CADERNO DE PRÁTICAS

OFICINA MINICURSO SBRC2016

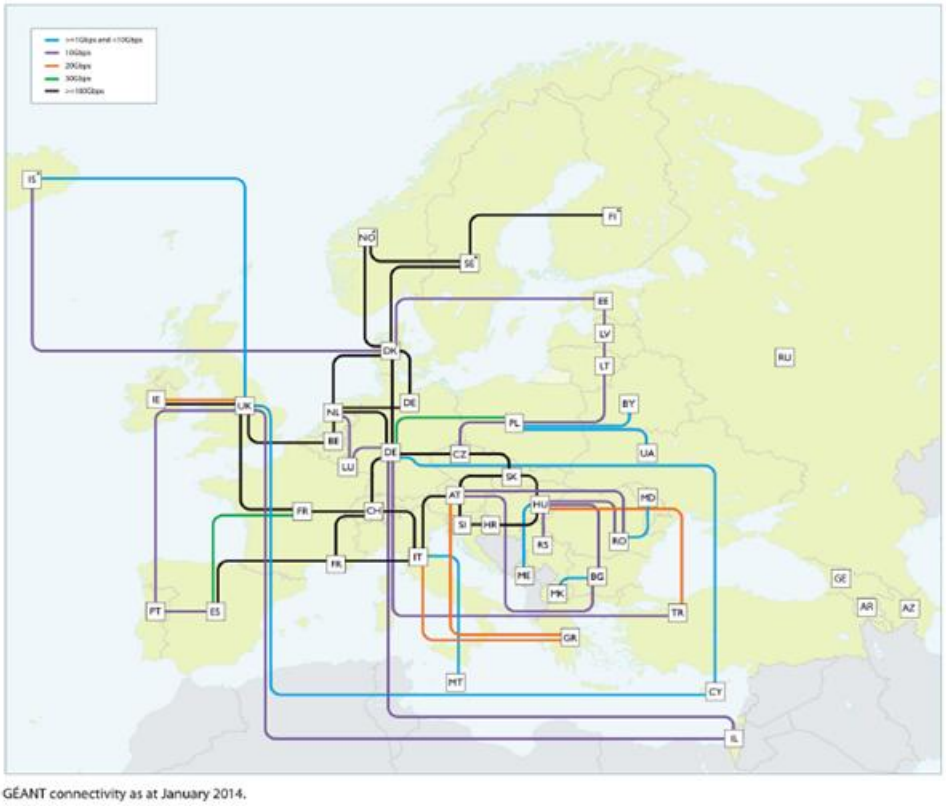
"ROTEAMENTO POR SEGMENTOS"

Departamento de Eletrônica – Escola Politécnica
Programa de Engenharia Elétrica – COPPE
Universidade Federal do Rio de Janeiro

Antonio José Silvério, Miguel Elias M. Campista,
Luís Henrique M. K. Costa

Seção 5: Experimentação Prática

Na seção de experimentação prática utilizaremos a ferramenta de simulação OSHI-Open Source Hybrid IP/SDN networking desenvolvida para o Projeto Dreamer (*Distributed Resilient Architecture meeting Carrier Grade Requirements*) da GÉANT que representa a Rede de Pesquisa e Educação Pan-Européia, interconectando os centros de pesquisa e de educação da Europa conforme mostrado na Figura 5.1. A ferramenta foi desenvolvida com o objetivo de investigar e prototipar redes definidas por software baseadas em protocolo Openflow e provendo as mesmas funcionalidades das redes IP/MPLS. A ideia foi o desenvolvimento de um simulador que permite trabalhar com nós de rede virtuais e reais, com implementação de software aberto em IP e SDN. A interface Southbound com o controlador SDN é Openflow, e a interface Northbound com o servidor de gerenciamento são APIs REST. O tráfego é enviado em vários tipos de túneis tais como MPLS, VLAN, Q in Q, Ethernet PBB (Provider Backbone Bridge). através do controlador SDN. O sistema operacional de rede escolhido foram o Ryu e o ONOS desenvolvido pelo ON.LAB (*Open Networking Lab*). O simulador de rede híbrida IP/SDN possui três formas de implementação: no Virtual Box "run time", no ambiente de simulação do Mininet, e em provas de conceito de redes SDN como o projeto OFELIA (*OpenFlow in Europe: Linking Infrastructure and Applications*) aplicando o conceito de experimento como serviço (testbed as a service). Utilizaremos nesta seção a segunda opção com o emulador Mininet. O simulador utiliza nós virtuais de rede de núcleo denominados Roteadores de Núcleo em software aberto (do inglês, Open Source Label Switch Routers - OpenLSR), que geram pacotes OSPF e LDP utilizando o Quagga computando os rótulos MPLS que são instalados nos switches usando o protocolo OpenFlow. O controlador SDN também possui um módulo de software para engenharia de Tráfego e roteamento por segmentos



GÉANT connectivity as at January 2014.

Figura 5.1 – Rede GÉANT

EXPERIMENTO 1: CONSTRUÇÃO DE UMA REDE SIMPLES COM ROTEAMENTO NO MININET

5.1. Introdução ao Mininet

O Mininet é um emulador de rede que cria redes com servidores, switches, controladores e enlaces virtuais. O Mininet cria uma rede virtual realística rodando em um núcleo real (Kernel Linux), switches e códigos de aplicação em uma única máquina física ou virtual (*Virtual Machine* - VM) que pode ser nativa ou em nuvem (*Cloud)*. O Mininet possui linha de comando própria (Command Line Interface - CLI) e APIs, que permitem a criação de rede e serviços, customização e compartilhamento com outros usuários, e também a implantação em hardware real. O Mininet é uma ferramenta útil de desenvolvimento, aprendizado e pesquisa rodando em um laptop ou PC. O Mininet é ideal para experimentos com OpenFlow e Redes Definidas por Software (*Software Defined Networks* – SDN). As principais limitações das redes emuladas no Mininet referem-se à capacidade de banda disponível e CPU que não podem exceder à capacidade e banda disponível no servidor onde o Mininet está instalado, e ao fato do Mininet não rodar aplicações que não sejam compatíveis com Linux. A Figura 5.2 mostra um exemplo de rede simples (topologia mínima) disponível como exemplo no Mininet.

