UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Henrique Montezelo

Práticas Laboratoriais em SDN usando Mininet e POX

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Henrique Montezelo

Práticas Laboratoriais em SDN usando Mininet e POX

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Luís Fernando Faina

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2017

Gabriel Henrique Montezelo

Práticas Laboratoriais em SDN usando Mininet e POX

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 27 de julho de 2017:

Professor Dr. Luís Fernando Faina
Orientador

Professor Dr. Rafael Pasquini

Professor Dr. Marcelo Rodrigues de Sousa

Uberlândia, Brasil
2017

Dedico a meus pais e irmã.

Agradecimentos

Agradeço primeiramente aos meus pais e irmã pelo apoio durante a minha graduação. Aos meus amigos que me ajudaram a superar dificuldades encontradas. Ao meu orientador, Luís Fernando Faina, pelos ensinamentos durante o desenvolvimento deste projeto.

"Às vezes as pessoas que ninguém imagina que poderiam fazer algo são as que fazem coisas que ninguém poderia imaginar."

Alan Turing

Resumo

Este trabalho foi desenvolvido visando expor alguns dos problemas da arquitetura de redes atual e contrapor com um novo padrão que vem sendo desenvolvido nos últimos anos. Após a definição de Redes Definidas por *Software* são apresentadas três práticas laboratoriais que visam explicar aos alunos esses conceitos de forma fácil e rápida através da utilização de Mininet e POX.

Palavras-chave: Redes Definidas por *Software*, práticas laboratoriais, Mininet, POX.

Lista de ilustrações

Lista de abreviaturas e siglas

SDN	*Software Defined Network*
SNMP	*Simple Network Management Protocol*
API	*Application Programming Interface*
MAC	*Media Access Control*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
OSPF	*Open Shortest Path First*

Sumário

1 Introdução

Neste capítulo é apresentada uma breve introdução deste trabalho, contendo a visão geral do trabalho na Seção 1.1, os objetivos do mesmo na Seção 1.2, suas justificativas na Seção 1.3 e a metodologia utilizada na Seção 1.4.

1.1 Visão Geral

Nos últimos anos, com o crescente número de pessoas e dispositivos conectados à *Internet*, a arquitetura de rede tradicional se tornou insuficiente para atender as novas demandas de utilização das redes de computadores. Isto se deve ao fato de que nesse padrão a hierarquia de *switches* está disposta em árvores, nas quais a comunicação é realizada de forma vertical. Esse tipo de comunicação permitiu a disseminação e domínio da arquitetura cliente-servidor.

Atualmente, o aumento no número de serviços móveis e computação em nuvem faz com que o ambiente dominante não seja mais o ideal visto que a maioria das comunicações passaram a ser realizadas de forma horizontal na árvore estrutural (ONF, 2012).

Outra diferença entre as arquiteturas é que na estrutura atual o plano de dados e o plano de controle estão unidos dentro dos dispositivos de redes, reduzindo a flexibilidade e restringindo a inovação e evolução da infraestrutura de redes.

Já nas Redes Definidas por Software (SDN - *Software Defined Networking*), existe uma separação de plano de controle e plano de dados. Com essa separação, os *switches* da rede se tornam apenas dispositivos de repasse enquanto o controle lógico se torna uma entidade externa centralizada simplificando as políticas de configuração e evolução, o que torna a rede programável, visto que há aplicações sendo executadas no topo do plano de controle (KREUTZ et al., 2015).

Nesse trabalho, propõe-se o estudo dos conceitos de Redes SDN com auxílio de Práticas Laboratoriais tendo por base o Mininet e o POX. O Mininet tem a função de criar o ambiente necessário para a estrutura da rede através da instanciação de *switches*. Por sua vez, o POX assume o papel do plano de controle da rede, tomando as decisões sobre o fluxo dos dados. O maior desafio do projeto é identificar os principais tópicos de redes definidas por *software* a serem tratados na disciplina e como lecioná-los aos alunos devido ao pouco tempo hábil para abordagem do assunto (ANTONENKO; SMELYANSKIY, 2013), (NOXREPO, 2012).

1.2 Objetivos

Esse trabalho tem como objetivo principal a introdução de tópicos avançados de redes de computadores em cursos de graduação em Ciência da Computação através de Práticas Laboratoriais, explorando para tanto os conceitos de Redes SDN. Dentre os objetivos, destacam-se: estudo dos princípios fundamentais de Redes SDN; implementação de Redes SDN tendo por base o Mininet e POX; e migração de um cenário já existente na topologia tradicional para um ambiente SDN.

1.3 Justificativas

A utilização de novas arquiteturas de redes vem crescendo consideravelmente nos últimos anos. No entanto, as disciplinas de Redes de Computadores em cursos de graduação normalmente não possuem tópicos referentes à essas arquiteturas em seu plano de ensino. Isto de deve ao fato de que muitas vezes essas disciplinas se encontram sobrecarregadas de conteúdos, dificultando a inserção de novos tópicos em seu programa.

Nesse contexto, o ensino de Redes SDN está entre os conceitos dessas novas arquiteturas. Por outro lado, por se tratar de tópicos bem mais complexos que tópicos tradicionais em redes de computadores, práticas laboratoriais podem facilitar a compreensão desses conceitos através de exemplos.

Se considerarmos a apresentação e discussão desse tópico em aulas teóricas, a percepção e análise do mesmo por parte dos alunos pode ser comprometida, já que a visão da separação dos planos de controle e dados não é um conceito simples, ao passo em que cenários em práticas laboratoriais podem passar essa percepção.

1.4 Metodologia

Este tópico tem como foco principal a apresentação da metodologia a ser aplicada para alcançar os objetivos citados anteriormente. Esses objetivos são divididos em duas categorias: estudo da estrutura de Redes SDN e implementação das práticas laboratoriais.

Conceitos e Princípios de Redes SDN - Estudo dos conceitos e princípios fundamentais de Redes SDN, afim de selecionar os pontos relevantes dessa arquitetura a serem explorados em práticas laboratoriais.

Configuração e Utilização do Mininet e POX - Instalação e configuração do Mininet como estrutura base para a instanciação de *switches*, visto que o *software* em questão tem como função a representação do plano de dados em Redes SDN. Por outro lado, a instalação e configuração do POX permite a configuração do *software*

responsável pelo controle do plano de dados, ou seja, exerce o papel do plano de controle.

Implementação do Cenário SDN - Implementar a prática laboratorial já vista pelos alunos, mas utilizando agora redes definidas por *software*.

2 Referencial Teórico e Trabalhos Correlatos

Este capítulo tem como objetivo estabelecer uma visão geral a respeito da topologia das redes de computadores, citar como está organizado o modelo tradicional, explicar os conceitos de SDN especificando seus componentes principais, e por fim mencionar alguns trabalhos na área.

2.1 Visão Geral de Redes de Computadores

Redes de Computadores podem ser divididas em três planos de funcionalidades: Plano de Dados, referente aos dispositivos físicos da rede e o repasse de dados; Plano de Controle, que diz respeito aos protocolos utilizados para o povoamento das tabelas de fluxo; Plano de Gerenciamento, responsável pelo monitoramento remoto e configurações de funcionalidades, incluindo serviços de *software*, como por exemplo, as ferramentas SNMP (*Simple Network Management Protocol* - Protocolo Simples de Gerência de Rede). Resumindo, o Plano de Gerenciamento define as políticas da rede, o Plano de Controle reforça essas políticas e o Plano de Dados encaminha os pacotes pela rede.

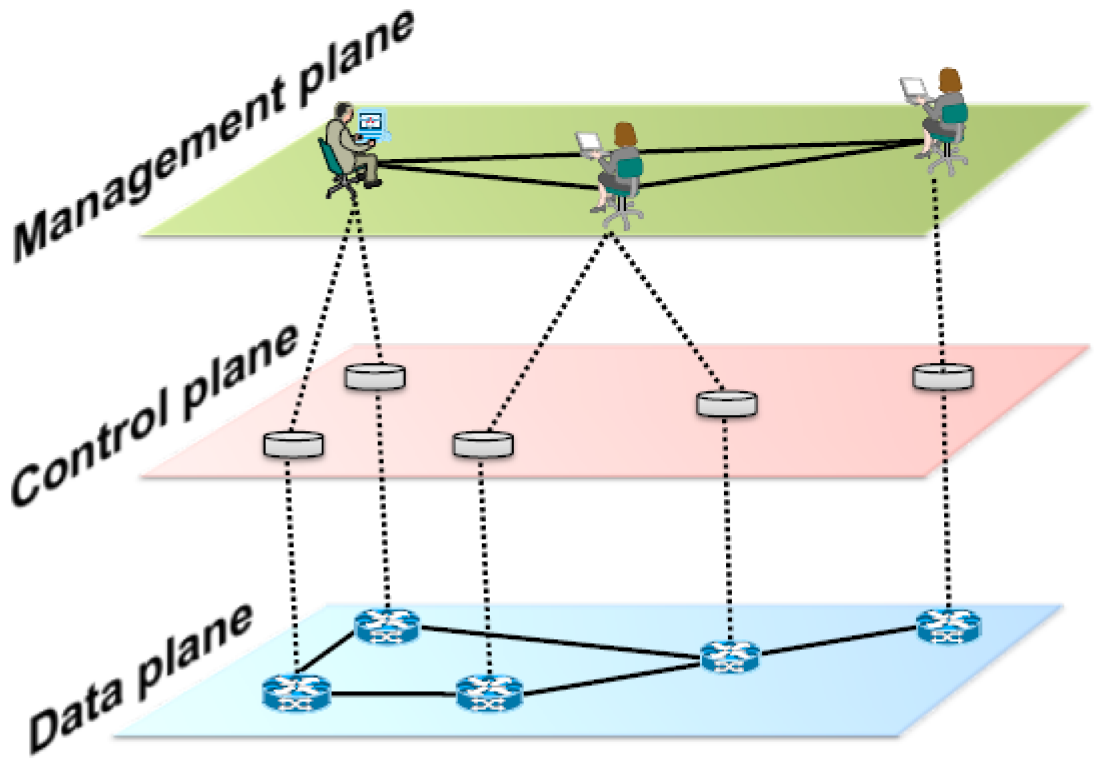


Figura 1 – Representação dos Planos de Rede / Fonte: (KREUTZ et al., 2015)

2.2 Modelo Tradicional

No modelo tradicional, os Plano de Dados e de Controle são unificados, visto que essa união foi considerada extremamente importante no *design* da rede porque era possível garantir a integridade da mesma.

O ponto fraco dessa unificação é a dificuldade de gerenciamento da rede, pois se faz necessária a configuração manual e individual dos dispositivos físicos da rede, além de não existir uma reconfiguração automática em caso de falhas ou mecanismos de resposta, tornando assim o ambiente pouco dinâmico (BENSON; AKELLA; MALTZ, 2009).

Outro aspecto negativo da implementação atual é a baixa flexibilidade da rede bem como o impedimento da inovação e evolução da infraestrutura de sua estrutura. Um exemplo é a demora na transição do protocolo IPV4 para o IPV6 já que se trata apenas de uma atualização de protocolo e está em curso a mais de 10 anos (KREUTZ et al., 2015).

2.3 Redes SDN

Um dos pontos centrais de Redes Definidas por *Software* é a separação dos planos de Dados e Controles em unidades distintas. Os detalhes dessa partição ainda serão explicados, mas em suma, o Plano de Dados - *switches* - se torna um conjunto de dispositivos de repasse enquanto o Plano de Controle é logicamente centralizado - com a possibilidade de ser fisicamente distribuído - e pode ser programado e reprogramado a qualquer momento. A comunicação entre esses planos é realizada por meio de APIs (*Application Programming Interface* - Interface de Programação de Aplicações), sendo o OpenFlow o software mais popular.

