a subutilização de *switches* e necessidade constante de alterações físicas na rede, se tornando uma prática constante nas infraestruturas de rede atuais. Em contrapartida cria uma tarefa, importante e repetitiva, de manter as configurações de todas as VLANs consistentes em todos os equipamentos de rede. Paralelamente o conceito de redes definidas por software separa o plano responsável por definir o comportamento da rede do plano responsável por direcionar os pacotes, permitindo que o controle seja centralizado. Neste trabalho propomos e implementamos uma ferramenta compatível com ambas tecnologias, permitindo uma migração gradual da infraestrutura legada para uma nova, com maior visibilidade e simplicidade de configuração. Além de diminuir a curva de aprendizagem inicial para a implantação de uma rede empresarial segmentada.

SUMÁRIO

1 **INTRODUÇÃO**

Redes institucionais costumam ser hierárquicas, com cada departamento ou grupo de trabalho tendo sua própria rede local, conectada a outras redes locais por uma hierarquia de *switches* (Kurose, 2013). Um dos benefícios desta segmentação é a restrição do domínio de *broadcast* da rede, pois em segmentos de rede com muitos dispositivos este tráfego reduz o desempenho da rede. A segmentação também traz um aumento de segurança, restringindo o alcance de alguns tipos de ataque e facilitando a prevenção contra os mesmos (Yu, 2011). Algumas das desvantagens deste tipo de segmentação poderia ser o uso ineficiente de *switches*, caso um *switch* de noventa e seis portas seja alocado para um departamento com dez pessoas, e a gerência de usuários nas redes, pois cada movimentação do usuários entre os departamentos, acarretaria em uma movimentação física na rede (Kurose, 2013). Em redes menores, muitas vezes pode parecer tentador manter uma única rede local para fugir destas desvantagens.

Neste cenário, o conceito de VLANs (Virtual Local Area Network) nos permite ter diversas redes locais virtuais em uma única rede local física, trazendo os benefícios da segmentação sem as desvantagens citadas (Kurose, 2013).

Em uma infraestrutura tradicional, a configuração de cada rede virtual deve ser feita em cada *switch*, trazendo para o administrador da rede a responsabilidade de manter esta configuração consistente em todos os switches, sendo esta uma tarefa repetitiva e suscetível a erros. Observando isto tivemos alguns trabalhos como o de Krothapalli (2009), que propôs uma ferramenta para automatizar o processo e permitir uma visualização da configuração. E Nadeau e Gray (2013) propuseram o uso de ferramentas de configuração centralizada como Puppet, Cheff e outros para a mesma automatização.

No paper Software-Defined Networking: A Comprehensive Survey, Diego Kreutz et al. comenta sobre a complexidade das redes IP tradicionais e da dificuldade de gerenciar as mesmas, trazendo as redes definidas por software (SDNs) como uma alternativa que separa a lógica de controle da rede dos dispositivos responsáveis pelo encaminhamento do tráfego. Permitindo que a lógica de controle seja implementada em um controlador centralizado.

Observando as vantagens trazidas para a gerência de redes pelo uso de SDN em infraestruturas na nuvem, grandes *datacenters* e *backbones*, este trabalho propõe integrar ferramentas de código aberto para a construção de um sistema que traga essa facilidade para a gerência de pequenos escritórios, permitindo uma migração gradual da infraestrutura legada e mantendo um potencial de expansão.

Uma ferramenta para um controle centralizado de redes poderia trazer uma grande variedade de funcionalidades. Para tornar viável a execução do projeto trazendo um resultado funcional, foi decidido restringir o escopo do mesmo para uma ferramenta capaz de criar uma segmentação de rede em *switches* openflow. Desta forma, também adicionamos as vantagens de usar um padrão aberto, ficando livre das travas de usar um único fabricante ou conjunto de fabricantes.

Alguns trabalhos semelhantes foram o trabalho de Nair (2016), em que o mesmo apresentou uma implementação do padrão de VLANs em um controlador SDN, demonstrando a viabilidade da implementação, sem a construção de uma ferramenta para controle do mesmo. E o trabalho de Nguyen (2016), que criou uma ferramenta semelhante à proposta, porém utilizando um outro controlador, onde o mesmo criou uma interface de suporte a VLANs para o modo de trabalho padrão do controlador, permitindo uma melhor compatibilidade com as redes atuais.

Apresentaremos, no capítulo dois, uma fundamentação teórica direcionada para os conceitos chave do trabalho, iniciando pelo conceito de redes locais virtuais e pelo padrão estabelecido pelo IEEE (*Institute of Electrical and Electronics Engineers*), seguida dos conceitos de redes definidas por software, do protocolo OpenFlow, do Open vSwitch e da descrição de alguns controladores SDN existentes. No capítulo três apresentaremos uma proposta de solução, detalhando a arquitetura e seus motivos. No capítulo quatro abordaremos os detalhes de implementação e no capítulo cinco os testes da aplicação. No capítulo seis encerraremos com as conclusões e propostas de trabalhos futuros.

2 **FUNDAMENTAÇÃO TEÓRICA**

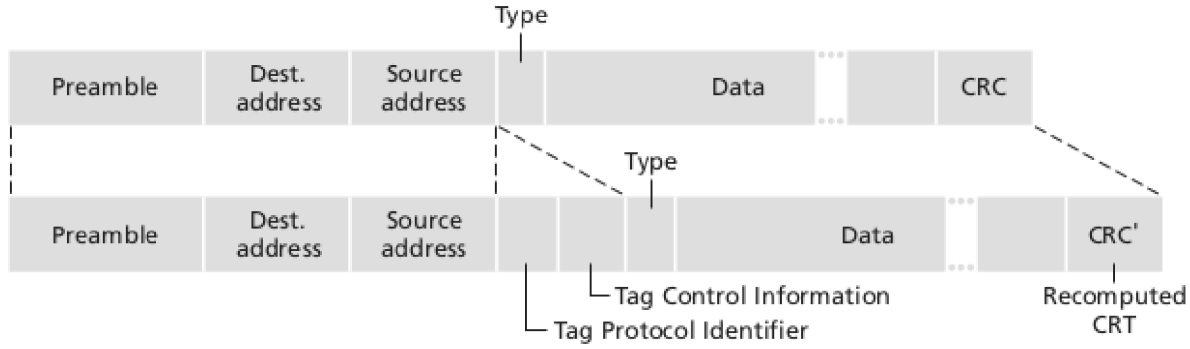
2.1 VLAN - Virtual Local Area Network

Como já dito, o uso de VLANs permite manter diversas redes locais virtuais dentro de uma única rede local física, reduzindo a subutilização de *switches* e a necessidade de mudanças na infraestrutura física da rede. Esta segmentação

pode se dar de forma estática, associando uma porta do *switch* a uma determinada VLAN, ou de forma dinâmica, como por exemplo, associando um endereço MAC (*Medium Access Control*) a uma VLAN, permitindo que um dispositivo mude de porta e se mantenha na mesma VLAN sem a necessidade de mudança de configuração (Kurose, 2013).

Segundo Kurose, 2013, para que fosse possível interconectar diversos *switches* de forma escalável, foi criado o conceito de trunking, onde uma porta em modo trunk pode pertencer a diversas VLANs. Para identificar os pacotes pertencentes às diversas VLANs de uma porta neste modo, o pacote Ethernet é alterado, de forma a adicionar um identificador de VLAN chamado VLAN Tag. Esta alteração aumenta o tamanho do pacote em quatro bytes, e não é reconhecido por equipamentos não compatíveis com o padrão, tornando necessário que as alterações sejam desfeitas antes que o pacote seja enviado para um dispositivo comum. O padrão que define esta alteração é o 802.1Q do IEEE.