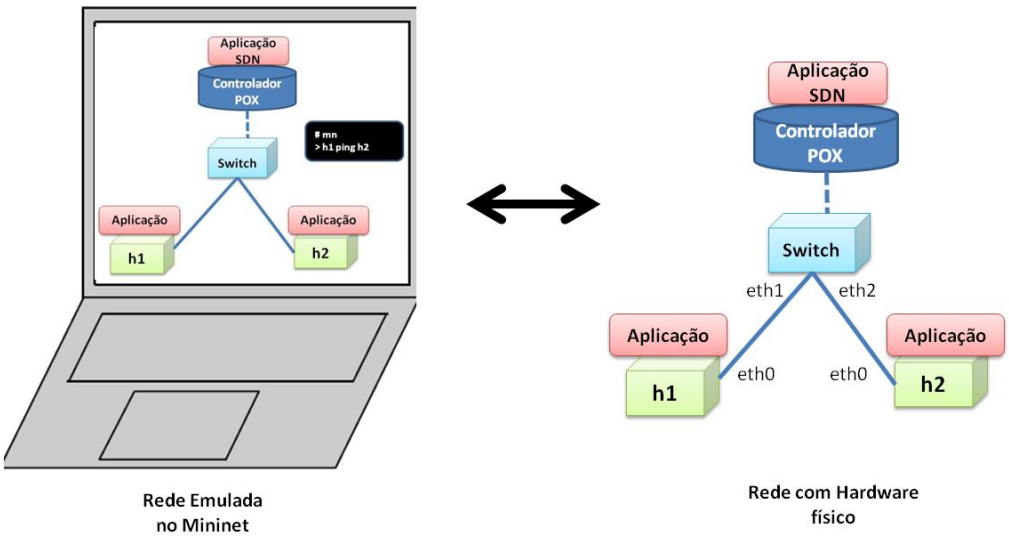


Figura 5.2 – Abstração da rede física no Mininet – Topologia Exemplo

Os passos seguintes ilustram a linha de comando para uma topologia mínima (minimal topology) do Mininet, que inclui o controlador POX , um switch OpenFlow e dois servidores conforme mostrado na Figura 5.2.

Passo 1: Iniciando o Mininet com a topologia mínima:

```
mininet@mininet-vm: ~$  sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
```

```
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Passo 2: Verificando os nós da rede:

```
mininet> nodes
available nodes are:
c0 h1 h2 s1
```

Passo 3: Verificando os enlaces da rede:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Passo 4: Verificando os endereços lógicos dos dispositivos da rede. Os nós são configurados em uma subrede padrão (default) de IP 10.0.0.0/8:

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=8683>
<Host h2: h2-eth0:10.0.0.2 pid=8684>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=8687>
<OVSController c0: 127.0.0.1:6633 pid=8675>
```

Passo 5: Acessando os servidores h1 e h2 através do Xterm :

```
mininet> xterm h1 h2 s1
```

Passo 6: Testando a conectividade da rede através do comando PING :

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

O Mininet permite a criação de topologias padrões mais complexas com mais switches e servidores, em árvore ou linear. O Mininet permite a construção de topologias padronizadas através de APIs em Python, que é à base de construção do Mininet, as APIs em Python são usadas para orquestração no entanto a emulação é em C compilado. No exemplo abaixo o arquivo`Router.py'' mostra a criação de uma rede ponto a ponto com 2 switches com 1 servidor cada e um roteador interligando os switches:

```
mininet@mininet-vm: ~$  cat Router.py

from mininet.topo import Topo

class Router_Topo(Topo):
   def __init__(self):
      "Create P2P topology."

      # Initialize topology
      Topo.__init__(self)

      # Add hosts and switches
      H1 = self.addHost('h1')
      H2 = self.addHost('h2')
      H3 = self.addHost('h3')
      S1 = self.addSwitch('s1')
      S2 = self.addSwitch('s2')

      # Add links
      self.addLink(H1, S1)
      self.addLink(H2, S1)
      self.addLink(H2, S2)
      self.addLink(H3, S2)

topos = {
      'router': (lambda: Router_Topo())
}

mininet@mininet-vm: ~$ sudo mn --custom /home/mininet/Router.py --topo router
```

A seguir é listada a linha de comando para criação do serviço (cliente-servidor HTTP) apresentada na Figura 5.2, onde as aplicações são um cliente e servidor

```
mininet@mininet-vm: ~$  sudo  mn      { Iniciando o Mininet }
mininet@mininet-vm: ~$  mn --topo tree,depth=3,fanout=3 -- link=tc,bw=10  {
carregando topologia padrão parametrizada. Ex. bw banda do link igual a 10 Mbit/s }
mininet> xterm h1 h2  { acessando os servidores h1 e h2 }
h1# wireshark & { habilitando o wireshark para captura de pacotes na interface local de
h1 }
h2# python -m SimpleHTTPServer 80 & { habilitando a aplicação de HTTP Server em
h1 }
h1# firefox & { habilitando a aplicação do cliente em h2 }
```

Para ver a qualquer momento a lista das opções da linha de comando, iniciar com a topologia mínima (minimal) e digitar:

```
mininet>help
```

A Tabela 5.1 ilustra algumas das principais APIs do Mininet

API do Mininet	Propósito
net = Mininet()	net é um objeto de Mininet( ).
h1 = net.addHost('h1')	h1 é um objeto de Servidor( ).
s1 = net.addSwitch('s1')	s1 é um objeto de Switch( ).
c0 = net.addController('c0')	c0 é um controlador.
net.addLink(h1, s1)	cria um objeto Link( ).
net.start( )	Aloca os recursos computacionais e inicia a rede net.
h2.cmd (`python –m SimpleHTTPServer 80 &'	Na linha de comando de h2 roda a aplicação python de HTTP Server.
net.stop( )	Desaloca os recursos computacionais para a rede net.
Net = Mininet(link, TCLink, host=CPULimitedHost)	Habilita na rede net modelo de desempenho dos enlaces e classes de ``hosts''.
net.addLink(h2, s1, bw=10, delay='50ms')	Limita a banda e adiciona um atraso em um determinado enlace.
net.addHost('h1', cpu=.2)	Limita o processamento da CPU.
CLI (net)	Abre os terminais com a CLI em cada nó da rede net.
h2.cmd (`kill %python')	Envia um comando, aguarda uma saída e retorna.

Tabela 5.1 – Principais APIs do Mininet

5.2. Construção de uma rede simples com roteamento no Mininet

Este experimento com o Mininet visa criar uma rede simples com roteamento estático, onde o roteador receberá quadros ethernet e irá processar os pacotes como um roteador físico real, encaminhando os pacotes pelas interfaces corretas. O roteador emulado encaminhará pacotes de um host cliente para dois servidores de aplicação HTTP conforme mostrado na Figura 5.3. Ao final deste experimento, os seguintes testes poderão ser realizados:

- Teste de conectividade (*ping*) do cliente para as interfaces do roteador (192.168.2.1, 172.64.3.1 e 10.0.1.1).
- Teste de rastreio da rota (*traceroute*) do cliente para qualquer interface do roteador.
- Teste de conectividade (*ping*) do cliente para as interfaces dos servidores de aplicação.
- Baixar um arquivo utilizando HTTP de um dos servidores.