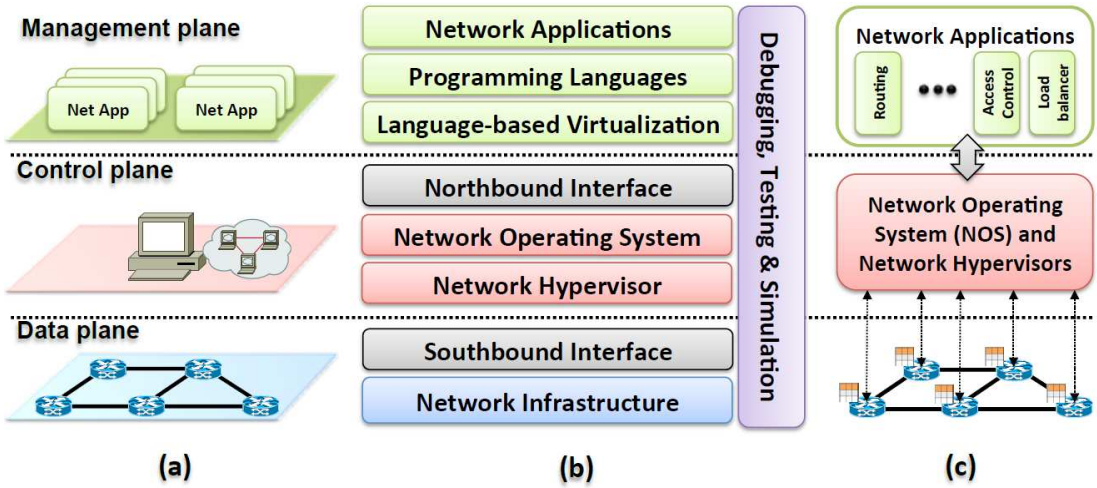
Redes SDN são construídas sobre quatro pilares:

- Separação dos Planos de Controle e Dados;
- Decisões sobre o envio de dados na rede são tomadas baseadas em fluxo e não em destino. Fluxo é amplamente definido como um conjunto de pacotes atuando sob um critério de correspondência e um conjunto de ações. Especificamente no OpenFlow, fluxo é uma sequência de pacotes entre uma origem e um destino, que possuem políticas de serviço para envio idênticas. Isto permite unir o comportamento de diferentes tipos de dispositivos resultando em uma flexibilidade extremamente alta, limitada apenas pela capacidade de implementação das tabelas de fluxo (MCKEOWN et al., 2008);
- Controle lógico movido para uma unidade externa denominada Controlador ou NOS (*Network Operating System* - Sistema Operacional da Rede). O NOS é uma plata-

forma de *software* que provê recursos essenciais e abstrações para facilitar a programação de dispositivos de repasse baseados na centralização lógica. Esse Controlador funciona similarmente a um Sistema Operacional tradicional;

- A rede é programável, pois possui aplicações rodando no topo do NOS e interagindo com os dispositivos físicos. É considerada por muitos como o pilar principal de Redes Definidas por *Software*.

Para a implementação desses pilares, a topologia da rede foi dividida em oito camadas conforme exemplificado na Figura 2.



Software-Defined Networks in (a) planes, (b) layers, and (c) system design architecture

Figura 2 – Topologia de Redes SDN / Fonte: (KREUTZ et al., 2015)

2.3.1 Camada 1 - Infraestrutura

A diferença principal nessa camada é o fato dos elementos físicos serem apenas dispositivos de repasse sem nenhum tipo de controle embutido. Tendo isto em vista, as redes são construídas sobre interfaces padrões e abertas - como é o caso do OpenFlow - permitindo a compatibilidade de configuração e comunicação entre diversos tipos de dispositivos de dados e controle. Esses dispositivos são baseados em um pipeline de tabelas de fluxo, cada uma delas divididas em três partes: regras de correspondência, ações que devem ser executadas em caso de correspondência e contadores que medem estatísticas de correspondência. Essa equivalência pode ser obtida entre diferentes campos como a porta do *switch*, o endereço MAC (*Media Access Control*) de destino ou origem, dentre outros.

As regras de fluxo no OpenFlow possuem um caminho pré-determinado a ser percorrido. Quando um pacote chega, o mesmo inicia um processo de varredura das tabelas de fluxo existentes a fim de encontrar uma combinação. Esse processo termina quando existe uma correspondência ou quando todas as tabelas forem analisadas. Caso não exista

uma regra específica para esse pacote, ele pode ser descartado ou ser encaminhado para o controlador, dependendo da opção padrão que foi configurada. Várias ações podem ser tomadas de acordo com o estado do pacote como, por exemplo, enviar para uma porta de saída, descartar, pular para a próxima tabela de fluxo ou para tabelas especiais são as mais comuns (KREUTZ et al., 2015).

2.3.2 Camada 2 - *Southbound Interfaces*

Também conhecidas por *Southbound* APIs, são as pontes que conectam os elementos de controle e de repasse, instrumentos cruciais na separação dos planos de Dados e Controle. Entretanto, essas APIs ainda estão fortemente amarradas aos dispositivos de repasse da camada física.

Southbound APIs ainda representam a maior barreira na introdução e aceitação de muitas tecnologias de redes, pois o tempo para a comercialização de um novo *switch* desenvolvido é muito grande assim como seu tempo de implementação, tornando-se um grande investimento e de alto risco. Se essas APIs não forem projetadas de forma eficiente, todo o investimento é perdido. Pensando nisso, foi criado o OpenFlow que promove uma interoperabilidade de diferentes dispositivos (KATO et al., 2013).

O protocolo OpenFlow possui três fontes de informações. A primeira é um evento baseado na ativação de alguma porta, ou seja, quando uma porta é ativada, uma mensagem é repassada. A próxima é obtida através de estatísticas de fluxo que são geradas pelos dispositivos de repasse e capturadas pelo controlador. E a última são mensagens enviadas para o controlador quando não há correspondência na tabela de repasse ou quando está explícito na mensagem recebida (KREUTZ et al., 2015).

2.3.3 Camada 3 - *Network Hypervisors*

Hypervisors são utilizados em redes que possuem grande parte de seus dispositivos alocados em máquinas virtuais. Eles permitem que máquinas distintas compartilhem os mesmos recursos. Um exemplo disso a infraestrutura da nuvem, onde cada usuário tem seus próprios recursos virtuais, mas esses são oferecidos de um único espaço de armazenamento (KREUTZ et al., 2015).

O propósito principal é a execução de aplicações desenvolvidas em diferentes linguagens de programação ou conceitos de diversas plataformas de controle, oferecendo interoperabilidade e portabilidade (JIN; REXFORD; WALKER, 2014).

2.3.4 Camada 4 - Sistema Operacional da Rede

Nos ambientes de redes tradicionais, o gerenciamento e a configuração do sistema são feitos utilizando *lower level* - conjunto de instruções específicas e sistemas operacio-

nais de redes proprietárias fechadas. A ideia de sistemas abstratos é inexistente, por isso os *designers* precisam lidar com protocolos de roteamento com algorítmos distribuídos complexos (KREUTZ et al., 2015).

O uso de SDN facilita essa administração centralizando o gerenciamento lógico da rede com a utilização do NOS. Por sua vez, o NOS oferece serviços como funcionalidades genéricas, estado e topologia da rede, descobrimento de novos dispositivos e configuração de redes distribuídas, sendo assim um ponto crucial de SDN (GUDE et al., 2008).

2.3.5 Camada 5 - *Northbound Interfaces*

Ainda não existem muitas *Northbound Interfaces* e as existentes possuem suas próprias definições, ou seja, cada controlador possui sua própria interface. Pode-se dizer que é o gargalo desta tecnologia.

Para a evolução de Redes SDN é necessária uma padronização que, por sua vez, deve conter uma abstração que permita as aplicações de redes não dependerem de implementações específicas promovendo portabilidade e interoperabilidade entre diferentes controladores (KREUTZ et al., 2015).

2.3.6 Camada 6 - Virtualização Baseada em Linguagem

A Virtualização possui duas características essenciais: capacidade de modularização e possibilidade de utilização de diferentes tipos de abstrações ao mesmo tempo em que assegura que as propriedades desejadas, tais como proteção, não sejam desprezadas.

Quando bem aplicadas, técnicas de virtualização podem proporcionar diferentes visões de uma única infraestrutura, simplificando a tarefa dos desenvolvedores de aplicações já que não é mais necessário preocupar-se com pequenos detalhes.

Especificamente no OpenFlow, é utilizado um controlador para gerenciar a infraestrutura inferior aplicando regras de fluxo de tabelas de rotas para criar uma rede virtual isolada oferecendo suporte para diferentes tecnologias, tornando-se uma ponte para redes heterogêneas (KREUTZ et al., 2015).

2.3.7 Camada 7 - Linguagens de Programação

Bem como no desenvolvimento de aplicações em geral, as linguagens de programação de redes evoluíram de linguagem de máquina / *lower-levels* para linguagens *high-level*, pois empregava-se mais tempo em pequenos detalhes de implementação do que solucionando o problema. Outra vantagem é a possibilidade de modularização e reutilização de códigos existentes em novos projetos (KREUTZ et al., 2015).

Linguagens de programação podem ser utilizadas para diferentes propósitos, dentre os quais se destacam tanto o gerenciamento de instalação de regras, atualização de contadores e obtenção de respostas quanto o desenvolvimento de linguagens abstratas para a criação e escrita de programas para topologias virtuais (semelhante a orientação à objeto) (REICH et al., 2013).

2.3.8 Camada 8 - Aplicações de Rede

As aplicações de rede podem ser vistas como o cérebro da rede, pois implementam o controle lógico que será traduzido em comandos que devem ser instalados no Plano de Dados, ditando o comportamento dos dispositivos de repasse.

A arquitetura SDN pode ser aplicada em qualquer ambiente, variando desde redes domésticas/empresariais até *data-centers* e pontos de troca da *Internet*, visto que oferece uma grande diversidade de benefícios. Esses benefícios podem ser aproveitados em aplicações que funcionem como roteador ou balanceador de carga, para segurança, recuperação de falhas e controle de qualidade de serviço fim-a-fim e até como gerenciador móvel de redes *wireless* (REINECKE, 2014).

As aplicações de rede podem ser divididas em cinco categorias:

- Engenharia de Tráfego visando minimizar o consumo de energia, maximizar a utilização da rede global e providenciar um balanceamento de carga otimizado;
- Mobilidade/*Wireless*, pois os planos de controle atuais estão abaixo do ideal nos quesitos de gerenciamento de espectro limitado, alocação de rádio-recursos e implementação de mecanismos de entrega;
- Medida/Monitoramento tanto para funções que fornecem novas funcionalidades para outros dispositivos quanto para propostas que visam melhorar as características SDN baseadas no OpenFlow como, por exemplo, a sobrecarga do computador na coleta de dados (KREUTZ et al., 2015);
- Segurança/Confiança;
- Gerenciamento de rede de *data-centers* tanto na prevenção de falha eminente quanto em uma possível migração de rede (AREFIN et al., 2013), (KELLER et al., 2012).

Com esses conceitos definidos, serão mostrados agora alguns trabalhos realizados na área de redes definidas por *software*.

2.4 Trabalhos Correlatos

Os simuladores utilizados nesse trabalho possuem várias áreas de aplicação dentro do ambiente SDN independente de atuarem juntos ou separadamente, conforme demonstrado a seguir.

Neto (2014) faz uso tanto do Mininet quanto do Floodlight para simulação da rede e do controlador para realizar testes sobre os principais depuradores de redes SDN existentes no mercado afim de qualificá-los em diferentes aspectos como, por exemplo, eficiência e usabilidade.