Figura 1 - Pacote original Ethernet na parte superior e pacote alterado pelo padrão 802.1Q na parte inferior.



Fonte: Kurose (2013)

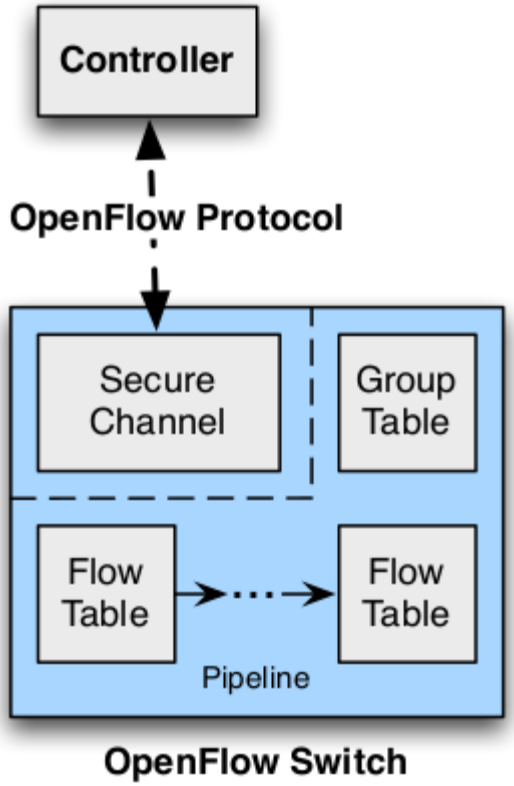
2.2 SDN - *Software Defined Network* e OpenFlow

Conforme citado anteriormente, as redes definidas por software trazem uma separação entre o plano de controle, responsável por definir como a rede se comporta, e o plano de dados, responsável por encaminhar e alterar os pacotes. Também é proposto um plano de gerência que seria a interface com o usuário.

Para que esta separação seja possível, é necessário que haja uma interface de comunicação bem definida entre o plano de controle e o plano de dados. O exemplo mais notável é o protocolo OpenFlow (Kreutz, 2015).

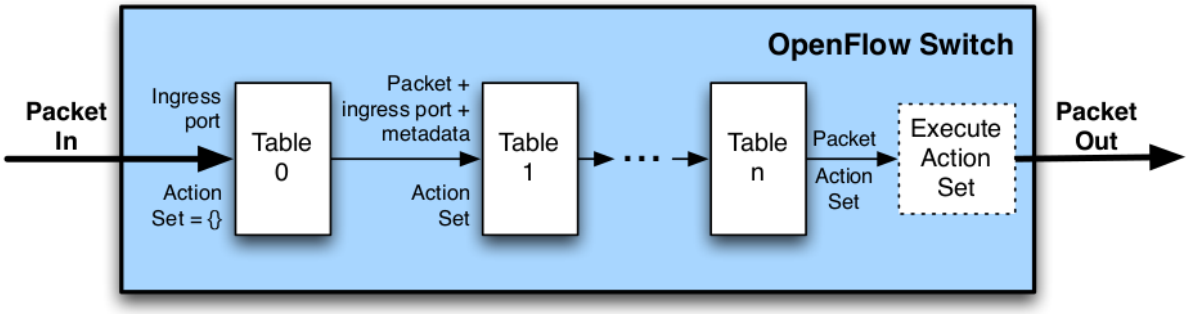
Um switch OpenFlow consiste em uma ou mais tabelas de fluxo que executam alterações e encaminhamento de pacotes, e um canal OpenFlow para um controlador externo. O controlador gerencia o *switch* através do protocolo OpenFlow, podendo adicionar, atualizar e excluir entradas de fluxo em tabelas de fluxo. Cada entrada de fluxo consiste em campos de correspondência, contadores e um conjunto de instruções para aplicar à correspondência. Cada entrada tem uma prioridade, e caso nenhuma correspondência seja encontrada deve ser aplicada uma regra padrão (Open Network Foundation, 2015).

Figura 2 - Conexão entre *switch* e controlador.



Fonte: Open Networking Foundation (2012)

Figura 3 - Fluxo do pacote em um *switch* OpenFlow..

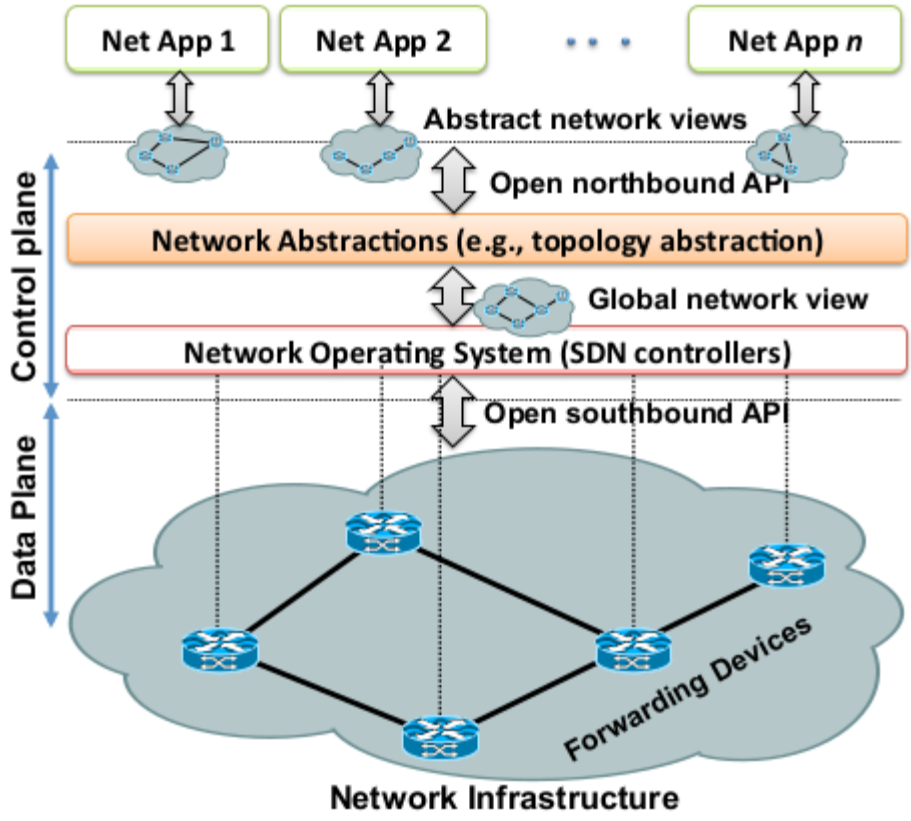


Fonte: Open Networking Foundation (2012)

2.3 CONTROLADORES

O controlador é responsável pelo plano de controle da rede, fornecendo uma interface de software bem definida para que aplicações terceiras mudem o seu comportamento, e se comunicando com os dispositivos de redes através de outras interfaces de software bem definidas. Na figura abaixo podemos observar estes componentes.

Figura 4 - Arquitetura SDN e suas principais abstrações.



Fonte: Kreutz (2015)

Para definir o controlador utilizado foram estudados diversos controladores de código aberto, dentre eles Floodlight, OpenDaylight e Ryu.