O experimento utiliza "scripts" prontos com as APIs do Mininet:

Passo 1: Checando o código que foi baixado:

mininet@mininet-vm: git clone https://huangty@bitbucket.org/huangty/cs144_lab3.git
mininet@mininet-vm: cd cs144_lab3/

mininet@mininet-vm: git checkout --track remotes/origin/standalone

Passo 2: Instalando o módulo do controlador POX:

mininet@mininet-vm: cd ~/cs144_lab3
./config.sh

Passo 3: Verificando os arquivos de configuração IPCONFIG, que lista os endereços dos *hosts* emulados e a RTABLE que lista as rotas estáticas da tabela de rotas:

mininet@mininet-vm: cat ~/cs144_lab3/IP_CONFIG
server1 192.168.2.2
server2 172.64.3.10
client 10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1

mininet@mininet-vm: cat ~/cs144_lab3/rtable
10.0.1.100 10.0.1.100 255.255.255.255 eth3
192.168.2.2 192.168.2.2 255.255.255.255 eth1
172.64.3.10 172.64.3.10 255.255.255.255 eth2

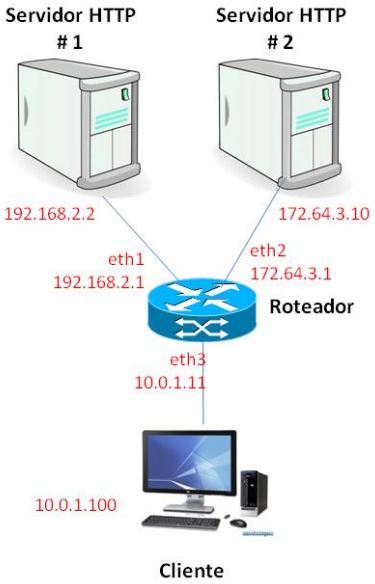


Figura 5.3 – Construção de uma rede simples com roteamento no Mininet

Passo 4: Configurando o ambiente executando o arquivo *config.sh*:

mininet@mininet-vm: cd ~/cs144_lab3/
mininet@mininet-vm: ./config.sh

Iniciando o Mininet

mininet@mininet-vm: cd ~/cs144_lab3/
mininet@mininet-vm: ./run_mininet.sh

*** Shutting down stale SimpleHTTPServers

```
*** Shutting down stale webservers
server1 192.168.2.2
server2 172.64.3.10
client 10.0.1.100
sw0-eth1 192.168.2.1
sw0-eth2 172.64.3.1
sw0-eth3 10.0.1.1
*** Successfully loaded ip settings for hosts
 {'server1': '192.168.2.2', 'sw0-eth3': '10.0.1.1', 'sw0-eth1': '192.168.2.1', 'sw0-eth2':
'172.64.3.1', 'client': '10.0.1.100', 'server2': '172.64.3.10'}
*** Creating network
*** Creating network
*** Adding controller
*** Adding hosts:
client server1 server2
*** Adding switches:
sw0
*** Adding links:
(client, sw0) (server1, sw0) (server2, sw0)
*** Configuring hosts
client server1 server2
*** Starting controller
*** Starting 1 switches
sw0
*** setting default gateway of host server1
server1 192.168.2.1
*** setting default gateway of host server2
server2 172.64.3.1
*** setting default gateway of host client
client 10.0.1.1
*** Starting SimpleHTTPServer on host server1
*** Starting SimpleHTTPServer on host server2
*** Starting CLI:
mininet>
```

Passo 5: Deve-se abrir outro terminal, para iniciar o controlador POX. É necessário aguardar o Mininet se conectar ao controlador antes de prosseguir para os próximos passos.

```
mininet@mininet-vm:  cd ~/cs144_lab3/
mininet@mininet-vm:  ln -s ../pox
mininet@mininet-vm:  ./run_pox.sh
```

Quando o Mininet conectar ao controlador, a seguinte aparecerá a seguinte mensagem:

```
INFO:openflow.of_01:[Con 1/249473472573510] Connected to e2-e5-11-b6-b0-46
DEBUG:.home.ubuntu.cs144_lab3.pox_module.cs144.ofhandler:Connection [Con
1/249473472573510]
DEBUG:.home.ubuntu.cs144_lab3.pox_module.cs144.srhandler:SRServerListener
catch RouterInfo even, info={'eth3': ('10.0.1.1', '86:05:70:7e:eb:56', '10Gbps', 3), 'eth2':
('172.64.3.1', 'b2:9e:54:d8:9d:cd', '10Gbps', 2), 'eth1': ('192.168.2.1', '36:61:7c:4f:b6:7b',
'10Gbps', 1)}, rtable=[]
```

Passo 6: Para realizar os testes propostos é necessário executar o arquivo binário da solução completa (*sr_solution*):

```
mininet@mininet-vm:   cd ~/cs144_lab3/
mininet@mininet-vm:   ./sr_solution
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100          10.0.1.100      255.255.255.255 eth3
192.168.2.2          192.168.2.2     255.255.255.255 eth1
172.64.3.10          172.64.3.10     255.255.255.255 eth2
---------------------------------------------
Client ubuntu connecting to Server localhost:8888
Requesting topology 0
successfully authenticated as ubuntu
Loading routing table from server, clear local routing table.
Loading routing table
---------------------------------------------
Destination     Gateway         Mask    Iface
10.0.1.100          10.0.1.100      255.255.255.255 eth3
192.168.2.2          192.168.2.2     255.255.255.255 eth1
172.64.3.10          172.64.3.10     255.255.255.255 eth2
---------------------------------------------
Router interfaces:
eth3    HWaddr86:05:70:7e:eb:56
        inet addr 10.0.1.1
eth2    HWaddrb2:9e:54:d8:9d:cd
        inet addr 172.64.3.1
eth1    HWaddr36:61:7c:4f:b6:7b
        inet addr 192.168.2.1
 <-- Ready to process packets -->
```

Passo 7: Iniciar os testes propostos, no Mininet:

```
mininet> client ping -c 3 192.168.2.2

PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_req=1 ttl=63 time=66.9 ms
64 bytes from 192.168.2.2: icmp_req=2 ttl=63 time=49.9 ms
64 bytes from 192.168.2.2: icmp_req=3 ttl=63 time=68.8 ms

mininet> client traceroute -n 192.168.2.2

traceroute to 192.168.2.2 (192.168.2.2), 30 hops max, 60 byte packets
 1  10.0.1.1  146.069 ms  143.739 ms  143.523 ms
 2  192.168.2.2  226.260 ms  226.070 ms  225.868 ms

mininet> client wget http://192.168.2.2

--2012-12-17 06:52:23--  http://192.168.2.2/
Connecting to 192.168.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 161 [text/html]
```

```
Saving to: `index.html'

    0K                                        100% 17.2M=0s

2012-12-17 06:52:24 (17.2 MB/s) - `index.html' saved [161/161]
```

Passo 8: Captura dos pacotes no formato .pcap através do executável *starter source code* (*sr*). É possível capturar os pacotes no formato .pcap e posterior leitura no *Wireshark.* Em outro terminal executar o arquivo sr:

```
mininet@mininet-vm: cd ~/cs144_lab3/router/
mininet@mininet-vm: make
mininet@mininet-vm:./sr
mininet@mininet-vm: ./sr -l logname.pcap
```

Final do Experimento.