Um bom caso de uso dos simuladores trabalhando separamente é o trabalho de Araújo (2013), já que o mesmo utiliza apenas o Floodlight, pois é usado o NetFPGA para o Plano de Dados. O trabalho em questão tem por finalidade oferecer qualidade de serviço em redes baseadas em OpenFlow.

O trabalho anterior também demonstra o potencial do simulador do controlador, visto que NetFPGA é um *hardware*, ou seja, o Floodlight não depende apenas de ambientes virtuais para funcionar.

No que diz respeito a cenários de implementação, Costa (2013) trata algumas implementações simples com o intuito de apresentar os componentes da rede de forma mais detalhada. Um desses cenários assemelha-se muito com a prática laboratorial que será desenvolvida nesse trabalho. Trata-se do cenário criado para o teste do componente Roteador e possui três *switches* OpenFlow ligados entre si e com dois *hosts* em cada subrede. Um exemplo desse cenário pode ser visto na Figura 3.

Tendo em vista essas ideias, no próximo capítulo será apresentado o desenvolvimento desse trabalho através da implementação da prática laboratorial.

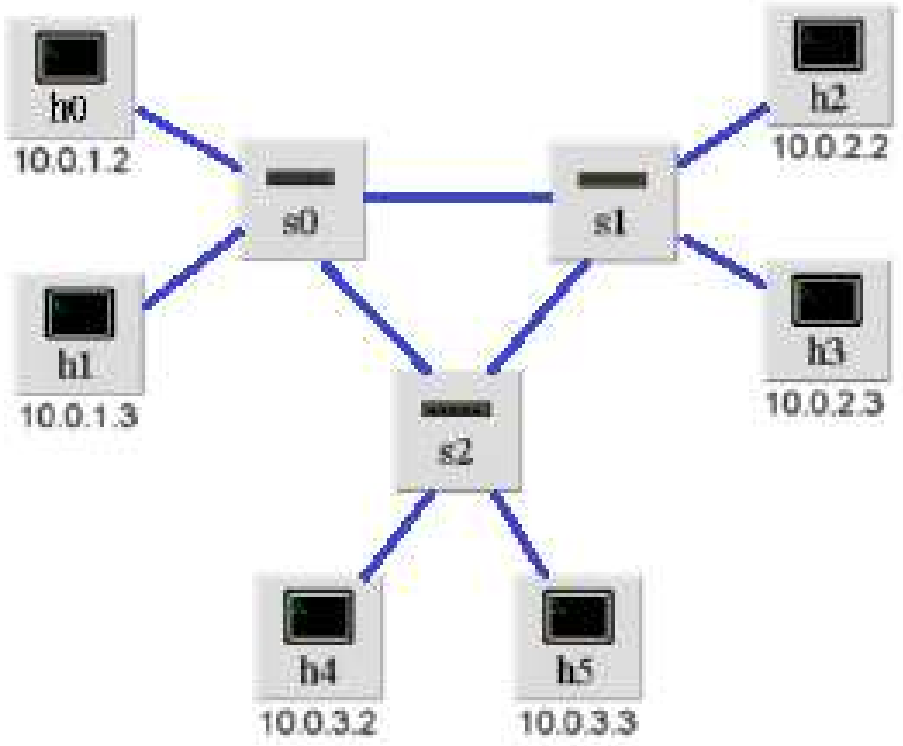


Figura 3 – Cenário de Implementação: 3 *switches* - 6 *hosts* / Fonte: (COSTA, 2013)

3 Desenvolvimento

Nesse capítulo são apresentadas informações a respeito do desenvolvimento das práticas laboratoriais tais como o ambiente, tecnologia e detalhes sobre as topologias utilizadas nesse projeto.

3.1 Ambiente

É possível executar as práticas em um dispositivo físico, mas devido ao tempo necessário de configuração entre os cenários propostos e também pelo fato de uma rede emulada ser de mais fácil acesso ao aluno, pode-se efetuar testes de qualquer lugar, facilitando o aprendizado.

Para virtualizar a rede foi utilizado o Mininet, um emulador de rede com suporte a OpenFlow, capaz de criar *hosts*, *switches*, controladores e *links*.

Com o Mininet é possível criar uma rede e adicionar uma grande quantidade de nós em um só computador, como também é possível conectar com outras redes de outros computadores. Outro fator importante na escolha do Mininet é disponibilidade de uma VM já configurada com o OpenFlow, POX e Wireshark (MININET TEAM, 2016).

3.2 Controlador

POX é uma plataforma para criação de controladores de rede escrito em Python, que possui ampla interação com o OpenFlow para ajudar no desenvolvimento de Redes SDN.

O POX possui alguns controladores nativos, dentre os quais será utilizado nesse trabalho o *forwarding.l2_learning* que foi desenvolvido para fins didáticos.

Outra funcionalidade é a opção de acrescentar componentes, tal como, o *openflow.spanning_tree* que cria uma *Spanning Tree* da rede que será executada uma vez que o controlador escolhido não possui suporte a rede com *loops* (NOXREPO, 2012).

3.3 Análises

A análise sobre o envio de pacotes pela rede foi realizada através do Wireshark, um dos mais famosos nessa área por analisar microscopicamente a rede, sendo amplamente utilizado tanto em empresas como em instituições de ensino.

Possui várias aplicações, sendo as mais úteis nesse projeto, a verificação de origem/destino dos pacotes bem como por quais portas / *switches* os dados percorreram (WIRESHARK FOUNDATION, 2016).

3.4 Topologias

A seguir serão explicadas as topologias desenvolvidas nos laboratórios, assim como o motivo de escolha de cada uma.

3.4.1 Linear

Este é um cenário padrão do Mininet, o mais simples deles. Foi inserido nesse trabalho para facilitar o entendimento sobre o funcionamento do Mininet, visto que grande parte dos alunos estarão tendo um primeiro contato com o mesmo.

A rede nesse cenário nada mais é do que dois *hosts* ligados a um *switch*.

Figura 4 – Topologia Linear

3.4.2 Árvore

Mais um cenário padrão, a fim de reforçar a compreensão do ambiente. Esse cenário retrata uma árvore de *switches* ligados a dois *hosts* cada. Nele é possível realizar testes entre dois *hosts* de um mesmo *switch* e entre dois *hosts* de *switches* diferentes, com os dados percorrendo a árvore. Ver Figura 5.

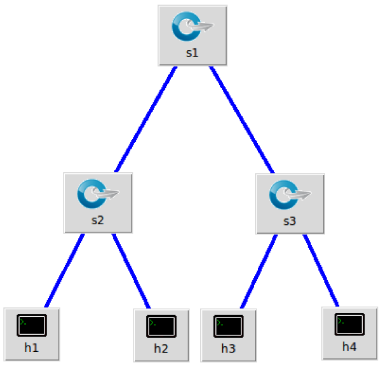


Figura 5 – Topologia em Árvore

3.4.3 Anel

É um cenário customizado interligando cinco *switches* em forma de anel, cada um ligado a dois *hosts*. Nele se faz necessária a utilização do controlador sobre a rede. Essa topologia é similar a uma já existente em uma prática laboratorial sobre o protocolo de roteamento OSPF (Open Shortest Path First) e tem como objetivo comparar os resultados desse teste com os já vistos anteriormente no Laboratório de Redes. Ver Figura 6.

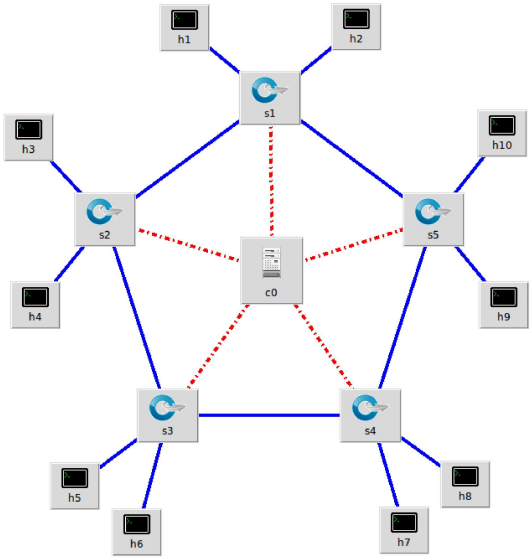


Figura 6 – Topologia em Anel

3.4.4 Diamante

É outro cenário customizado, dessa vez em formato de diamante, com o intuito de demonstrar o funcionamento de Redes SDN sobre multicaminhos. Também é necessária a utilização do controlador. Ver Figura 7.

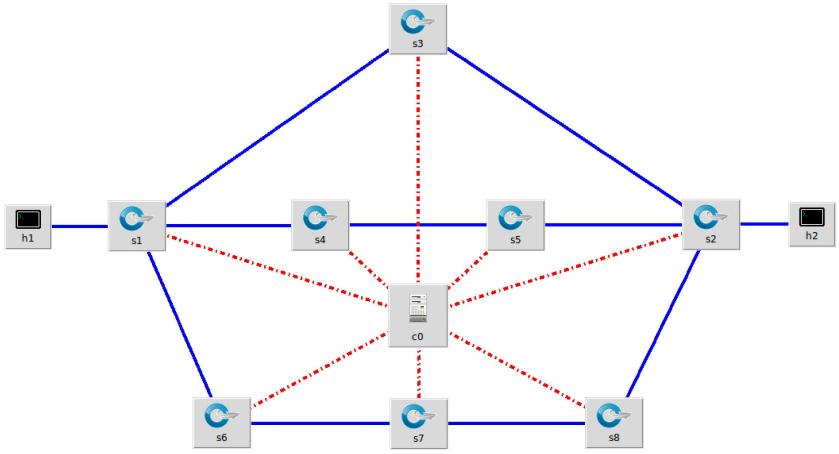


Figura 7 – Topologia em Diamante

4 Resultados

Este capítulo visa conectar a definição de Redes SDN com as práticas laboratoriais propostas, através da identificação de quais pontos foram abordados em cada aula.

4.1 Plano de Dados

A primeira prática tem como objetivo explicar a instanciação e o funcionamento do Plano de Dados. Analisando, por exemplo, o código da topologia Linear percebe-se que não há nenhuma lógica e/ou controle embutidos no mesmo. Trata-se apenas da criação e conexão dos dispositivos de repasse (*switches* e *hosts*) conforme exposto em 2.3.1. O mesmo acontece em todos os cenários utilizados nesse trabalho.

```
class SingleSwitchTopo( Topo ):
  #Single switch connected to k hosts.

  def build( self, k=2, **_opts ):
    #k: number of hosts
    self.k = k
    switch = self.addSwitch( 's1' )
    for h in irange( 1, k ):
      host = self.addHost( 'h%s' % h )
      self.addLink( host, switch )
```

Ao executar qualquer uma dentre as topologias citadas anteriormente obtém-se uma estrutura similar a da Figura 8, retirada a partir da execução do Cenário 2 por possuir mais nós na rede.

```
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth2
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s3-eth3
s2 lo:  s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2
c0
mininet> links
s1-eth1<->s2-eth3 (OK OK)
s1-eth2<->s3-eth3 (OK OK)
s2-eth1<->h1-eth0 (OK OK)
s2-eth2<->h2-eth0 (OK OK)
s3-eth1<->h3-eth0 (OK OK)
s3-eth2<->h4-eth0 (OK OK)
```

Figura 8 – Configuração da rede no Cenário 2 (Árvore)

4.2 Plano de Controle

Por sua vez a segunda aula tem o propósito de expor os conceitos do Plano de Controle. O controlador utilizado ao ser inicializado já executa uma série de procedimentos para registrar todos os nós e suas respectivas conexões como é mostrado na Figura 9 e Figura 10.