2.3.1 **Floodlight**

Conforme a documentação do projeto Floodlight (2018), o mesmo é um controlador SDN que utiliza o protocolo OpenFlow e que permite ambientes mistos com switches OpenFlow e não OpenFlow, utilizando o conceito de ilhas, onde grupos de hardware gerenciados pelo controlador podem estar ligados a grupos de hardware de uma infraestrutura tradicional. Um detalhe do conceito de ilhas adotada pelo Floodlight é que sua documentação deixa claro que cada ilha de hardware não OpenFlow é restrita a um único domínio de *broadcast*, sendo assim, não pode estar segmentada por VLANs.

Para aplicações externas, o Floodlight oferece uma API Rest que permite alterar parâmetros de suas aplicações internas. Por ser um projeto aberto também é possível fazer alterações no comportamento de suas aplicações internas, adicionar novas aplicações, criar *patches* entre outros.

2.3.2 **OpenDaylight**

Conforme sua documentação, o OpenDaylight é uma infraestrutura de controladores altamente disponível, modular, extensível, escalável e multiprotocolo criada para implementações de SDN em redes modernas de múltiplos fornecedores. O OpenDaylight fornece uma plataforma de abstração de serviços orientada a modelos que permite aos usuários escrever aplicativos que funcionam facilmente em uma grande variedade de hardware e de protocolos de comunicação com os dispositivos de rede (OpenDaylight project, 2013?).

Devido à quantidade de funcionalidades inclusas, o mesmo possui um complexo ambiente de desenvolvimento bem documentado na sua Wiki oficial, trazendo uma API em Java para o desenvolvimento de novas aplicações.

Conforme a documentação da comunidade do OpenDaylight, para segmentação de rede, é introduzido o conceito de VTN (*Virtual Tenant Network*), uma segmentação lógica feita internamente pelo controlador que permite o mapeamento para VLANs de uma infraestrutura tradicional.

Devido à proposta do projeto de ser flexível para qualquer ambiente, o OpenDaylight impõe a barreira de que, para implantar o mesmo, é necessário conhecer o seu complexo funcionamento e sua grande variedade de módulos para chegar em uma configuração que atenda suas necessidades.

2.3.3 **Ryu**

O Ryu se define como um conjunto de ferramentas para redes definidas por software baseado em componentes. Provê uma interface bem definida para que desenvolvedores possam criar ferramentas de controle e gerenciamento de rede, e também dá suporte a diversas versões do protocolo openflow, assim como outros protocolos (Ryu SDN Framework Community, 2016?).

O mesmo é totalmente construído em Python, possuindo uma documentação detalhada tanto das ferramentas quando de alguns conceitos. É focado em desenvolvimento e não possui um controlador final construído, porém possui diversas aplicações de exemplo que implementam funcionalidades como: firewall, *switch* sem segmentação, protocolo *spanning tree* e outros.

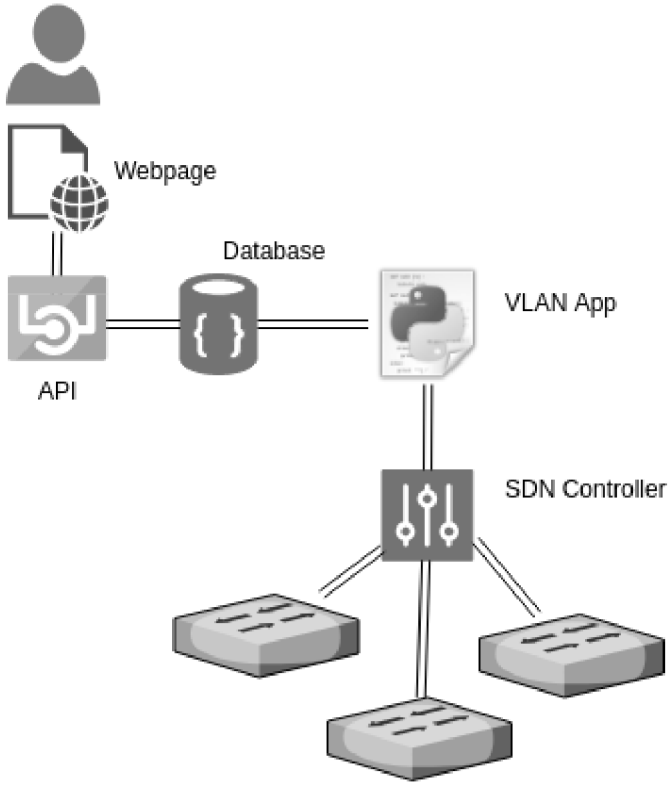
O foco no desenvolvimento permite que o projeto já tenha compatibilidade com versões mais recentes do protocolo OpenFlow, como a 1.5, assim como suporte a funcionalidades estendidas.

3 **FERRAMENTA PROPOSTA**

Para a solução pensada, propomos a construção de um sistema integrando diversas ferramentas de código aberto. Como base para o projeto teríamos três ferramentas externas principais: uma base de dados, responsável por persistir as informações de configuração desejadas pelo usuário e as informações de topologia fornecidas pelo controlador; um controlador SDN, responsável por gerenciar os *switches* através do protocolo OpenFlow, além de permitir que aplicações externas se conectem a ele modelando seu comportamento; e por fim os *switches* compatíveis com o protocolo OpenFlow, que são responsáveis por direcionar os pacotes recebidos, de acordo com as tabelas de fluxo fornecidas pelo controlador.

Para desenvolvimento no projeto propomos três módulos, uma API Rest que permite que algumas operações sejam feitas na base de dados da aplicação, uma página web que dá ao usuário uma interface para executar as operações disponibilizadas pela API e uma aplicação diretamente conectada ao controlador, que é responsável por definir as tabelas de fluxo de acordo com as informações da base de dados e das informações obtidas dinamicamente pelo controlador.

Figura 5 - Arquitetura dos componentes do projeto, contendo as ferramentas externas (base de dados, controlador SDN e *switches*) e ferramentas desenvolvidas (API, Página Web e Aplicação SDN).



Fonte: Autoria própria (2018)

3.1 API REST

A API deve permitir a execução de um conjunto de operações sobre os conjuntos de dados necessários para o funcionamento da aplicação. Todas as mudanças de configurações ou consultas de informações do controlador deverão se dar pela API, sendo papel da mesma manter a consistência dos dados.

Durante o desenvolvimento chegamos a três conjuntos de dados: redes, *switches* sob gerência e *switches* conectados ao controlador.

As redes estão diretamente relacionadas às VLANs, atualmente elas possuem apenas um nome e o identificador da VLAN. Atualmente a API permite ler, atualizar e criar redes. Uma possível expansão é incluir a faixa de endereços CIDR da rede, de forma a evitar que duas redes sejam planejadas com colisão de endereços.

As informações sobre os *switches* conectados ao controlador contêm o identificador do *switch* e identificador, nome e endereço MAC de cada porta. A

API trata este dado como apenas leitura, estas informações são inseridas na base de dados pela aplicação conectada ao controlador.

Os *switches* sob gerência são os *switches* que o usuário reconhece como presentes na sua topologia. Contêm todas informações de um *switch* conectado ao controlador, com a adição em cada porta de uma lista de identificadores de VLANs e um modo de operação que pode ser Trunk ou Access. Assim como no caso das redes, a API permite a criação, atualização e remoção destes.

A implementação da API foi feita em Python2.7 utilizando as bibliotecas abertas Flask-RESTFul e Mongoengine.