EXPERIMENTO 2: CRIAÇÃO DE UM SERVIÇO EM REDE MPLS-TE ATRAVÉS DA FERRAMENTA OSHI

5.3. Open Source Hybrid IP/SDN Networking (OSHI)

A ferramenta denominada ``OSHI'' (*Open Source Hybrid IP/SDN networking*) foi projetada e implementada no sistema operacional Linux. Um nó Oshi combina o encaminhamento tradicional IP (com OSPF) e encaminhamento em SDN implementada por meio de Open vSwitches em um nó IP/SDN híbrido. Não há soluções de código aberto para nós híbridos, então foi projetado um código aberto combinando switches baseadas em OpenFlow (*OpenFlow Capable Switch* - OFCS), um encaminhador de pacotes IP e um ``daemon'' de roteamento IP. O OFCS é conectado ao conjunto de interfaces de rede físicas pertencentes à rede IP/SDN integrada, enquanto o encaminhador de pacotes IP é ligado a um conjunto de portas virtuais OFCS, como mostrado na Figura 5.4. No nó OSHI, o componente é OFCS implementado usando o Open vSwitch (OVS), o encaminhador de pacotes IP é o kernel do Linux e o Quagga atua como o ``daemon'' de roteamento.

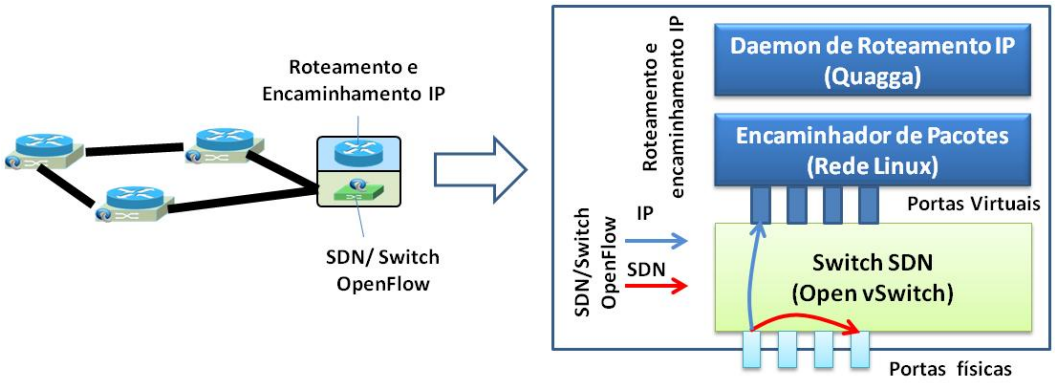


Figura 5.4 – Nó híbrido IP/SDN OSHI

Na arquitetura IP/MPLS há uma noção clara dos túneis MPLS, chamados LSP (*Label Switch Path*). Dentro uma rede SDN vários tipos de túneis ou mais genericamente caminhos de rede podem ser criados, proporcionado pela programabilidade da tecnologia SDN classificando o tráfego com base em vários campos tais como endereços MAC ou IP, ``tags'' de VLAN e rótulos MPLS. Como não há um padrão estabelecido ou terminologia para tal conceito, esses caminhos são referidos como caminhos baseados em SDN (*SDN Based Paths* - SBP). Um SBP é uma configuração ``circuito virtual´´ usando a tecnologia SDN para transmitir um fluxo de pacotes específico entre dois pontos finais através de um conjunto de nós SDN. A noção de fluxo de pacotes é muito ampla e pode variar de um microfluxo, ou seja, uma ligação TCP específica entre duas máquinas, ou ainda de macrofluxo como por exemplo quando todo o tráfego é orientado para uma determinada sub-rede IP. Um fluxo pode ser classificado a partir de campos dos cabeçalhos em diferentes protocolos. É considerado em uma rede híbrida IP/SDN: mecanismos para coexistência do tráfego IP legado e dos SBPs, conjunto de serviços que podem oferecer os SBPs, classificação do tráfego ingresso e mecanismos de tunelamento. As portas virtuais internas que interconectam os OFCSs com o encaminhador de pacotes IP são realizados usando o recurso de ``Porta Virtual'', oferecido pela Open vSwitch. Cada porta virtual é conectada a uma porta física da rede IP/SDN, de modo que o

mecanismo de roteamento IP é processado em termos das portas virtuais, ignorando as físicas. O OFCS diferencia entre os pacotes IP legados e pacotes pertencentes a SBPs. Por padrão, ele encaminha os pacotes IP legados das portas físicas para as portas virtuais, de modo que eles podem ser processados pelo encaminhador de pacotes IP, controlado pelo ``daemon'' de roteamento IP. Esta abordagem evita a necessidade de traduzir tabela de roteamento IP em regras SDN para popular a tabela do OFSCS.

Os serviços a serem simluados pela ferramenta OSHI, considerados no projeto Dreamer são do tipo IP ponto a ponto como Circuitos Virtuais Alugados (*Virtual Leased Lines* - VLL), Pseudo fio (*Pseudowires* - PW) de camada 2 e switches virtuais de camada 2 (*Virtual Switched Services*-VSS) sobre um Backbone IP/MPLS, permitindo também simular engenharia de tráfego em redes IP/MPLS (túneis TE) e Roteamento por Segmentos.

A fim de apoiar tanto o desenvolvimento, aspectos de teste e de avaliação comparativa, adicionalmente foram desenvolvidas de ferramentas de código aberto (*Open Source*), denominada Mantoo (*Management Tools*) que se destinam a apoiar experiências SDN sobre simuladores e ``testbeds'' distribuídos sendo capaz de conduzir e ajudar os experimentadores nas três fases que compõem uma experiência: projeto, implantação e controle As ferramentas Mantoo incluem conjunto de scripts python para configurar e controlar os simuladores ou ``testbeds'' distribuídos e um conjunto de scripts para medições de desempenho e um visualizador de topologia 3D (*Topology 3D*) em uma interface gráfica Web mostrada na Figura 5.5.