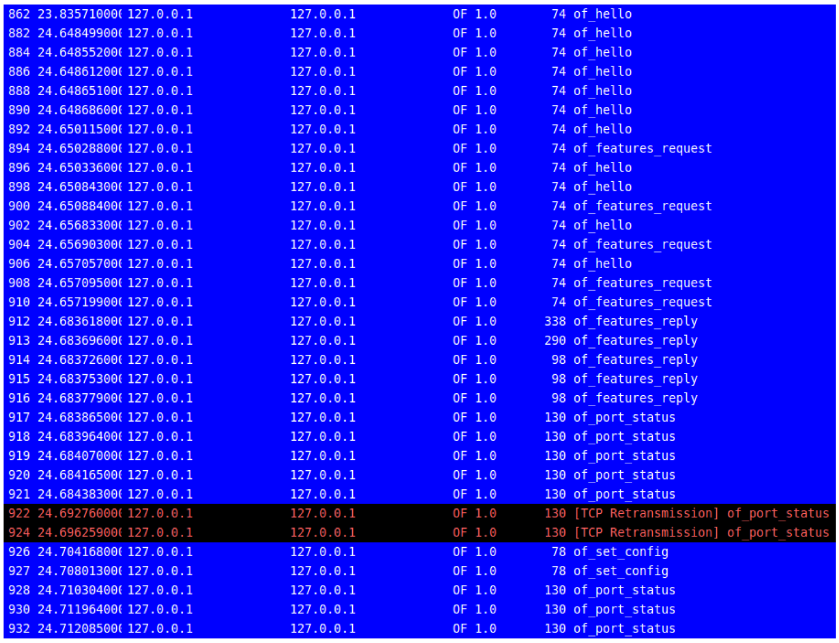


Figura 9 – Inicialização do Controlador da rede (Parte 1)

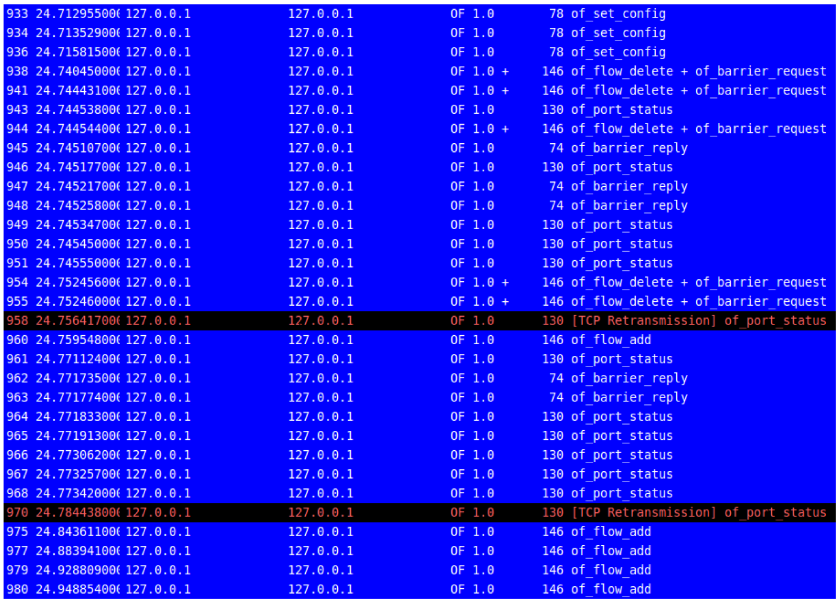


Figura 10 – Inicialização do Controlador da rede (Parte 2)

O cenário utilizado nesse teste foi a topologia em Anel. Pode-se perceber a inicialização dos 5 *switches* bem como a interação deles com o controlador através dos tipos

de pacotes enviados nessa comunicação (*of_features_request* e *of_features_reply*). Em seguida são atualizados os *status* das portas e as tabelas de rotas.

Após essa configuração a rede entra em estado de espera, apenas enviando pacotes para manter a conexão ativa e aguardando o recebimento de alguma operação. O controlador ao receber um pacote inicia uma uma série de procedimentos listados a seguir:

1. Atualiza a tabela de rotas com o endereço e a porta do switch.
2. Ethertype é LLDP ou a rede possui filtros?
    a) Sim? Descarta o pacote.
3. É multicast?
    a) Sim? Inunda a rede.
4. O destino está na tabela?
    a) Não? Inunda a rede.
5. Origem está na mesma porta que o destino?
    a) Sim? Descarta o pacote.
6. Atualiza a tabela com o destino
    a) Envia o pacote.

No trecho de código do controlador escrito abaixo é perceptível cada um desses procedimentos do controlador.

```
self.macToPort[packet.src] = event.port # 1
if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or
  packet.dst.isBridgeFiltered():
    drop() # 2a
return
if packet.dst.is_multicast:
  flood() # 3a
else:
  if packet.dst not in self.macToPort: # 4
flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
  else:
port = self.macToPort[packet.dst]
if port == event.port: # 5
```

```
  # 5a
  log.warning("Same port from %s -> %s on %s.%s.  Drop."
  % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
  drop(10)
  return
# 6
log.debug("installing flow for %s.%i -> %s.%i" %
  (packet.src, event.port, packet.dst, port))
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, event.port)
msg.idle_timeout = 10
msg.hard_timeout = 30
msg.actions.append(of.ofp_action_output(port = port))
msg.data = event.ofp # 6a
self.connection.send(msg)
```

Caso fosse necessário realizar alguma modificação na rede esse controlador poderia ser atualizado, se a alteração fosse muito ampla seria possível utilizar um novo controlador, tudo sem modificar a infraestrutura da rede de acordo com o disposto em 2.3.4. Este é um dos motivos das redes SDN estarem em evidência nos últimos anos.

Mais detalhes sobre a implementação e execução dessas práticas laboratoriais se encontram nos Anexos.

5 Conclusão

Como já explicado anteriormente, a Rede Definida por Software é uma tecnologia atual que ganhou muitos adeptos nos últimos anos devido, principalmente, a simplicidade na configuração / reconfiguração da rede e por ser altamente flexível.

Nesse contexto as aulas práticas criadas visam introduzir os alunos nesse ambiente recente, explicando de forma simples seus principais componentes. Outro fator que pode despertar interesse dos alunos nessa área é a existência do ambiente virtualizado por ser de fácil acesso e também pelo fato de ser possível testar qualquer configuração de rede em um comando ou dois ao invés de ficar trabalhando com uma grande quantidade de fios.

Novos trabalhos nessa área podem ser desenvolvidos de diversas formas tais como:

- Cenários mais complexos através da criação de topologias com uma grande quantidade de nós e *links*;
- Diferentes tipos de controladores podendo utilizar algum já existente ou criar um novo seguindo determinadas características;
- Separando os *switches* VMs distintas, ou seja, cada grupo cria uma máquina virtual contendo apenas um *switch* em seu próprio computador e conectam-se os computadores de forma a se respeitar um dos cenários propostos acima;
- Utilizando um *switch* SDN físico presente no Laboratório de Redes da UFU;
- e mais.

Portanto, percebe-se que essa é uma área que tem muito a crescer e possui múltiplas áreas de atuação facilitando a criação de novos trabalhos.

Referências

ANTONENKO, V.; SMELYANSKIY, R. Global network modelling based on mininet approach. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, n. 13, p. 145–146, 2013. Citado na página 11.

ARAúJO, M. R. A. G. d. Uma abordagem para aprovisionamento de qos em redes definidas por software baseadas em openflow. 2013. Citado na página 20.

AREFIN, A. et al. Diagnosing data center behavior flow by flow. *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, n. 13, p. 11–20, 2013. Citado na página 19.

BENSON, T.; AKELLA, A.; MALTZ, D. Unraveling the complexity of network management. *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, n. 09, p. 335–348, 2009. Citado na página 15.

COSTA, L. R. Openflow e o paradigma de redes definidas por software. 2013. Citado 3 vezes nas páginas 7, 20 e 21.

GUDE, N. et al. Nox: Towards an operating system for networks. *Computer Communication Review*, n. 38, p. 105–110, 2008. Citado na página 18.

JIN, X.; REXFORD, J.; WALKER, D. Incremental update for a compositional sdn hypervisor. *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, n. 14, p. 187–192, 2014. Citado na página 17.

KATO, T. et al. Case study of applying sple to development of network switch products. *Proceedings of the 17th International Software Product Line Conference*, n. 13, p. 198–207, 2013. Citado na página 17.

KELLER, E. et al. Live migration of an entire network (and its hosts). *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, n. 11, p. 109–114, 2012. Citado na página 19.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, n. 103, p. 14–76, 2015. Citado 8 vezes nas páginas 7, 11, 14, 15, 16, 17, 18 e 19.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *Computer Communication Review*, v. 2, n. 38, p. 69–74, 2008. Citado na página 15.

MININET TEAM. In: _____. *Documentation*. 2016. Disponível em: <http://mininet.org>. Citado na página 22.

NETO, H. S. Análise de depuradores para redes. 2014. Citado na página 20.

NOXREPO. In: _____. *POX Wiki*. [s.n.], 2012. Disponível em: <https://openflow.stanford.edu/display/ONL/POX+Wiki>. Citado 2 vezes nas páginas 11 e 22.

OPEN NETWORKING FOUNDATION. *Software-Defined Networking*: The new norm for networks. [S.l.], 2012. 12 p. Citado na página 11.

REICH, J. et al. Modular sdn programming with pyretic. *USENIX ;login*, v. 5, n. 38, 2013. Citado na página 19.

REINECKE, E. In: _____. *Mapping the Future of Software-Defined Networking*. [s.n.], 2014. Disponível em: <http://goo.gl/fQCvRF>. Citado na página 19.

WIRESHARK FOUNDATION. In: _____. *About*. 2016. Disponível em: <https://wireshark.org>. Citado na página 23.

Anexos

ANEXO A – Laboratório 01

Objetivo: Este laboratório experimenta os conceitos do Plano de Dados de Redes Definidas por *Software* (SDN) utilizando Mininet para simular sua estrutura e POX como controlador, todos sendo executados em uma máquina virtual.

A.1 Fundamentos Teóricos

Esta atividade tem como objetivo discutir os conceitos relacionados ao Plano de Dados das Redes Definidas por *Software* utilizando Mininet para experimentação. Para isso é importante relembrar os conceitos de SDN.

A característica principal de SDN é que o plano de dados e o plano de controle são separados, ou seja, os *switches* da rede se tornam apenas dispositivos de repasse enquanto o controle lógico se torna uma entidade externa centralizada simplificando as políticas de configuração e evolução, tornando a rede programável.

A.1.1 Plano de Dados

A diferença principal nesta camada é o fato dos elementos físicos serem apenas dispositivos de repasse sem nenhum tipo de controle embutido. Tendo isto em vista, as redes são construídas sobre interfaces padrões e abertas - como é o caso do OpenFlow utilizado neste experimento - permitindo a compatibilidade de configuração e comunicação entre diversos tipos de dispositivos de dados e controle.

As regras de fluxo no OpenFlow possuem um caminho pré-determinado a ser percorrido. Quando um pacote chega, o mesmo inicia um processo de varredura das tabelas de fluxo existentes a fim de encontrar uma combinação. Esse processo termina quando existe uma correspondência ou quando todas as tabelas forem analisadas. Caso não exista uma regra específica para esse pacote, ele pode ser descartado ou ser encaminhado para o controlador, dependendo da opção padrão que foi configurada.

A.2 Materiais e Métodos

Para o desenvolvimento dessa prática, os computadores disponíveis no Laboratório de Redes precisarão ter instalados um *Software* de emulação (Vmware ou VirtualBox) com a VM Mininet instanciada. A VM Mininet é um ambiente disponibilizado para *download* que já possui tanto o Mininet quanto o controlador POX configurados.