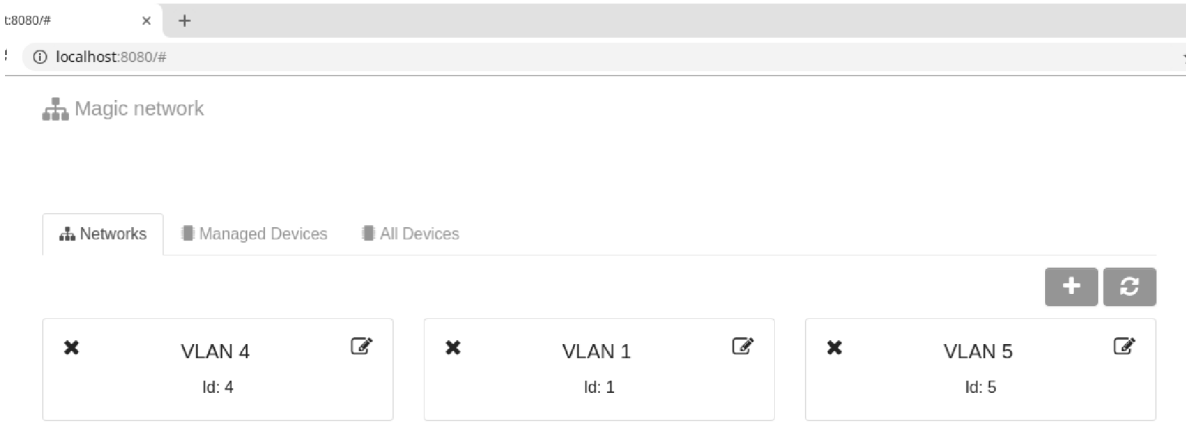
3.2 PÁGINA WEB

A função da página WEB deve ser permitir a utilização de todos os itens da API, fazendo as devidas simplificações. A mesma deve ser intuitiva e simples, requerendo o mínimo de conhecimento prévio possível sobre a aplicação.

As simplificações de uso da API presentes na implementação foram que, na seção de *switches* conectados ao controlador é possível importar seus dados para a seção de *switches* gerenciados, facilitando a configuração inicial. Outro ponto de simplificação é que pela página, só é possível associar a uma porta, redes previamente cadastradas. Para a construção da página foram utilizados os projetos de código aberto Bootstrap e Vue.js.

A página contém três abas, cada uma responsável por um conjunto de dados. A primeira aba é responsável pelas redes, contém uma visualização de cada rede criada, com seu nome e identificador de VLAN, permitindo a criação de novas redes e exclusão das existentes.

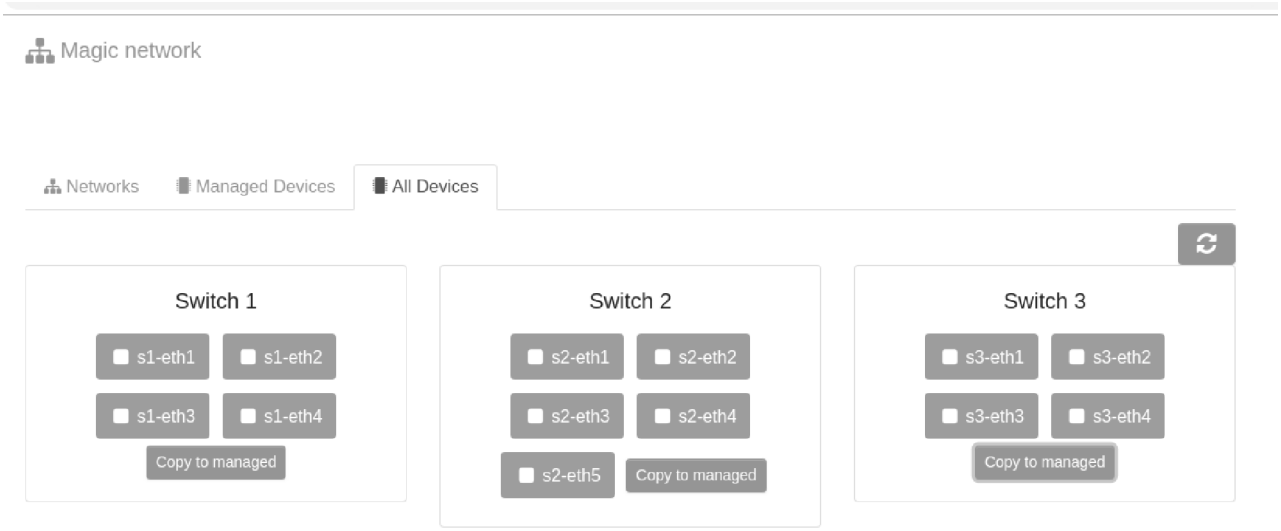
Figura 6 - Aba de edição de redes da aplicação proposta.



Fonte: Captura de tela da aplicação (2018)

A terceira aba é responsável por exibir os *switches* conectados ao controlador, como este dado é apenas de leitura, não é possível editar ou excluir nenhum deles, existe apenas uma opção de copiar os dados para a seção de *switches* sob gerência. Nesta aba podemos visualizar a identificação do *switch* e todas as suas portas.

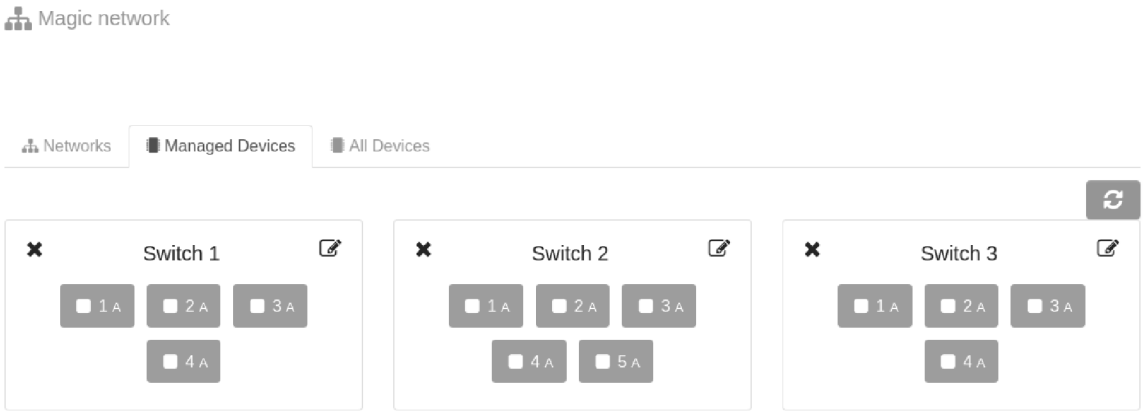
Figura 7 - Aba de *switches* conectados à rede da aplicação proposta.



Fonte: Captura de tela da aplicação (2018)

A segunda aba é responsável pelos *switches* sob gerência, permitindo a edição e remoção de *switches*. Para que um novo *switch* seja adicionado, ele deve ser primeiramente reconhecido pelo controlador e copiado para a seção de *switches* sob gerência. Quando um *switch* é copiado, as informações adicionais ficam de forma padrão, com o modo em Access e o identificador de VLAN 1.