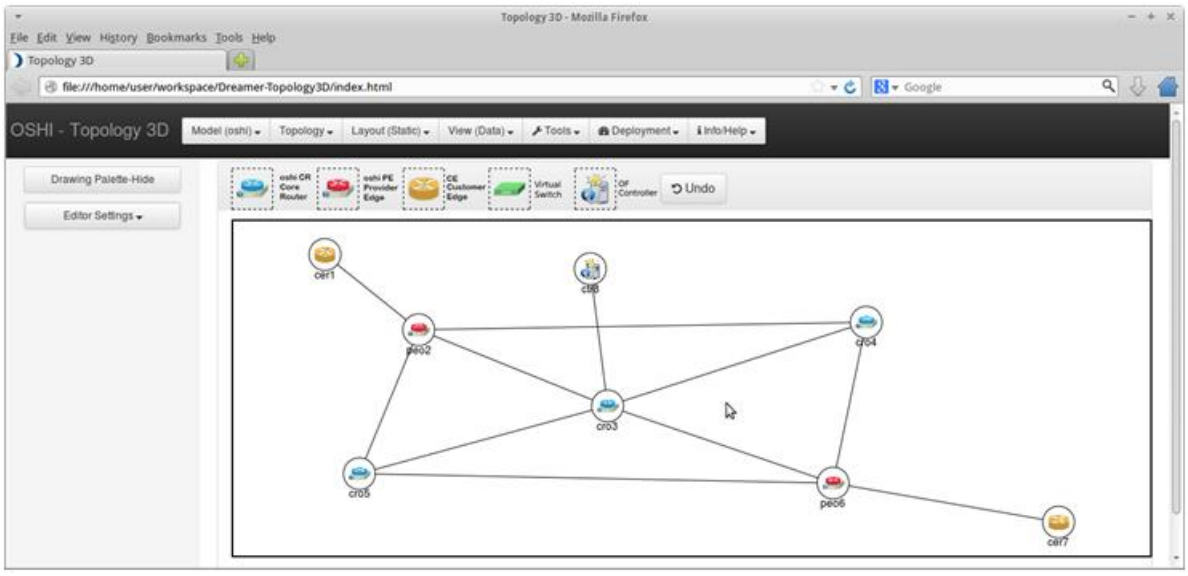


Figura 5.5 – Ferramenta Topology 3D do conjunto de ferramentas Mantoo

A representação da topologia é através de arquivos no formato JSON, e a interface gráfica possui um módulo de criação automático da topologia a partir dos dados desses arquivos. Os ``scripts" de configuração incluem um analisador' da topologia (*parser*), incluem as extensões das bibliotecas do Mininet para emulação dos dispositivos de rede e o configurador OSHI, cujo controlador utiliza APIs Rest, implementando os serviços como Roteamento por Segmentos em aplicações na linguagem python.

5.4. Criação de um serviço em rede MPLS TE através do OSHI-TE

Os serviços oferecidos pela ferramenta, considerados no projeto Dreamer são do tipo IP ponto a ponto como Circuitos Virtuais Alugados (*Virtual Leased Lines-VLL*), Pseudo

fio (*Pseudowire Emulação Edge* - PWE3) de camada 2 e switches virtuais de camada 2 (*Virtual Switched Services* - VSS) sobre um Backbone IP/MPLS. Estes serviços são implementados como SBPs, e são estabelecidos pelos controladores SDN Ryu ou Floodlight centralizados. Duas propostas foram concebidas e implementadas para os SBPs, a primeira proposta baseia-se em VLAN Tag, e tem algumas limitações em termos de escalabilidade (no máximo 2^{12} = 4096 VLANs id) e problemas de interoperabilidade com equipamentos legados na administração de VLANs já existentes, mas tem a vantagem de poder ser implementada usando-se a versão mais simples do OpenFlow, a versão 1.0. A segunda proposta baseia-se em rótulos MPLS, que não sofrem problemas de escalabilidade e podem interoperar melhor com equipamentos legados, mas requerem a versão 1.3 do OpenFlow, que possui mais recursos. Dos serviços básicos implementados pela ferramenta, o serviço IP VLL garante que os pontos finais IP sejam diretamente interligados como se eles estivessem na mesma Ethernet LAN e enviando pacotes IP e ARP aos dispositivos conectados. O serviço IP VLL foi implementado usando VLAN ``tags'' e rótulos MPLS. Em ambos implementações os endereços MAC de origem e destino, são preservadas nos roteadores intermediários do núcleo da rede. Outro serviço básico oferecido pela ferramenta é o serviço de pseudo fio (*Pseudowire Emulação Edge* - PWE3), descrito na RFC 3985. O serviço PW oferece um serviço de ``cabo'' virtual totalmente transparente, onde os dispositivos podem enviar pacotes com Ethertype arbitrário (por exemplo, incluindo VLAN ou Q-em-Q). O serviço PW foi implementado usando apenas as rótulos MPLS, encapsulando o pacote Ethernet cliente, como mostrado na Figura 5.6.

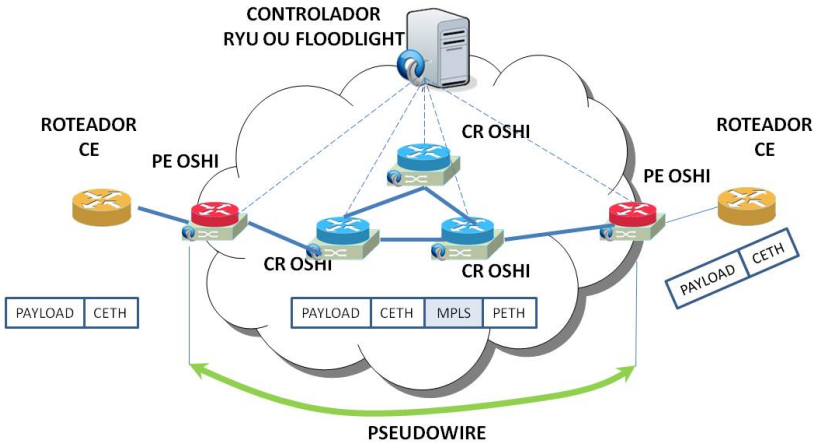


Figura 5.6 – Implementação do serviço IP VLL com a ferramenta OSHI

A Figura 5.6 representa a rede de roteadores de núcleo MPLS, com roteadores de borda (*Label Edge Router*- LER ou *Provider Edge* - PE) e roteadores intermediários (*Label Switch Router* - LSR ou *Core Routers* - CR), ambos implementados no conceito híbrido IP/SDN de código aberto (*Open Source Hybrid SDN IP* - OSHI). Os roteadores cliente (*Customer Equipment* - CE) são roteadores apenas com plano de encaminhamento IP. O serviço de pseudo fio em um rede híbrida IP/SDN tem as mesmas características dos serviços implementados em rede tradicional IP/MPLS.
O controlador utilizado foi o Ryu e um ``script'', denominado VLLPusher, escrito em python e adaptado para executar a instalação dos SBPs. O ``script'' utiliza a API Rest do controlador Ryu, a fim de recuperar a topologia de rede, em seguida, avaliar o

caminho mais curto, que interliga os pontos finais do serviço. Neste passo, é possível introduzir os aspectos de engenharia de tráfego (*Traffic Engineering* - TE) para a seleção do trajeto. Finalmente, o ``script'' aloca os rótulos MPLS e usa a API REST do controlador Openflow para definir as regras para o encaminhamento de pacotes e a comutação de rótulos MPLS. A Figura 5.7 mostra as operações de tunelamento desempenhadas pelo nó OSHI PE para o serviço de pseudo fio. O mecanismo de tunelamento é uma solução similar ao ethernet sobre MPLS (*Ethernet over MPLS* - EoMPLS), onde o pacote do roteador cliente é encapsulado incluindo o cabeçalho ethernet original em um pacote MPLS, sendo este transportado em um pacote ethernet com um novo cabeçalho (PETH). Como o protocolo OpenFlow não suporta nativamente o serviço EoMPLS, incluindo a OFCS usada na ferramenta. Para efeito de simulação do serviço EoMPLS, foi utilizando um túnel GRE (*Generic Routing Encapsulation*). O cabeçalho extra do túnel GRE e do pacote adiciona de 20 bytes ao encapsulamento do serviço EoMPLS. Para suportar o encapsulamento GRE, a na arquitetura do nó OSHI da Figura 5.4, é adicionado um encapsulador de acesso (*ACcess Encapsulator* - ACE) permitindo configurar o túnel GRE nos pontos finais do pseudo fio.