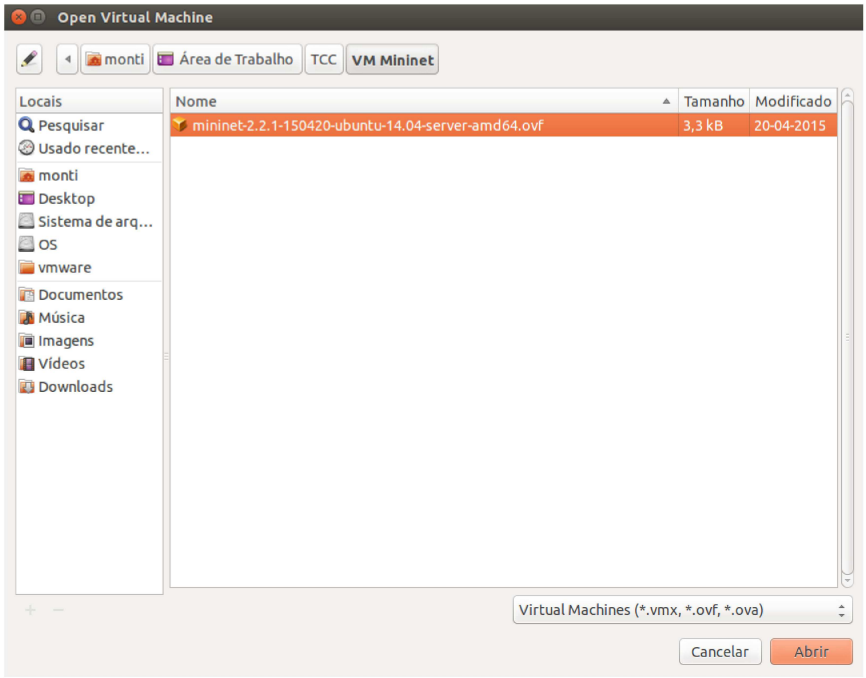
Descrição dos componentes:

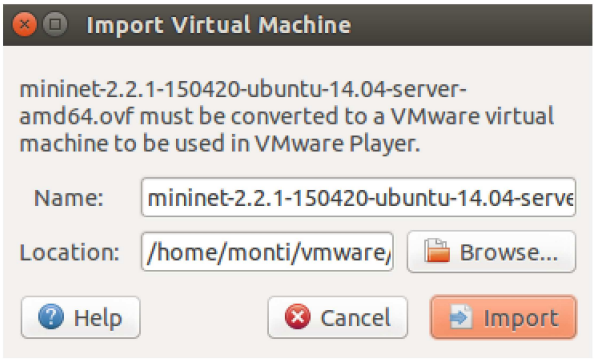
- Mininet: Ferramenta utilizada para simulação de redes SDN onde é possível instanciar *switches* e *hosts* dentre outros componentes. Com ele também é possível conectar os nós, criando assim a rede.
- POX: Controlador da rede. Será aprofundado na próxima prática.

Ambos os componentes possuem cenários básicos por padrão mas também podem ser customizados através da criação de *scripts* escritos em linguagem de programação Python.

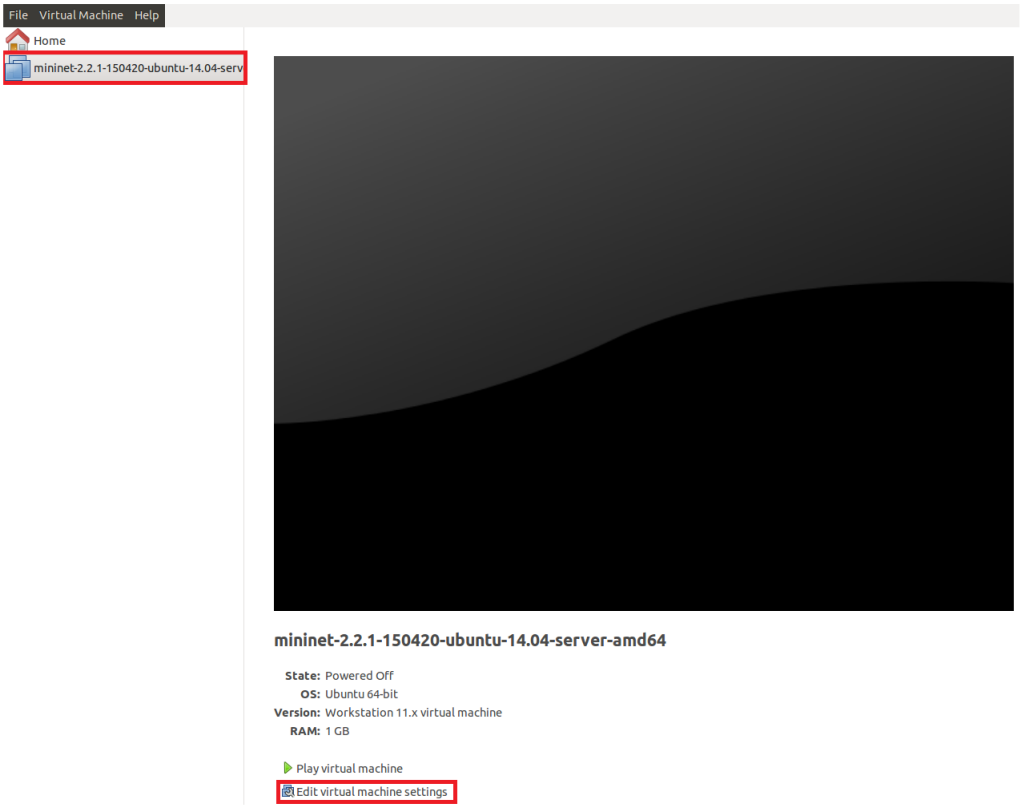
A.3 Configuração do Ambiente

Para execução dos testes devemos primeiro preparar o ambiente Mininet. Para isso, no VMware abra a imagem de da VM Mininet e em seguida escolha o local para importação conforme apresentado nas imagens a seguir.



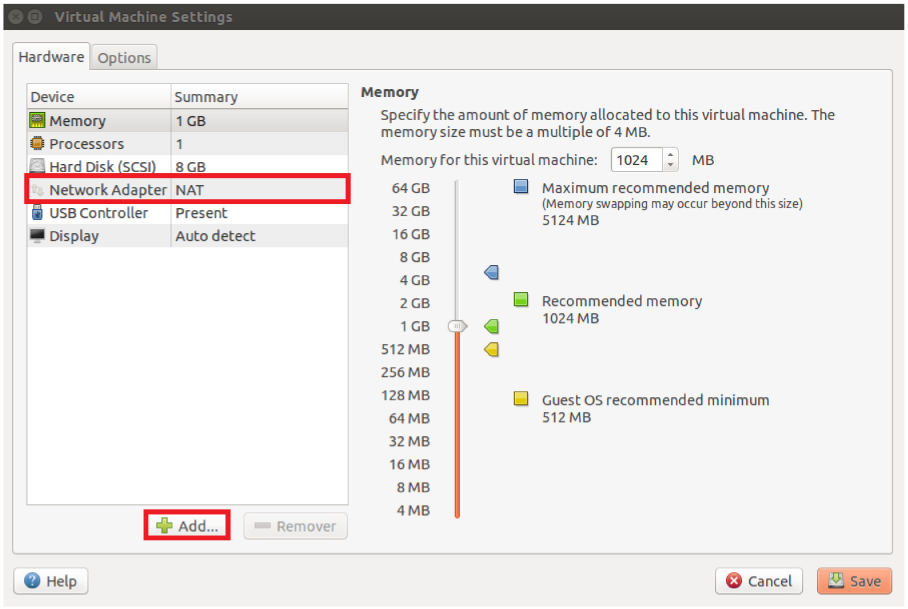


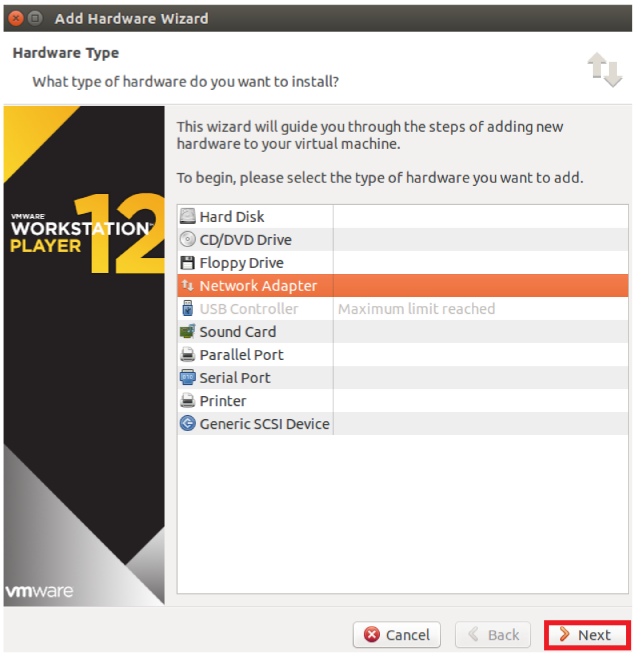
Após a conclusão da importação clique sobre a imagem gerada e abra o painel de configuração da máquina virtual.

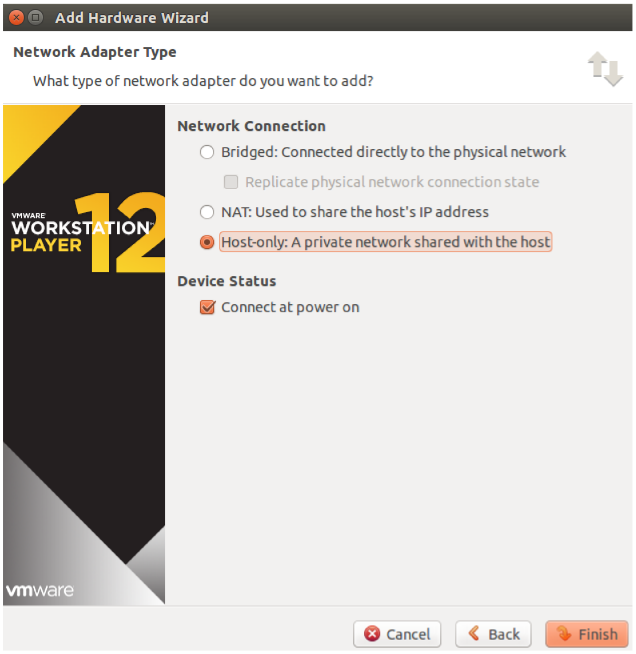


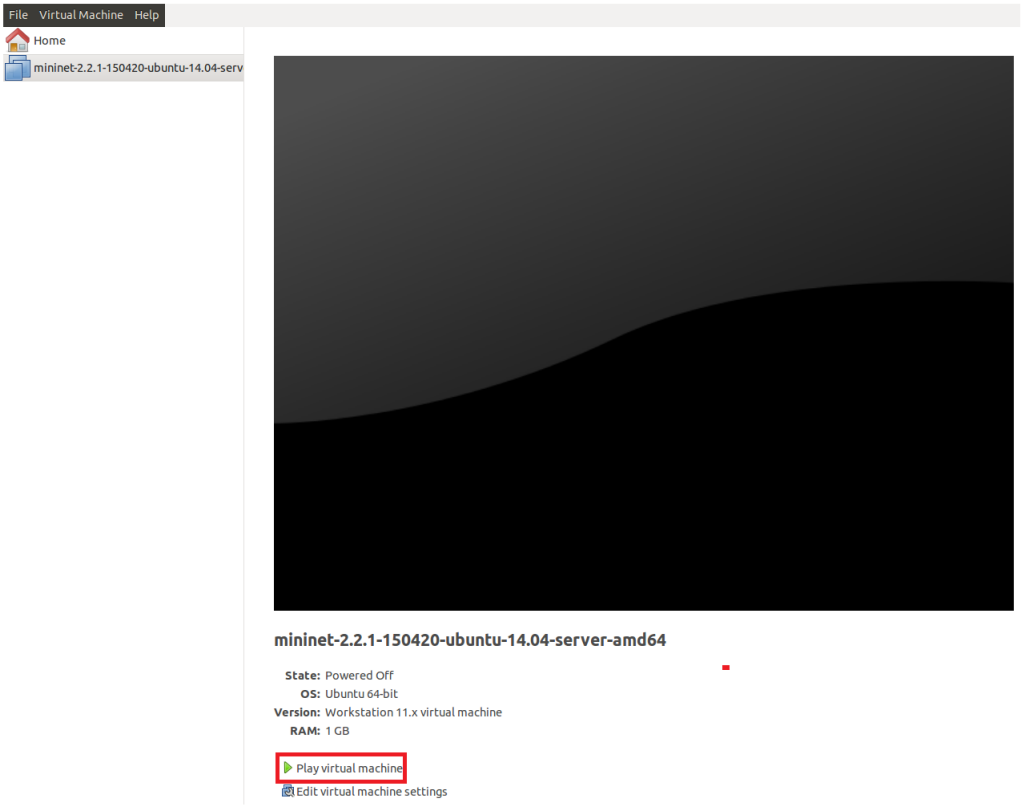
Nas configurações devemos nos certificar de ter duas interfaces de rede (NAT e Host-Only). No caso de alguma interface não estar devidamente configurada deve-se adicionar um novo componente.