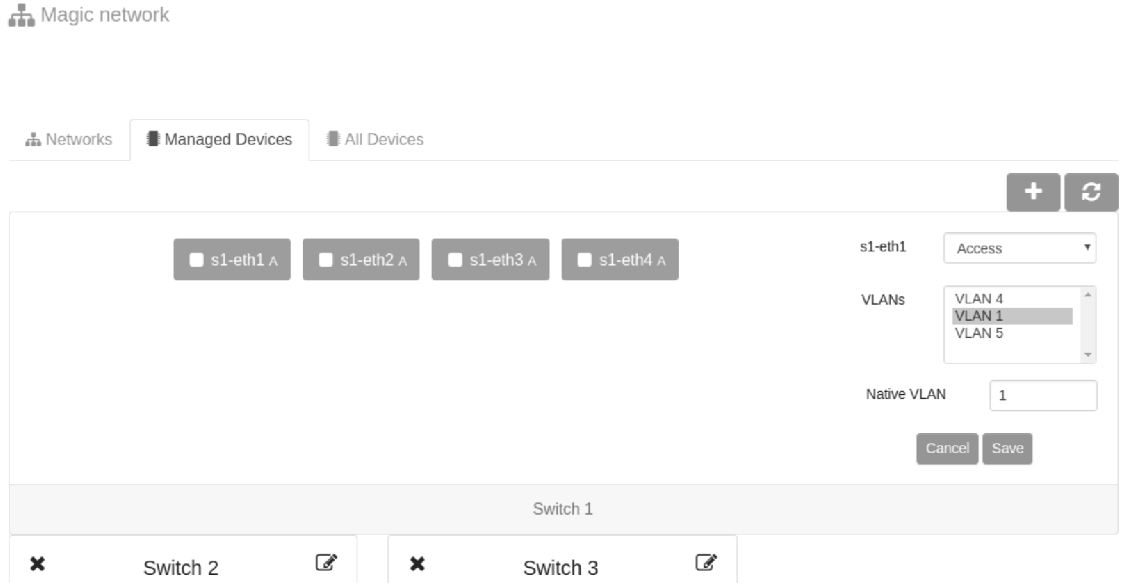
Figura 8 - Aba de *switches* gerenciados da aplicação proposta.



Fonte: Captura de tela da aplicação (2018)

Ao editar um *switch* sob gerência, é possível selecionar qualquer uma de suas portas e escolher as configurações utilizadas pela mesma, nas VLANs da porta é possível escolher uma ou várias entre as redes criadas no sistema, também é possível definir o modo entre Access e Trunk.

Figura 9 - *Switch* gerenciado tendo a configuração de uma das portas sendo editada na aplicação proposta.



Fonte: Captura de tela da aplicação (2018)

3.3 APLICAÇÃO SDN

A aplicação SDN será responsável pela criação de todos os fluxos presentes nos switches, se baseando apenas nas configurações presentes na base de dados e

nas informações descobertas durante sua execução. Ao mesmo tempo, a mesma deve ser capaz de manter na base de dados informações sobre os dispositivos conectados no controlador, e outras informações que possam ser pertinentes ao responsável por configurar a rede. Também é esperado que a aplicação trate pacotes que sigam o padrão 802.1Q da forma correta, tratando os identificadores de VLAN e adicionando ou removendo o cabeçalho adicional quando necessário.

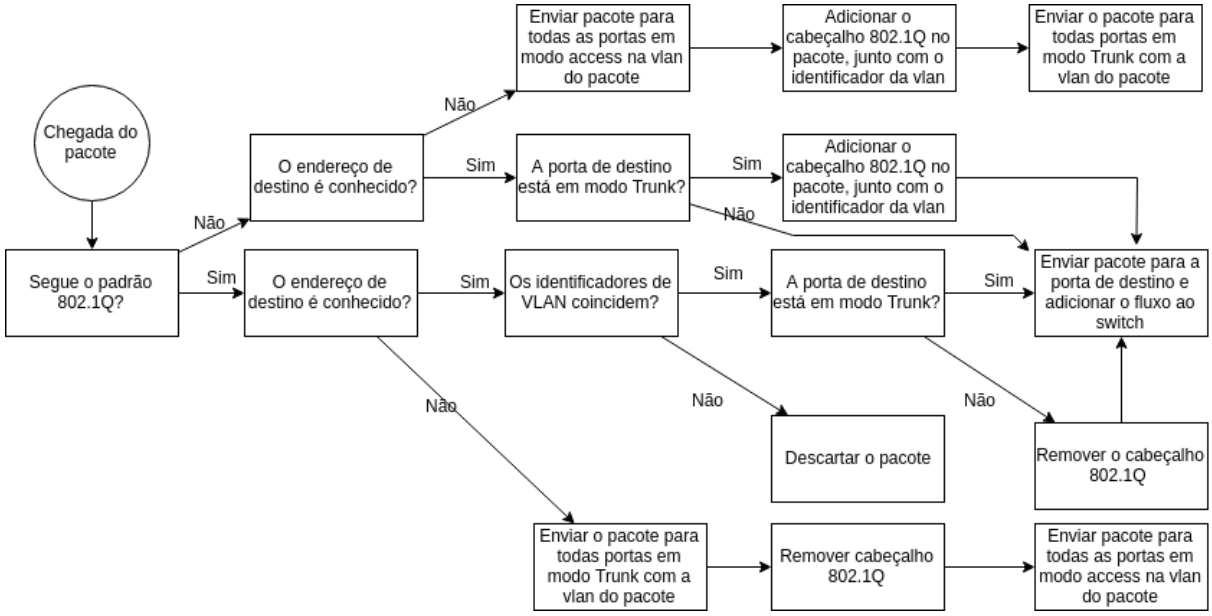
Devido ao foco maior em desenvolvimento e a simplicidade do ambiente necessário, o controlador SDN utilizado no projeto foi o Ryu, de forma a tornar tanto a implementação quanto a implantação da solução proposta mais simples. No seu funcionamento básico, o Ryu não trata os fluxos de dados nos *switches*, então todo este tratamento é feito pela aplicação.

A criação de um fluxo na aplicação consiste em definir um conjunto de condições, um conjunto de ações caso estas condições sejam verdadeiras, uma prioridade e um tempo de expiração. Essas informações são empacotadas e enviadas para um *switch* como um pedido de alteração da sua tabela de fluxo

Seguindo um modelo comum de outras aplicações do controlador, a aplicação escuta dois eventos específicos: novo *switch* conectado ao controlador e novo pacote enviado ao controlador. Ao receber o evento de um novo switch conectado, a aplicação instala nele um fluxo padrão com prioridade mínima, um conjunto de condições sempre verdadeiras, sem tempo de expiração e a ação de encaminhar os pacotes para o controlador. Desta forma, caso um pacote não case com nenhum outro fluxo, o controlador deverá tratar isto. Ao iniciar, a aplicação também cria uma *thread* que é responsável por periodicamente consultar a base de dados sobre a configuração atual do sistema. O intervalo entre as consultas foi fixado em dez segundos, podendo ser modificado para ajustes de desempenho.

Quando pacotes de dados chegam no controlador, a aplicação identifica inicialmente se o cabeçalho do pacote segue o padrão 802.1Q, identifica a VLAN do pacote, porta de origem e endereçamento MAC de origem e destino. Em seguida, encaminha para as portas necessárias, removendo ou adicionando o cabeçalho 802.1Q quando necessário. Abaixo podemos observar um diagrama mais detalhado de todo o fluxo.

Diagrama 1 - Diagrama de fluxo da aplicação SDN .



Fonte: Autoria própria (2018)

As ações de enviar pacotes para todas as portas em uma mesma VLAN não instalam nenhum fluxo no *switch*, pois, o controlador utiliza esta ação para aprender o endereço MAC de cada dispositivo na rede. Caso o fluxo fosse feito diretamente pelo *switch*, os fluxos de enviar para uma porta específica nunca seriam instalados, tornando todo tráfego *broadcast*. Mesmo não instalando um fluxo, o controlador não executa as modificações, apenas envia de volta para o *switch* uma mensagem contendo o pacote original e o conjunto de ações a serem executadas com o pacote, sendo estas ações restritas às mesmas ações que podem ser executadas por um fluxo.