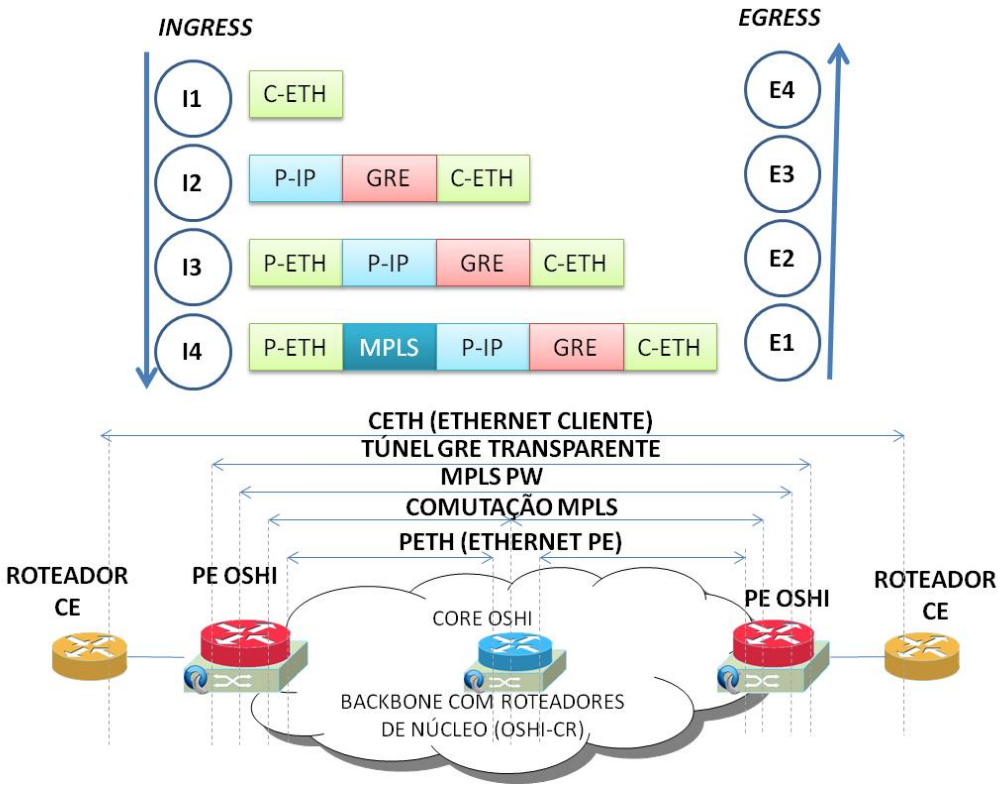


Figura 5.7 – Implementação do PW em um PE OSHI

Esta arquitetura do nó OSHI mais complexa permite não somente a criação dos serviços de pseudo fio (*PW*), mas também de switches virtuais (*VSS*). A switch Open Flow (*OFCS*) suporta a inserção e retirada de rótulos, enquanto o encapsulador de acesso (*ACE*) provê o túnel GRE. O ACE é implementado com uma nova instância de switch virtual aberta (*Open vSwitch* - OVS), utilizando duas funções de virtualização recentemente inseridas no ``Kernel'' Linux: espaço de nomes de redes (network namespaces) e pares de portas ethernet virtuais (*virtual ethernet ports*). Em cada pseudo fio, a ACE possui duas portas, uma local ligada ao roteador cliente (*CE*), e outra remota, ligada ao nó provedor de serviço (Provider Edge - PE). Nó nó OSHI-PE a ligação com o PE é feita internamente ao OFCS, da arquitetura padrão do nó OSHI.

A porta de saída do OVS da ACE é um túnel GRE. A Figura 5.8 mostra o detalhe da arquitetura do nó OSHI PE, com os respectivos encapsulamento de ingresso e egresso do pseudo fio, detalhados anteriormente na Figura 5.7.

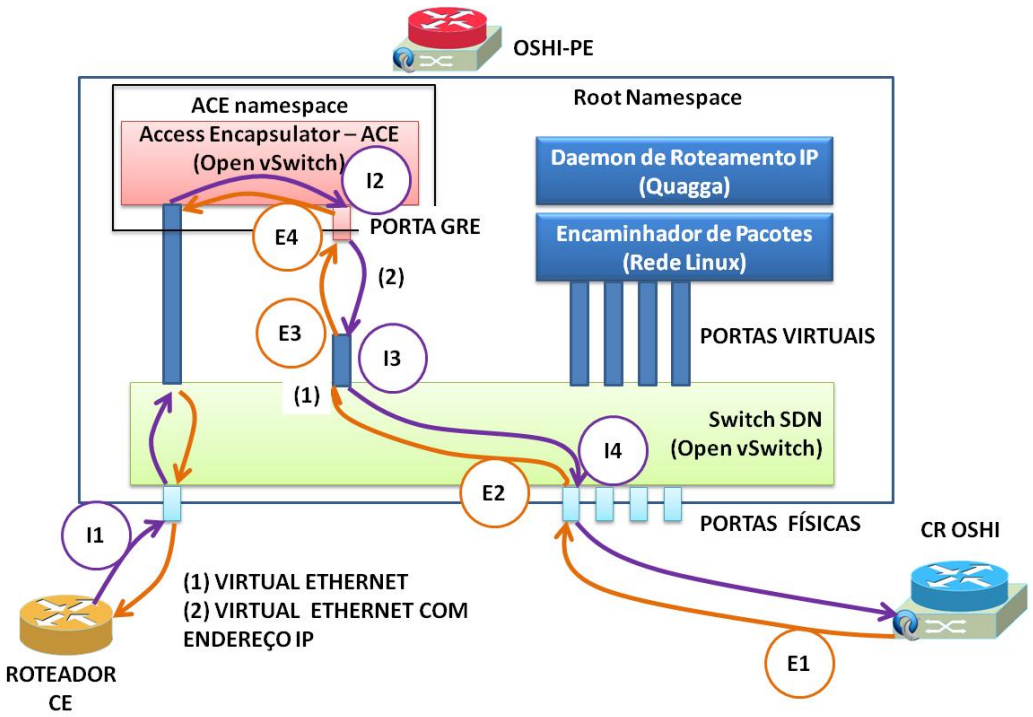


Figura 5.8 – Arquitetura do nó OSHI PE

A seguir são enumerados os passos para a criação de um serviço de pseudo fio (*Pseudowire*), que pode ser executado através da linha de comando ou da interface gráfica do conjunto de ferramentas Mantoo (Topology 3D):

Passo 1: iniciar a emulação usando a linha de comando, executando o ``script'' de carga da topologia (em formato JSON) de um serviço de pseudo fio (PW) usando o Mininet, ou através da interface gráfica (*Topology 3D*), iniciando a topologia no Mininet com o comando ``Deploy'':

user@OSHI-VM:~/workspace/Dreamer-Mininet-Extensions$ sudo
./mininet_deployer.py --topology topo/topo_vll_pw.json

Passo 2: Aguardar o final da execução, até o aparecimento da linha de comando do Mininet (``Mininet>''). A partir desse momento a topologia está carregada.