No exemplo a seguir não há uma interface de rede Host-Only, para adicioná-la clique em +Add... , em seguida selecione Network Adapter, selecione a opção Host-only: A private network shared with the host e salve as configurações e inicie a máquina virtual.


Virtual Machine Settings
Hardware
Options
Device
Summary
Memory
1 GB
Processors
1
Hard Disk (SCSI)
8 GB
Network Adapter
NAT
USB Controller
Present
Display
Auto detect
Add...
Remover
Memory
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.
Memory for this virtual machine:
1024
MB
64 GB
32 GB
16 GB
8 GB
4 GB
2 GB
1 GB
512 MB
256 MB
128 MB
64 MB
32 MB
16 MB
8 MB
4 MB
Maximum recommended memory
(Memory swapping may occur beyond this size)
5124 MB
Recommended memory
1024 MB
Guest OS recommended minimum
512 MB
Help
Cancel
Save


Add Hardware Wizard
Hardware Type
What type of hardware do you want to install?
VMWARE
WORKSTATION
PLAYER
12
vmware
This wizard will guide you through the steps of adding new hardware to your virtual machine.
To begin, please select the type of hardware you want to add.
Hard Disk
CD/DVD Drive
Floppy Drive
Network Adapter
USB Controller
Maximum limit reached
Sound Card
Parallel Port
Serial Port
Printer
Generic SCSI Device
Cancel
Back
Next





Após iniciar a imagem, será solicitado um login e senha para continuar, em ambos os casos a resposta é mininet. Logo após digite no terminal da máquina virtual o seguinte comando ifconfig. Caso não seja exibida alguma interface digite os comandos sudo dhclient eth0 e sudo dhclient eth1 e anote o IP da interface eth1.

```
Ubuntu 14.04 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Mon Apr 20 07:11:15 PDT 2015 on ttyS0
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:f7:a0:f1
          inet addr:172.16.181.132  Bcast:172.16.181.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7115 (7.1 KB)  TX bytes:3380 (3.3 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3832 (3.8 KB)  TX bytes:3832 (3.8 KB)

mininet@mininet-vm:~$ _


mininet@mininet-vm:~$ sudo dhclient eth0
mininet@mininet-vm:~$ sudo dhclient eth1
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:f7:a0:f1
          inet addr:172.16.181.132  Bcast:172.16.181.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8797 (8.7 KB)  TX bytes:5222 (5.2 KB)

eth1      Link encap:Ethernet  HWaddr 00:0c:29:f7:a0:fb
          inet addr:172.16.204.130  Bcast:172.16.204.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:746 (746.0 B)  TX bytes:684 (684.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3832 (3.8 KB)  TX bytes:3832 (3.8 KB)

mininet@mininet-vm:~$
```

Essas são todas as operações executadas na máquina virtual, a partir de agora usaremos o terminal Ubuntu para iniciarmos uma conexão SSH com a VM. Isto se faz necessário pois pelo VMware ficamos restringidos à janela exibida no terminal, já que não é possível usar o *scroll* na tela. Outra vantagem de utilizarmos uma conexão SSH é que no Ubuntu é possível instalar e utilizar o editor de texto de sua preferência com interface gráfica.

Para realizar a conexão basta digitar o seguinte comando: ssh -X mininet@IP, onde esse IP se refere ao IP da interface *eth1* anotado anteriormente. Será solicitada uma senha para login que é mininet.

```
mininet@mininet-vm: ~
monti@gabriel:~$ ssh -X mininet@172.16.204.130
mininet@172.16.204.130's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun May 22 19:05:44 2016 from 172.16.204.1
mininet@mininet-vm:~$
```

A.4 Cenário Mínimo (1 *switch* – 2 *hosts*)

Agora que estamos no ambiente Mininet podemos executar um teste em uma das arquiteturas básicas padrão para testarmos se as configurações estão corretas.

Para isso digite no terminal sudo mn --mac. Esse comando cria uma topologia mínina que consiste em 1 *switch* conectado à dois *hosts*. A opção *--mac* é útil pois inicializa os *hosts* com endereço MAC igual ao número do *host* (00:00:00:00:00:01 para o *host* 1). Também é inicializado um controlador genérico.

Em seguida, pode-se executar os comandos nodes, links, dump, pingall dentre outros. Esses comandos listam os nós, as conexões, as informações da rede e realiza os testes de todas as conexões. Para visualizar mais comandos digite help.

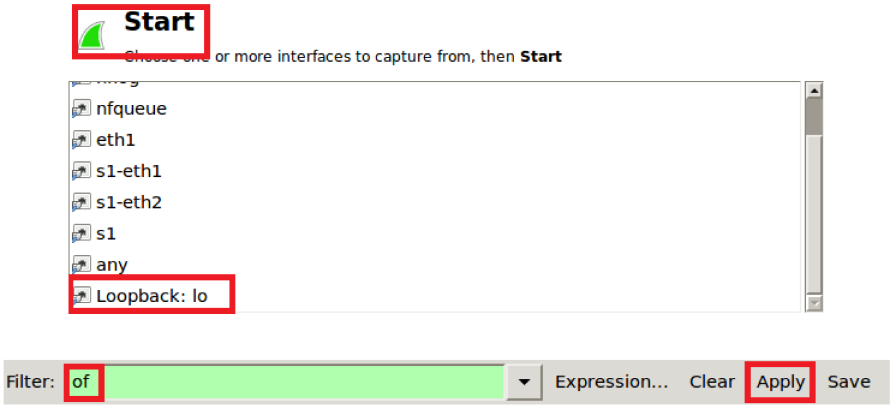
Segue a representação e o código Python dessa topologia executada:


```
class SingleSwitchTopo( Topo ):
  #Single switch connected to k hosts.

  def build( self, k=2, **_opts ):
    #k: number of hosts
    self.k = k
    switch = self.addSwitch( 's1' )
    for h in irange( 1, k ):
      host = self.addHost( 'h%s' % h )
      self.addLink( host, switch )
```

Após a execução dos comandos básicos e visualização do estado da rede deve-se analisar como é feito o envio de pacotes. Para isso abra em outro terminal uma nova conexão SSH e digite o comando sudo wireshark &. Em seguida selecione a opção Loopback: lo e clique em Start. Esse processo inicializará o envio de pacotes na rede e serão exibidos todos os tipos de pacotes.

Para facilitar o estudo deve-se aplicar um filtro para visualização apenas dos pacotes Openflow conforme mostrado na figura a seguir:



O próximo passo é executar o comando pingall no terminal principal. O Wireshark irá exibir todos os pacotes enviados para estabelecimento da conexão, procura pelo IP, resposta ao *ping* dentre outros. Ao final do comando a lista de chamadas ficará similar à imagem a seguir.

1219 9.203563000	00:00:00_00:00:01	Broadcast	OF 1.0	126 of_packet_in
1220 9.205431000	127.0.0.1	127.0.0.1	OF 1.0	90 of_packet_out
1222 9.205970000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.0	126 of_packet_in
1223 9.206992000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1224 9.208286000	10.0.0.1	10.0.0.2	OF 1.0	182 of_packet_in
1225 9.209232000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1226 9.210766000	10.0.0.2	10.0.0.1	OF 1.0	182 of_packet_in
1227 9.211321000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1228 9.219236000	10.0.0.2	10.0.0.1	OF 1.0	182 of_packet_in
1229 9.219611000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1230 9.220726000	10.0.0.1	10.0.0.2	OF 1.0	182 of_packet_in
1231 9.221675000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1265 13.973334000	127.0.0.1	127.0.0.1	OF 1.0	74 of_echo_request
1266 13.974686000	127.0.0.1	127.0.0.1	OF 1.0	74 of_echo_reply
1268 14.220792000	00:00:00_00:00:02	00:00:00_00:00:01	OF 1.0	126 of_packet_in
1269 14.221041000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add
1271 14.321258000	00:00:00_00:00:01	00:00:00_00:00:02	OF 1.0	126 of_packet_in
1272 14.321660000	127.0.0.1	127.0.0.1	OF 1.0	146 of_flow_add

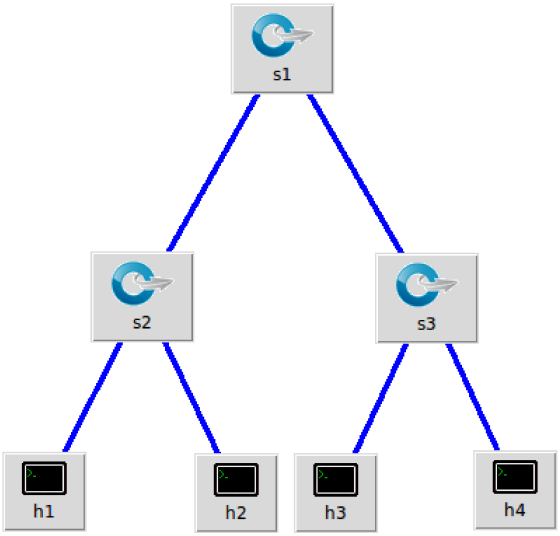
No início nenhum nó da rede conhece o outro nem sabe seu IP, por esse motivo o *host h1* envia um *broadcast* com seu MAC solicitando os IP dos nós conectados. O *host h2* envia seu IP para *h1* através de seu MAC (único conhecido até o momento). Então *h1* recebe a resposta com o IP de *h2*, salva em sua tabela de rotas e executa um *ping* nesse endereço.

Ao receber o *ping*, *h2* também salva o IP de *h1*. A próxima etapa do *pingall* é *h2* pingar *h1*. Como o IP já é conhecido, esse comando é executado imediatamente.

A.5 Cenário Básico (Árvore - 3 *switches* – 4 *hosts*)

Para isso digite no terminal sudo mn --mac --topo=tree,2,2. Esse comando cria uma topologia em árvore que consiste em 1 *switch* conectado à outros 2 *switches* com cada um desses ligados a 2 *hosts*.

Segue a representação e o código Python dessa topologia executada:




```
class TreeTopo( Topo ):
  #Topology for a tree network with a given depth and fanout.

  def build( self, depth=1, fanout=2 ):
    # Numbering: h1..N, s1..M
    self.hostNum = 1
    self.switchNum = 1
    # Build topology
    self.addTree( depth, fanout )

  def addTree( self, depth, fanout ):
    #Add a subtree starting with node n.
    isSwitch = depth > 0
    if isSwitch:
      node = self.addSwitch( 's%s' % self.switchNum )
      self.switchNum += 1
      for _ in range( fanout ):
        child = self.addTree( depth - 1, fanout )
        self.addLink( node, child )
    else:
      node = self.addHost( 'h%s' % self.hostNum )
      self.hostNum += 1
    return node
```

A.6 Exercício

A VM Mininet por padrão possui mais alguns cenários para teste. Cite 2 desses cenários e descreva seu funcionamento.