Para que os *switches* acompanhem as atualizações de configuração, todos os fluxos são adicionados com um tempo de expiração fixo, fixado também em dez segundos, podendo ser modificado para ajustes de desempenho, com exceção do fluxo padrão. Desta forma, temos um tempo entre a aplicação de uma nova configuração pelo usuário e o seu funcionamento na rede. Na configuração atual fixamos o tempo máximo para que todo sistema receba uma atualização de configuração como a soma do intervalo de consulta à base de dados, com o tempo de expiração dos fluxos.

3.4 BIBLIOTECAS E FERRAMENTAS

Como citado anteriormente, algumas bibliotecas e ferramentas de código aberto foram utilizadas na construção da aplicação. Esta seção se destina a descrever o

funcionamento básico e utilização de cada uma delas. Em sua maioria, as escolhas das ferramentas se deram por simplicidade de uso, atender às necessidades do projeto e conhecimento prévio. Itens necessários para tornar o projeto viável no espaço de tempo disponível.

Uma das bibliotecas utilizadas na construção da API foi a FlaskRESTful. A mesma é uma extensão da biblioteca Flask e é responsável por subir o socket servidor e tratar as requisições HTTP, permitindo que a construção da API fosse focada apenas em definir o comportamento dos dados em cada requisição.

Para a base de dados foi utilizado o banco MongoDB, que é um banco de dados não relacional baseado em documentos, permitindo operar de forma distribuída e sem a definição de um schema. A simplicidade de armazenar os dados em coleções de documentos sem definir previamente um schema dos dados, permitiu um desenvolvimento mais ágil da aplicação.

Outra biblioteca utilizada tanto na API quando na aplicação SDN foi a Mongoengine, que é responsável por abstrair as consultas à base de dados e a conversão de tipos, fazendo um mapeamento dos documentos salvos na base para objetos em Python, linguagem utilizada em ambos.

Na construção da página foi utilizado o projeto de código aberto Bootstrap. O mesmo é um framework HTML, CSS e JavaScript que permite a construção de sites utilizando itens prontos e uma estrutura planejada para ser compatível com diversos tamanhos de tela.

Outro framework de código aberto utilizado na construção da página foi o Vue.js. O mesmo é responsável por construir dinamicamente os itens do site, de acordo com os dados, além de manter uma ligação de mão dupla entre os itens exibidos e os dados, refletindo as mudanças de um no outro.

4 **TESTES**

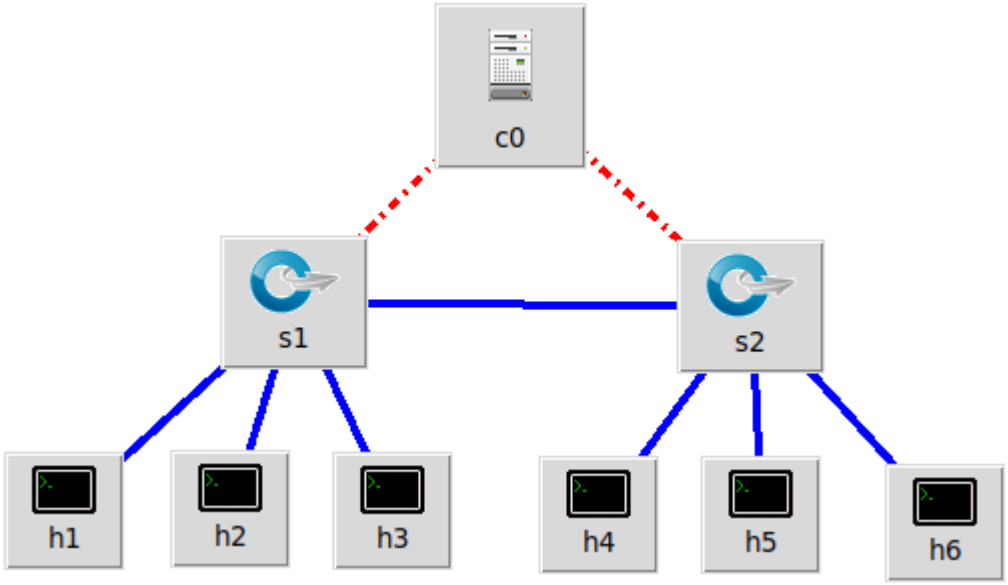
Para testar a aplicação proposta foi utilizado o Mininet, um projeto de código aberto capaz de criar uma rede virtual realista de forma simples e rápida. Através dele podemos adicionar na nossa rede virtual máquinas, *switches* legados, *switches* OpenFlow (utilizando Open vSwitch) e roteadores OpenFlow.

Para os testes do funcionamento da segmentação criamos duas topologias, e segmentamos cada topologia com uma e duas VLANs. Dado isto testamos a alcançabilidade entre as máquinas virtuais.

Também testamos o tempo de expiração dos fluxos e a eficácia de mudar a segmentação durante a operação fazendo alterações na configuração enquanto duas máquinas se comunicam. Mais detalhes dos testes serão abordados abaixo.

4.1 TOPOLOGIA COM DOIS *SWITCHES* E TRÊS MÁQUINAS POR *SWITCH*

Figura 10 - Topologia com dois *switches* e três máquinas por *switch.*

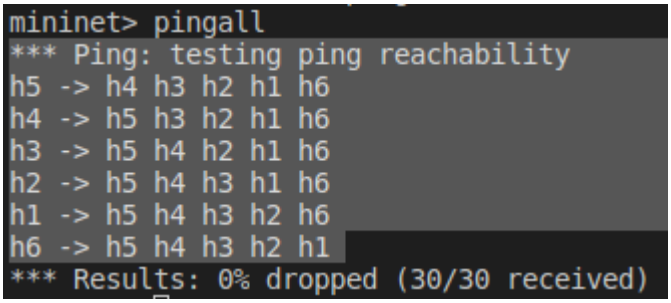


Fonte: Autoria própria (2018)

Esta topologia é composta por dois *switches* Open vSwitch, cada um com três máquinas virtuais conectadas no mesmo e ambos sendo controlados pelo controlador Ryu, em conjunto com a aplicação proposta neste trabalho.

Para o primeiro teste todos as portas foram colocadas em modo access e com identificador de VLAN 1 (um). O resultado dos testes de alcançabilidade podem ser vistos na imagem abaixo.

Figura 11 - Testes de conectividade na topologia com dois *switches* e três máquinas por *switch*, com VLAN única.

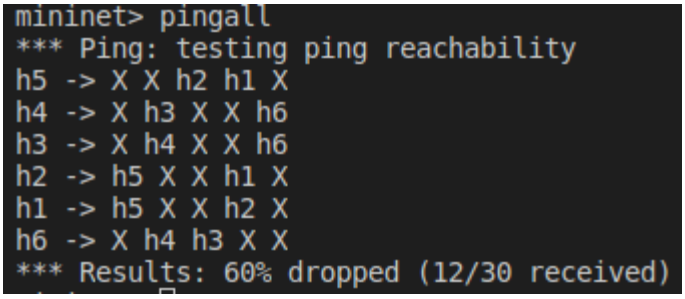


Fonte: Captura de tela da aplicação Mininet (2018)

Na imagem podemos observar que cada máquina consegue alcançar todas as outras, como era esperado com a configuração imposta ao sistema.