Passo 3: Acessando os nós da rede: através do comando ``xterm node'', onde ``node'' são os nós da topologia, é possível acessar cada um dos nós, sendo possível realizar testes de conectividade (*ping* e *traceroute*) e captura de pacotes com o Wireshark. Na interface gráfica clicando sobre o nó da topologia com o botão esquerdo do mouse e a tecla <ctrl>, para cada nó é aberta a tela do terminal referente a este nó em uma aba na interface gráfica.

Passo 4: Acessando o controlador abrindo dois terminais SSH (Secure Shell) no servidor onde se encontra o controlador SDN, ou utilizando a linha de comando do Mininet, para checar o se o controlador está endereçando a saída do ``script'' que foi iniciada pelo Mininet :

Mininet> xterm ctr8 ctr8 ou
user@OSHI-VM:~$ ssh -X root@10.255.248.1 (a senha é ´´root")

No primeiro terminal aberto iniciar o controlador Ryu:

user@OSHI-VM:~$ cd /home/user/workspace/dreamer-ryu/ryu/app
root@OSHI-VM:~/workspace/dreamer-ryu/ryu/app# ryu-manager rest_topology.py ofctl_rest.py --observe-links

Passo 5: Testar a conectividade entre os roteadores cliente (CER1 e CER7).

Mininet> xterm cer1
Mininet> ping 10.0.11.2

O teste de conectividade não irá funcionar pois os circuitos MPLS do serviço de pseudo fio não foram criados.

Passo 6: No segundo terminal SSH do controlador, copiar a configuração do serviço vll_pusher.cfg, que foi gerado pelo ´´script" do Mininet na pasta Dreamer-VLL-Pusher/ryu. Este arquivo é necessário para a configuração dos circuitos virtuais.

root@OSHI-VM:~/workspace/Dreamer-Mininet-Extensions# cp vll_pusher.cfg ../Dreamer-VLL-Pusher/ryu

Passo 7: Criar os circuitos virtuais executando os seguintes comandos no terminal SSH do controlador:

root@OSHI-VM:~/workspace/Dreamer-VLL-Pusher/ryu# rm *.json
root@OSHI-VM:~/workspace/Dreamer-VLL-Pusher/ryu# ./vll_pusher.py --controller localhost:8080 --add

O primeiro comando acima apaga os circuitos virtuais anteriores, e o segundo comando adiciona novos circuitos para emulação do serviço.

Passo 8: Testar a conectividade novamente proposta no passo 5, verificando que o serviço está operacional. Iniciar o Wireshark para captura de pacotes no nó OSHI PE (pe02 e pe06):

root@OSHI-VM:~$ wireshark &

- Verificar a continuidade fim a fim do serviço (ping do CER1 para o CER7).
- Capturar os pacotes OpenFlow do controlador Ryu trocados com os nós OSHI. Verificar a versão do OpenFlow (versão 1.3).
- Verificar os encapsulamentos no ingresso e no egresso no pe02 e pe06 do serviço de pseudo fio conforme Figura 5.7.
- Verificar os rótulos MPLS no pe02, cr03, cr04, cr05 e pe06.

Passo 9: Apagar os circuitos virtuais e fechar os terminais, parar a emulação do Mininet para próximas práticas:

root@OSHI-VM:~/workspace/Dreamer-VLL-Pusher/ryu# ./vll_pusher.py --controller localhost:8080 --del

Fechar os terminais : exit ou ctrl + D;

Parar a emulação na linha de comando:exit ou ctrl + D.

EXPERIMENTO 3: CRIAÇÃO DE UM SERVIÇO EM REDE MPLS TE COM ROTEAMENTO POR SEGMENTOS ATRAVÉS DA FERRAMENTA OSHI

5.5. Criação de um serviço em rede MPLS TE com Roteamento por Segmentos através do OSHI-TE

Vamos considerar o exemplo da Figura 5.9, que leva em conta uma rede MPLS. O Roteamento por Segmentos é baseado na inserção,retirada e comutação de rótulos MPLS que representam os segmentos. No exemplo da figura, o nó C e F anunciam seus segmentos globais (segmentos de nó) 69 e 90 com os endereços de suas interfaces ``loopback" para os outros nós através do IGP. No nó B o pacote com rótulo 69 no topo da sua pilha de rótulos, significa que deve ser retirado e encaminhado pelo enlace B-C, atualizando o plano de encaminhamento de dados MPLS através de regras de encaminhamento. O nó E anuncia o rótulo 23 com segmento de adjacência e C instala também uma regra de encaminhamento no plano de encaminhamento MPLS. Se o nó A deseja enviar um pacote para F cria uma lista de segmentos que contém os segmentos {90, 23, 69}. Os segmentos A-C e E-F correspondem os menores caminhos de A a C e de E a F, e são os caminhos encontrados pelo IGP para alcançar o destino.

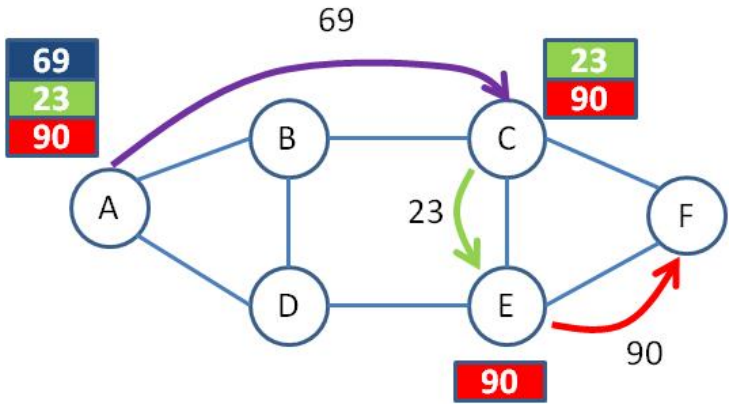


Figura 5.9 – Caso de uso de Roteamento por Segmentos simplificado.