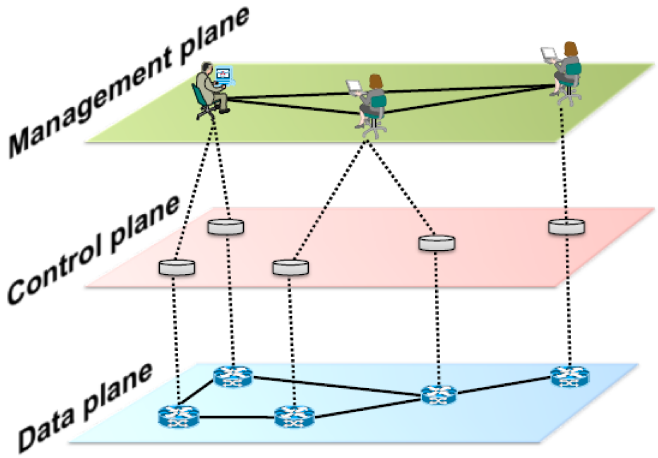
ANEXO B – Laboratório 02

Objetivo: Este laboratório experimenta os conceitos do Plano de Controle de Redes Definidas por *Software* (SDN) utilizando Mininet para simular sua estrutura e POX como controlador, todos sendo executados em uma máquina virtual.

B.1 Fundamentos Teóricos

Esta atividade tem como objetivo discutir os conceitos relacionados ao Plano de Controle das Redes Definidas por *Software* utilizando Mininet e POX para experimentação. Para isso é importante relembrar os conceitos de SDN.

A característica principal de SDN é que o plano de dados e o plano de controle são separados, ou seja, os *switches* da rede se tornam apenas dispositivos de repasse enquanto o controle lógico se torna uma entidade externa centralizada simplificando as políticas de configuração e evolução, tornando a rede programável.



B.1.1 Plano de Controle

Nos ambientes de redes tradicionais, o gerenciamento e a configuração do sistema são feitos utilizando um conjunto de instruções específicas e sistemas operacionais de redes proprietárias fechadas.

O uso de SDN facilita essa administração centralizando o gerenciamento lógico da rede com a utilização do NOS (*Network Operating System* - Sistema Operacional da Rede). Por sua vez, o NOS oferece serviços como funcionalidades genéricas, estado e topologia da rede, descobrimento de novos dispositivos e configuração de redes distribuídas, sendo assim um ponto crítico de SDN.

B.2 Materiais e Métodos

Para o desenvolvimento dessa prática, os computadores disponíveis no Laboratório de Redes precisarão ter instalados um *software* de emulação (Vmware ou VirtualBox) com a VM Mininet instanciada. A VM Mininet é um ambiente disponibilizado para *download* que já possui tanto o Mininet quanto o controlador POX configurados.

Descrição dos componentes:

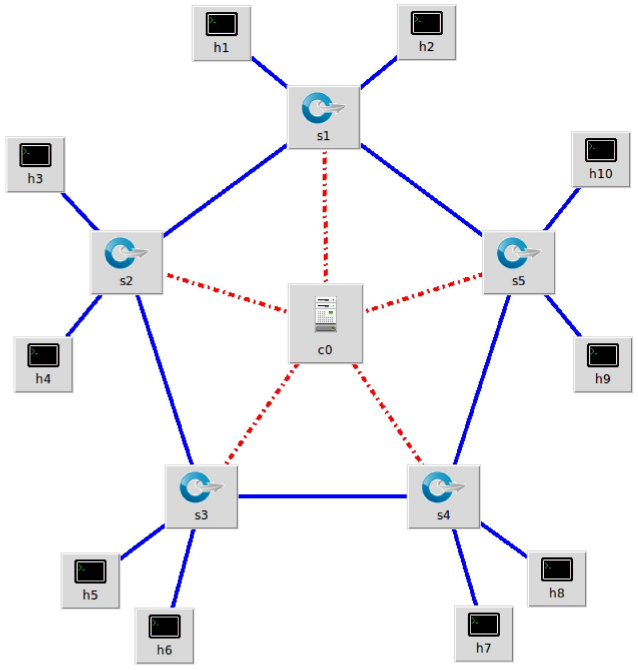
- Mininet: Especificado na aula anterior.
- POX: Controlador da rede, utilizado principalmente para fins educacionais (pesquisa e ensino). O POX possui vários componentes que ditam como será o funcionamento do controlador. Neste experimento utilizaremos o *forwarding.l2_learning*.

Ambos os componentes possuem cenários básicos por padrão mas também podem ser customizados através da criação de *scripts* escritos em linguagem de programação Python.

B.3 Topologia - Cenário Customizado (5 *switches* – 10 *hosts*)

Este cenário é constituído de um anel de 5 *switches* cada um ligado ao 2 *hosts*, ou seja, existe um *loop* na rede com dois caminhos distintos conectando dois nós diferentes. Nesse momento será utilizado também um controlador remoto ligado a todos os *switches*.

Segue a representação e o código dessa topologia:



```
from mininet.topo import Topo

class RingTopo( Topo ):
  def __init__( self ):
    # Initialize topology
    Topo.__init__( self )

  #Topology for a ring network with n switches.
  def build( self, switches=5, hosts=2 ):
    # Numbering:  h1..N, s1..M
    self.hostNum = 1
    self.switchNum = 1
    # Build topology
    self.addRing( switches, hosts )

  def addRing( self, switches, hosts ):
    for _ in range( switches ):
      switch = self.addSwitch( 's%s' % self.switchNum )
      self.switchNum += 1
      for _ in range( hosts ):
        host = self.addHost( 'h%s' % self.hostNum )
        self.hostNum += 1
        self.addLink( switch, host )
      if self.switchNum-1 != switches:
        switch2 = self.addSwitch( 's%s' % self.switchNum )
        self.addLink( switch, switch2 )
      else:
        self.addLink( switch, 's1' )
    return

topos = { 'topoRing': ( lambda: RingTopo() ) }
```

B.4 Controlador - *forwarding.l2_learning*

O funcionamento deste controlador é bem simples, observa o tráfego e preenche a tabela de rotas. Mas como isso é feito?

Quando um pacote chega na rede, sabe-se a qual porta ele pertence. Quando deseja-se enviar esse pacote é necessário consultar a tabela. Se a porta destino não é conhecida

é enviada uma mensagem para todas as portas exceto a porta de origem e, com base na resposta, a tabela de rotas é atualizada. Isto se torna um problema na presença de *loops*.

1. Atualiza a tabela de rotas com o endereço e a porta do switch.
2. Ethertype é LLDP ou a rede possui filtros?
    a) Sim? Descarta o pacote.
3. É multicast?
    a) Sim? Inunda a rede.
4. O destino está na tabela?
    a) Não? Inunda a rede.
5. Origem está na mesma porta que o destino?
    a) Sim? Descarta o pacote.
6. Atualiza a tabela com o destino
    a) Envia o pacote.

O problema desse controlador se deve ao fato do mesmo não aceitar *loops* na definição da rede. Esse problema pode ser contornado através da criação de uma *Spanning Tree* dos componentes presentes na rede para eliminação do *loop*.

B.5 Testes

Para execução dos testes dessa topologia serão necessários três terminais com conexões SSH para a VM Mininet, uma para criação da rede, uma para inicialização do controlador e outra para o Wireshark.

Primeiramente deve-se criar um arquivo com a topologia, visto que ela não é nativa da VM. Por convenção, códigos próprios devem ser criados na pasta *custom*. Esse arquivo pode ser criado da seguinte forma:

Em qualquer terminal digite pico mininet/custom/ring.py. Após copiar o código fornecido pressione CTRL+X, em seguida a tecla Y e depois ENTER.

O próximo passo é iniciar o controlador através do seguinte comando: sudo ~/pox/pox.py forwarding.l2_learning openflow.spanning_tree log.level --DEBUG samples.pretty_log openflow.discovery info.packet_dump.

Antes de prosseguir vamos entender o comando anterior.

- *forwarding.l2_learning*
  - É o tipo do controlador
- *openflow.spanning_tree*
  - É utilizado para criação da *spanning tree* visto que o exemplo contém *loops*
- *log.level --DEBUG*
  - É utilizado para uma visualização mais detalhada das informações produzidas pelo controlador
- *samples.pretty_log*
  - É utilizado para formatação das mensagens, tornando-as mais legível
- *openflow.discovery*
  - É utilizado para enviar e receber mensagens LLDP dos *switches* OpenFlow para descobrir a topologia da rede e detectar quando um *link* cai. Trabalha em conjunto com o *openflow.spanning_tree*
- *info.packet_dump*
  - É utilizado para exibir mensagens sobre os pacotes recebidos pelo controlador dos *switches*. Ajudará na visualização da interação do controlador com os *switches*

Com o controlador executando devemos iniciar a topologia através do comando: sudo mn --custom ~/mininet/custom/topo-ring.py --topo topoRing --switch ovsk --controller=remote.

A partir deste momento pode-se executar os comandos vistos na aula anterior para visualizar o estado da rede.

Após inicialização do controlador e criação da rede, pode-se observar a criação da *spanning tree* na janela do controlador. A cada execução, uma árvore diferente é criada. A seguir é mostrado um caso obtido.

```
mininet@mininet-vm:~$ sudo ~/pox/pox.py forwarding.l2_learning openflow.spanning_tree log.leve
l --DEBUG samples.pretty_log openflow.discovery info.packet_dump
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
[openflow.spanning_tree ] Spanning tree component ready
[info.packet_dump       ] Packet dumper running
[core                   ] POX 0.2.0 (carp) going up...
[core                   ] Running on CPython (2.7.6/Jun 22 2015 17:58:13)
[core                   ] Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
[core                   ] POX 0.2.0 (carp) is up.
[openflow.of_01         ] Listening on 0.0.0.0:6633
[openflow.of_01         ] [None 1] closed
[openflow.of_01         ] [00-00-00-00-00-04 5] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-04
[forwarding.l2_learning ] Connection [00-00-00-00-00-04 5]
[openflow.of_01         ] [00-00-00-00-00-03 6] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-03
[forwarding.l2_learning ] Connection [00-00-00-00-00-03 6]
[openflow.of_01         ] [00-00-00-00-00-05 2] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-05
[forwarding.l2_learning ] Connection [00-00-00-00-00-05 2]
[openflow.of_01         ] [00-00-00-00-00-02 3] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-02
[forwarding.l2_learning ] Connection [00-00-00-00-00-02 3]
[openflow.of_01         ] [00-00-00-00-00-01 4] connected
[openflow.discovery     ] Installing flow for 00-00-00-00-00-01
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 4]
[openflow.discovery     ] link detected: 00-00-00-00-00-04.4 -> 00-00-00-00-00-05.1
[openflow.spanning_tree ] Spanning tree updated
[openflow.discovery     ] link detected: 00-00-00-00-00-04.1 -> 00-00-00-00-00-03.4
[openflow.spanning_tree ] Spanning tree updated
[openflow.discovery     ] link detected: 00-00-00-00-00-03.1 -> 00-00-00-00-00-02.4
[openflow.spanning_tree ] Spanning tree updated
[openflow.discovery     ] link detected: 00-00-00-00-00-03.4 -> 00-00-00-00-00-04.1
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] 8 ports changed
[openflow.discovery     ] link detected: 00-00-00-00-00-05.1 -> 00-00-00-00-00-04.4
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] 5 ports changed
[openflow.discovery     ] link detected: 00-00-00-00-00-05.4 -> 00-00-00-00-00-01.4
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] 1 ports changed
[openflow.discovery     ] link detected: 00-00-00-00-00-02.1 -> 00-00-00-00-00-01.3
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-03 6]
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-04 5]
[openflow.discovery     ] link detected: 00-00-00-00-00-02.4 -> 00-00-00-00-00-03.1
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] 5 ports changed
[openflow.discovery     ] link detected: 00-00-00-00-00-01.3 -> 00-00-00-00-00-02.1
[openflow.spanning_tree ] Spanning tree updated
[openflow.spanning_tree ] 5 ports changed
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-05 2]
[openflow.discovery     ] link detected: 00-00-00-00-00-01.4 -> 00-00-00-00-00-05.4
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-02 3]
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-03 6]
[openflow.spanning_tree ] Requested switch features for [00-00-00-00-00-01 4]
```