Para o segundo teste foram criadas as redes com identificadores de VLAN 1 (um) e 3 (três). As máquinas foram segmentadas da seguinte forma: Na VLAN 1 ficam as máquinas h1, h2 e h5; na VLAN 2 ficam as máquinas h3, h4 e h6. As portas que interligam os *switches* foram colocadas em modo Trunk com ambas as VLANs. O resultado do teste pode ser visto na imagem abaixo.

Figura 12 - Testes de conectividade na topologia com dois *switches* e três máquinas por *switch*, com múltiplas VLANs.

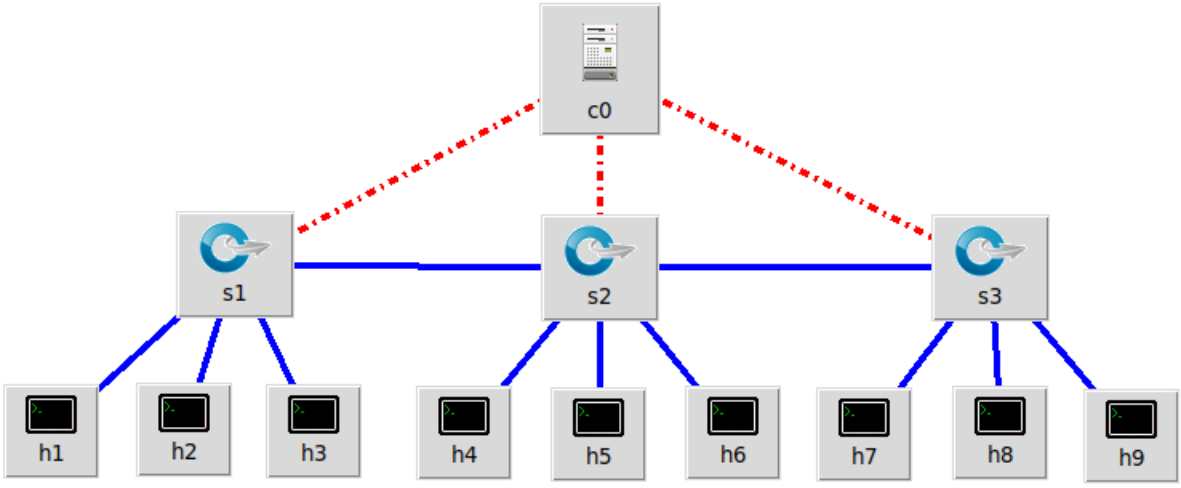


Fonte: Captura de tela da aplicação Mininet (2018)

Como esperado, as máquinas h1, h2 e h5 formam uma segmentação na rede, sendo visíveis apenas entre si; assim como as máquinas h3, h4 e h6 formam uma segunda segmentação, comprovando o funcionamento da porta em modo Trunk em uma topologia simples, assim como das portas em modo Access. Como cada máquina enxerga apenas duas de cinco máquinas que tenta alcançar, temos um resultado de quarenta por cento de resposta e sessenta por cento de pacotes perdidos.

4.2 TOPOLOGIA COM TRÊS *SWITCHES* E TRÊS MÁQUINAS POR *SWITCH*

Figura 13 - Topologia com três *switches* e três máquinas por *switch*.



Fonte: Autoria própria (2018)

Esta topologia é composta por três *switches* Open vSwitch, cada um com três máquinas virtuais conectadas no mesmo e todos sendo controlados pelo controlador Ryu, em conjunto com a aplicação proposta neste trabalho.

Para o primeiro teste foram tomados os mesmos procedimentos utilizados na topologia anterior, todos as portas foram colocadas em modo access e com identificador de VLAN 1 (um). O resultado dos testes de alcançabilidade podem ser vistos na imagem abaixo.

Figura 14 - Testes de conectividade na topologia com três *switches* e três máquinas por *switch*, com VLAN única.

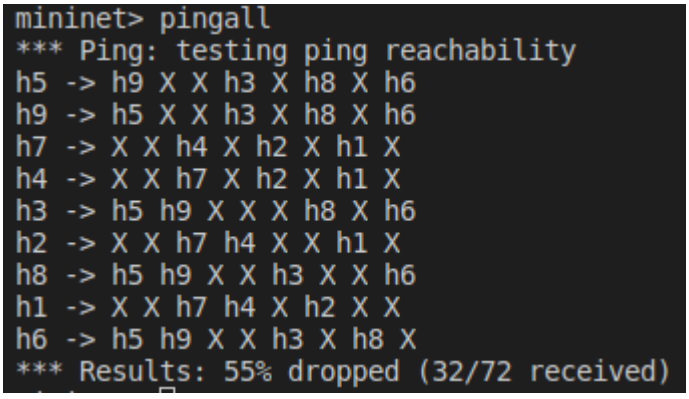
```
mininet> pingall
*** Ping: testing ping reachability
h5 -> h9 h7 h4 h3 h2 h8 h1 h6
h9 -> h5 h7 h4 h3 h2 h8 h1 h6
h7 -> h5 h9 h4 h3 h2 h8 h1 h6
h4 -> h5 h9 h7 h3 h2 h8 h1 h6
h3 -> h5 h9 h7 h4 h2 h8 h1 h6
h2 -> h5 h9 h7 h4 h3 h8 h1 h6
h8 -> h5 h9 h7 h4 h3 h2 h1 h6
h1 -> h5 h9 h7 h4 h3 h2 h8 h6
h6 -> h5 h9 h7 h4 h3 h2 h8 h1
*** Results: 0% dropped (72/72 received)
```

Fonte: Captura de tela da aplicação Mininet (2018)

Mais uma vez, conforme esperado, cada máquina consegue alcançar todas as outras.

Para o segundo teste repetimos os procedimentos da topologia anterior e foram criadas as redes com identificadores de VLAN 1 (um) e 3 (três). As máquinas foram segmentadas da seguinte forma: Na VLAN 1 ficam as máquinas h1, h2, h4 e h7; na VLAN 2 ficam as máquinas h3, h5, h6, h9 e h8. As portas que interligam os *switches* foram colocadas em modo Trunk com ambas as VLANs. O resultado do teste pode ser visto na imagem abaixo.

Figura 15 - Testes de conectividade na topologia com três *switches* e três máquinas por *switch*, com múltiplas VLANs.