Além dos comandos já citados na aula anterior pode-se ainda derrubar e subir um novo *link* na rede através dos comandos link sX sY up e link sX sY down. Ao fazer isto na janela do controlador poderá ser observado a mudança na *spanning tree*. Outro comando interessante é o dpctl dump-ports-desc que exibe os *switches* da rede bem como seus *links*. Segue um exemplo após derrubar o *link* entre os *switches s4* e *s5*.

```
*** s4 ----------------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s4-eth1): addr:8a:13:95:ee:e4:55
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s4-eth2): addr:e6:32:a5:85:fe:b2
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s4-eth3): addr:3e:d0:3d:a8:19:37
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s4-eth4): addr:66:a3:ae:98:b5:21
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s4): addr:6e:9e:24:37:f7:49
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
*** s5 ----------------------------------------
OFPST_PORT_DESC reply (xid=0x2):
 1(s5-eth1): addr:b6:af:c9:13:45:4d
     config:     PORT_DOWN
     state:      LINK_DOWN
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s5-eth2): addr:ea:d4:ac:d2:4c:28
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s5-eth3): addr:86:f2:a0:81:51:1b
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s5-eth4): addr:ee:3d:67:67:d4:9e
     config:     NO_FLOOD
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s5): addr:46:7e:e9:cb:9a:4d
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

B.6 Exercício

Exercício 1) Utilizando os comandos informados nesse laboratório crie uma rede – sem usar a opção --mac – e preencha a tabela a seguir com os endereços IP e MAC da rede.

SWITCHES	**IP**	**MAC**
Switch 1		
Switch 2		
Switch 3		
Switch 4		
Switch 5		

Tabela 1 – Identificando os nós da rede

ANEXO C – Laboratório 03

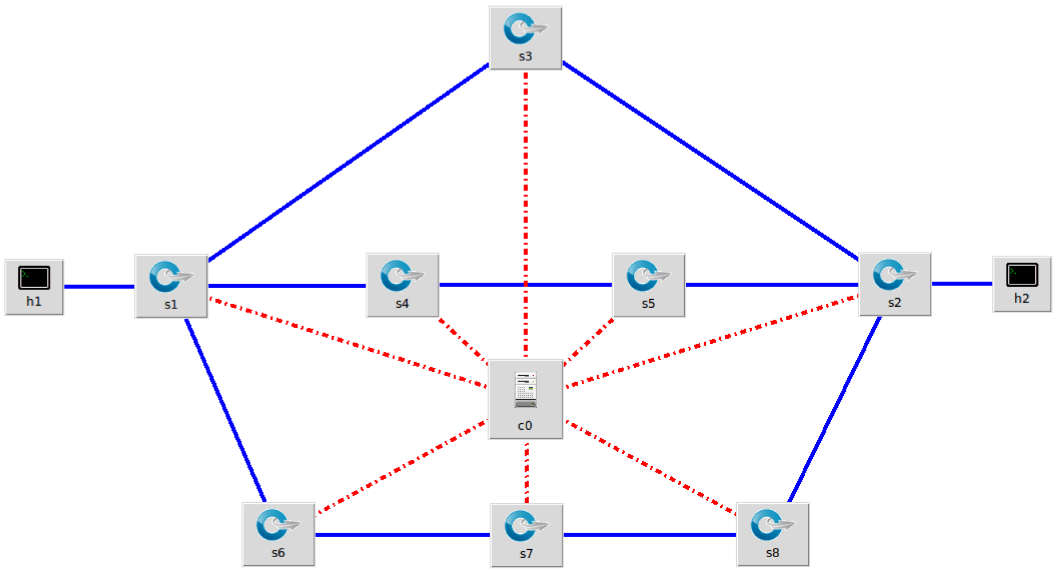
Objetivo: Este laboratório experimenta os conceitos de multi-caminhos do Plano de Controle de Redes Definidas por *Software* (SDN) utilizando Mininet para simular sua estrutura e POX como controlador, todos sendo executados em uma máquina virtual.

C.1 Fundamentos Teóricos

Esta atividade tem como objetivo discutir os conceitos de multi-caminhos relacionados ao Plano de Controle das Redes Definidas por *Software* utilizando Mininet e POX para experimentação.

C.2 Topologia - Cenário Customizado (8 *switches* – 2 *hosts*)

Este cenário é representado por um diamante contendo 2 *hosts* conectados à 8 *switches* ligados ao longo de 3 caminhos distintos, ou seja, existe um *loop* na rede conectando dois nós diferentes. Neste momento será utilizado também um controlador remoto ligado a todos os *switches*.




```
from mininet.topo import Topo

class MultipleTopo( Topo ):
  def __init__( self ):
    # Initialize topology
```

```
    Topo.__init__( self )

  #Topology for a network with multiple paths between 2 hosts.
  def build( self, paths = 3 ):
    # Numbering:  s1..N
    self.switchNum = 3
    # Build topology
    self.addMultiple( paths )

  def addMultiple( self, paths ):
    host1 = self.addHost( 'h1' )
    host2 = self.addHost( 'h2' )
    switch1 = self.addSwitch( 's1' )
    switch2 = self.addSwitch( 's2' )
    self.addLink( switch1, host1 )
    self.addLink( switch2, host2 )

    for i in range( paths ):
      switch = self.addSwitch( 's%s' % self.switchNum )
      self.addLink( switch1, switch )
      self.switchNum += 1
      for _ in range( i ):
        switchAux = self.addSwitch( 's%s' % self.switchNum )
        self.switchNum += 1
        self.addLink( switch, switchAux )
        switch = switchAux

    self.addLink( switch, switch2 )
    return

topos = { 'topoMultiple': ( lambda: MultipleTopo() ) }
```

C.3 Testes

Para execução dos testes dessa topologia serão necessários cinco terminais com conexões SSH para a VM Mininet, uma para criação da rede, uma para inicialização do controlador e os outros três para o Wireshark.

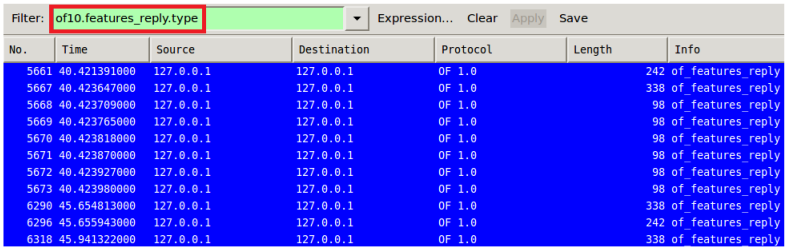
Neste ponto os testes diferem dos anteriores pois existem múltiplos caminhos e

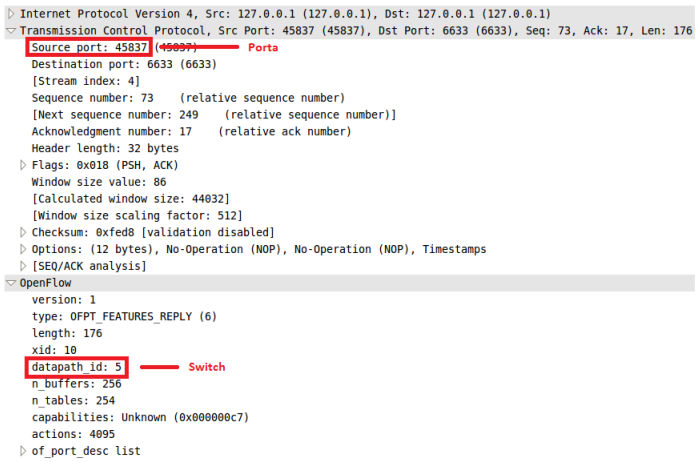
precisamos descobrir qual foi o escolhido durante a execução. Para isso devemos iniciar as três sessões do Wireshark e em cada uma delas e aplicar o filtro of10.features_reply.type. Esse filtro retornará as informações necessárias para os testes no momento da criação da rede.

O próximo passo é iniciar o controlador através do seguinte comando: sudo ~/pox/pox.py z forwarding.l2_learning openflow.spanning_tree log.level --DEBUG samples.pretty_log openflow.discovery info.packet_dump.

Com o controlador executando devemos iniciar a topologia através do comando: sudo mn --custom ~/mininet/custom/topo-multiple.py --topo topoMultiple --switch ovsk --controller=remote.

Após a inicialização da rede deve-se analisar o retorno gerado no Wireshark (pode-se escolher qualquer um das sessões abertas) pois serão exibidos todos os *switches* com suas respectivas portas. Faça um mapeamento com os resultados conforme as imagens a seguir.



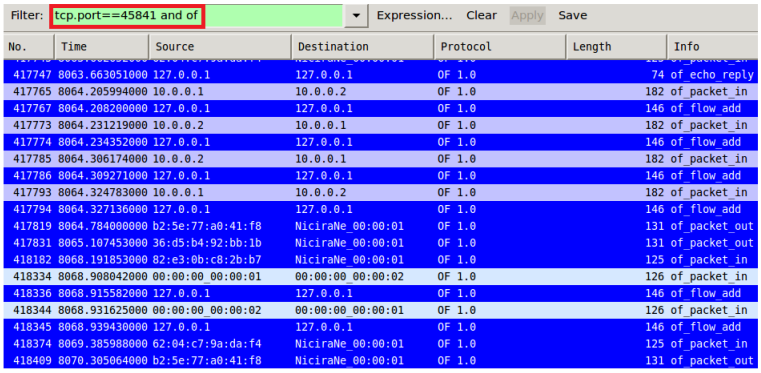


SWITCH - PORTA
S1 - 45840
S2 - 45838
S3 - 45841
S4 - 45842
S5 - 45837
S6 - 45844
S7 - 45843
S7 - 45839

Nesse teste a verificação da rota percorrida serão analisados os *switches s3*, *s5* e *s7*. Para essa verificação será necessário aplicar o filtro tcp.port==XXXXX and of onde

XXXXX é a porta de cada *switch*. Com tudo configurado o próximo passo é executar um pingall e analisar por qual *switch* o pacote passou.

No exemplo a seguir o caminho escolhido foi através do *switch s3*.



C.4 Exercícios

Exercício 1) Utilizando os comandos informados nesse laboratório crie uma rede – sem usar a opção --mac – e preencha a tabela a seguir com os endereços IP e MAC da rede.

SWITCHES	IP	MAC
Switch 1		
Switch 2		
Switch 3		
Switch 4		
Switch 5		
Switch 6		
Switch 7		
Switch 8		

Tabela 2 – Identificando os nós da rede

Exercício 2) Utilizando o Wireshark, descubra por qual switch os dados são enviados e marque com um X na Tabela 2. Feito isso, derrube o link relacionado a esse switch e refaça o teste. Por qual nó os dados foram transferidos?

SWITCHES	**ANTES**	**DEPOIS**
Switch 3		
Switch 5		
Switch 7		

Tabela 3 – Identificando as rotas de transferência