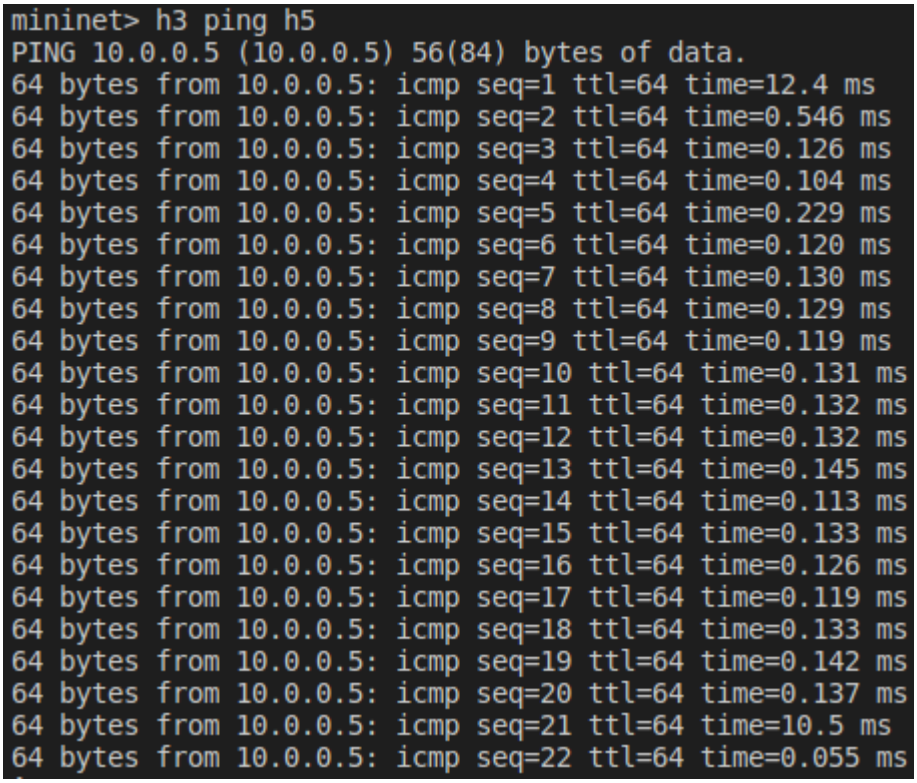
Fonte: Captura de tela da aplicação Mininet (2018)

Podemos observar que mesmo em uma topologia um pouco mais complexa, os resultados se comportaram conforme o esperado. Demonstrando uma segmentação de rede contendo as máquinas h1, h2, h4 e h7, e uma segunda segmentação contendo as máquinas h3, h5, h6, h8 e h9.

4.3 FLUXOS E MUDANÇAS DE CONFIGURAÇÃO

Para aferir que os fluxos instalados nos switches expiram de tempos em tempos podemos observar o tempo de ida e volta dos pacotes entre duas máquinas. Na figura abaixo podemos observar que periodicamente, um dos pacotes tem um tempo mais elevado, isto pode ser observado nos pacotes um e vinte e um. Isto se deve ao fato do fluxo ter expirado e o pacote ter sido novamente enviado ao controlador, esta forma de operação reduz o desempenho porém garante que fluxos antigos não se sobreponham a mudanças de configuração.

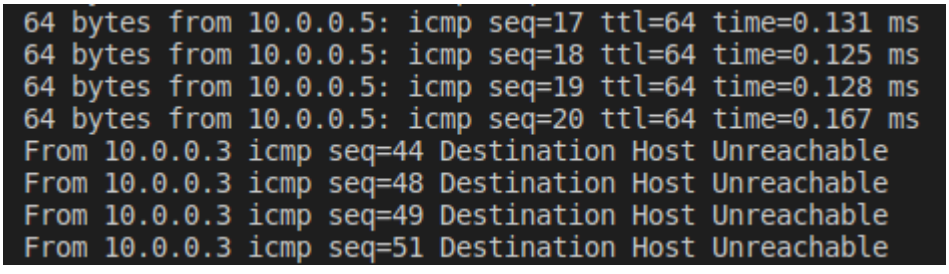
Figura 16 - Testes do efeito da expiração do fluxo na conectividade entre duas máquinas.



Fonte: Captura de tela da aplicação Mininet (2018)

Num segundo teste mantivemos duas máquinas em diferentes *switches* se comunicando e removemos a VLAN de uma das portas Trunk, desta forma era esperado confirmar que apenas pacotes das VLANs listadas na porta passariam pela mesma. Como pode ser visto na imagem abaixo, o comportamento foi confirmado e a comunicação interrompida.

Figura 17 - Testes da eficácia da aplicação de uma mudança de configuração no sistema..



Fonte: Captura de tela da aplicação Mininet (2018)

Com estes testes conseguimos demonstrar o comportamento esperado da aplicação, compatível com os sistemas atuais e com grande potencial de expansão.

Durante a implementação estes testes permitiram detectar comportamentos não esperados devido a erros no fluxo da aplicação. Numa primeira versão a segmentação funcionava normalmente para a configuração inicial. Porém, ao mudar a configuração, duas máquinas que se comunicavam anteriormente e deviam parar de se comunicar, não interrompiam a comunicação. Isto se devia ao fato de que o identificador de VLAN era verificado apenas no *broadcast*, criando uma segmentação inicial garantida pelo fato das máquinas não se conhecerem anteriormente, porém burlada em mudanças. Adicionar a verificação também em pacotes *unicast* corrigiu o comportamento.

5 **CONCLUSÕES E TRABALHOS FUTUROS**

Considerando os resultados obtidos durante a implementação e testes, chegamos à conclusão de que a elaboração de tal projeto é viável. O uso de outros projetos de código aberto durante a implementação o trabalho necessário para resolver problemas de outros domínios, permitindo focar no comportamento esperado da aplicação e controlador escolhido apresentou resultados satisfatórios nos testes além de trazer a simplicidade já esperada na aplicação.

Algumas funcionalidades e trabalhos futuros para o projeto são a adição de outras funcionalidades essenciais para redes empresariais, como roteamento entre diferentes segmentos, firewall de perímetro, trazer a opção de automatizar a configuração de conexões entre dois *switches*, permitir redundância de links compatível com Spanning Tree, permitir agregação de links e outras funções. Mantendo sempre o controle centralizado e tornando o mais simples possível a execução das atividades de gerência de rede.

REFERÊNCIAS

KUROSE, Jim; ROSS, Keith. **Computer Networking a top-down approach**. 6 ed. [S.L.]: Pearson, 2013.

KREUTZ, Diego; et al. **Software-Defined Networking: A Comprehensive Survey.** Proceedings of the IEEE, Volume: 103, Issue: 1, Jan. 2015.

KROTHAPALLI, S.A; et al. **Virtual MAN: a VLAN management system for enterprise networks**. Chicago: SafeConfig'09, 2009.

NADEAU, Thomas D; GRAY, Ken. **SDN: Software Defined Networks.** 1 ed. [S.L.]: O'Reilly, 2013.

NAIR, Varun. **Implementatios of IEEE 802.1Q VLAN Tagging using Ryu OpenFlow controller**. Electrical Engineering, Delft University of Technology, 2016.

NGUYEN, Van-Giang; KIM, Young-Han. **SDN-Based Enterprise and Campus Networks: A Case of VLAN Management.** Journal of Information Processing Systems, Vol.12, No.3, pp.511~524, Set. 2016.

OPENDAYLIGHT COMUNITY. **VTN Overview**. Disponível em: <https://github.com/opendaylight/docs/blob/stable/lithium/manuals/user-guide/src/main/asciidoc/vtn/VTN_Overview.adoc> Acesso em: 5 dez. 2018.

OPENDAYLIGHT PROJECT. **OpenDaylight**. Disponível em: <https://wiki.opendaylight.org> Acesso em: 5 dez. 2018.

OPEN NETWORKING FOUNDATION. **OpenFlow Switch Specification: Version 1.3.5**. Disponível em: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf> Acesso em: 7 set. 2018.

PROJECT FLOODLIGHT. **Floodlight**. Disponível em: <http://www.projectfloodlight.org/floodlight/> Acesso em: 5 dez. 2018.

RYU SDN FRAMEWORK COMMUNITY. **Welcome to RYU the Network Operating System(NOS)**. Disponível em: <https://ryu.readthedocs.io/en/latest/> Acesso em: 5 dez. 2018.

YU, Minlan; et al. **A Survey of Virtual LAN Usage in Campus Networks.** IEEE Communications Magazine, Jul, 2011.