O projeto de rede com nós OSHI pode acomodar o Roteamento por Segmentos graças à característica híbrida IP/SDN da arquitetura do nó. Algumas premissas foram consideradas no ajuste da ferramenta para Roteamento por Segmentos: os segmentos utilizam rótulos MPLS, segmentos locais não são suportados, são utilizados os 16 bits mais significativos à direita correspondente à interface ``loopback" para codificar o rótulo, cada interface OSHI em um nó tem o mesmo endereço físico (MAC address) e existe um mapeamento estático de MAC entre os nós OSHI usados para Roteamento por Segmentos, a ação de retirada do rótulo ocorre no último nó OSHI e não no penúltimo nó (*Penultimate Hop Pop* - PHP). Como na ferramenta não existe um plano de dados e de controle MPLS, este pode ser replicado utilizando tabelas OpenFlow e o uso do de switches habilitadas com OpenFlow (*OpenFlow Capable Switch* - OFCS), este passo é necessário para se implementar o comportamento do roteamento por segmentos nos nós OSHI, duas propostas foram consideradas: a primeira utiliza um ``daemon" local de Roteamento por Segmentos, a segunda usa o controlador agindo de forma remota. Em cada nó OSHI habilitado para Roteamento por Segmentos, o

``daemon' 'interage com o roteamento do nó obtendo as atualizações da tabela de roteamento e resultando na programação das tabelas OpenFlow com as regras necessárias para suportar o encaminhamento de pacotes baseado em Roteamento por Segmentos. Com o ``daemon'' de Roteamento por Segmentos é possível encaminhar pacotes na rede, no entanto não é possível desempenhar as funções de ingresso e egresso de pacotes na rede. No exemplo da Figura 5.9, o nó A deve criar uma lista de segmentos e o nó de egresso deve saber encaminhar o pacote adequadamente a cada interface, na ferramenta estas regras são implementadas no controlador, permitindo utilizar os serviços básico de VLL, PW e VSS com Roteamento por Segmentos. O ´´Daemon'' de Roteamento por Segmentos estende a implementação do gerenciador do plano de encaminhamento do Quagga (*Foward Plane Manager* - FPM) com as funcionalidades necessárias para gerenciar as tabelas OpenFlow na switch habilitada com OpenFlow (OpenFlow Capable Switch - OFCS), conforme ilustrado na Figura 5.10:

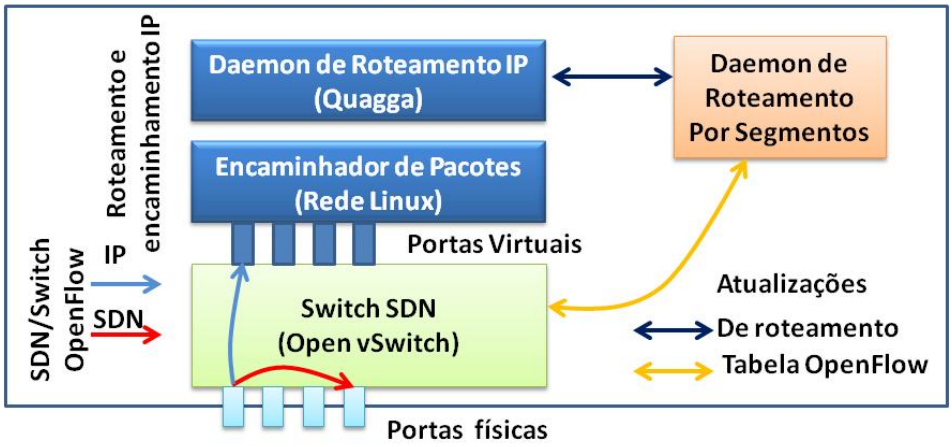


Figura 5.10 – Arquitetura do nó OSHI-TE para SR

A seguir são enumerados os passos para a criação de um serviço de circuito virtual alugado (*Virtual Leased Line* - VLL), que pode ser executado através da linha de comando ou da interface gráfica do conjunto de ferramentas Mantoo (Topology 3D):

Neste exemplo será analisada uma topologia gerada pela interface gráfica (Topology 3D) por um analisador e gerador (*parser generator*), que irá extrair da topologia do VLL o conjunto de fluxos e alocar segmentos no caminho de cada fluxo para então implantar o caminho de Roteamento por Segmentos no emulador Mininet, para emulação desta topologia.

Passo 1: verificar se as extensões do Mininet estão preparadas para Roteamento por Segmentos: no arquivo nodes.py verificar se a instrução SET ENABLE_SEGMENT_ROUTING = True na linha 32 do arquivo no diretório .

Passo 2: Na interface gráfica (Topology 3D), carregar o exemplo do diretório /home/user/workspace/sdn-te-sr-tools/parsers-generators/t3d/small-topo2-4-vll.t3d

Passo 3: Implantar a topologia através do comando ``deploy''.

Passo 4: Identificar o endereço IP do controlador e executar o ´´script'' de implantação e executar o controlador no console da VM (a senha do root é ´´root''):

user@OSHI-VM:~$ ssh -X root@10.255.245.1
root@OSHI-VM:~$ # ./ryu_start.sh

Ou manualmente:

user@OSHI-VM:~$ # cd /home/user/workspace/dreamer-ryu/ryu/app
user@OSHI-VM:~$ # ryu-manager rest_topology.py ofctl_rest.py --observe-links

Passo 5: Gerar um catálogo de fluxos para ser manuseado pelo algoritmo de alocação de Roteamento por Segmentos, trocando o endereço IP do controlador. De uma segunda console na VM, mover os arquivos de topologia e fluxos da pasta ``java-te-sr project'':

user@OSHI-VM:~$ $ cd /home/user/workspace/Mantoo-scripts-and-readme
user@OSHI-VM:~$ $ ./generate_topo_and_flow_cata.sh 10.255.245.1:8080

Ou manualmente:

user@OSHI-VM:~$ cd /home/user/workspace/sdn-te-sr-tools/parsers-generators
user@OSHI-VM:~$ python parse_transform_generate.py --in ctrl_ryu --out nx --generate_flow_cata_from_vll_pusher_cfg --controller 10.255.245.1:8080
user@OSHI-VM:~$ mv flow_catalogue.json ../java-te-sr/flow/
user@OSHI-VM:~$ mv links.json ../java-te-sr/topology/
user@OSHI-VM:~$ mv nodes.json ../java-te-sr/topology/

Checar o catálogo de fluxos gerado:

user@OSHI-VM:~$ cat /home/user/workspace/sdn-te-sr-tools/parsers-generators/flow_catalogue.json

Passo 6: Executar o algoritmo de Roteamento por Segmentos: abrir a ferramenta ´´Eclipse'' a partir da interface gráfica do OSHI, selecionando a aba Development, Eclipse. Clicar no projeto JAVA ``UniPR-SDN-TE-SR'' project executando Run as, Run Configurations, na ``Arguments Tab'' editar os seguintes argumentos do programa:

topo_in=topology/links.json
topo_out=topology/links.json.out
flows_in=flow/flow_catalogue.json
flows_out=flow/flow_catalogue.json.out
```

**Passo 7**: Executar o projeto com os argumentos modificados, clicando com o botão direito do mouse no projeto ``UniPR-SDN-TE-SR project'', `Run as-> Run Configurations, clicar no botão de execução. A principal classe do programa é a ``it.unipr.netsec.sdn.run.Main'', e não retorna nenhuma mensagem caso a compilação e execução do programa estiver sem problemas.

**Passo 8**: Mover o arquivo do catálogo de fluxos flow_catalogue.json para o OSHI-SR-pusher e executar o aplicativo sr_vll_pusher
```

user@OSHI-VM:~$ cd /home/user/workspace/Mantoo-scripts-and-readme
user@OSHI-VM:~$ ./sr_pusher_start.sh 10.255.245.1:8080 --add

Ou manualmente:

```
user@OSHI-VM:~$ cd /home/user/workspace/sdn-te-sr-tools
user@OSHI-VM:~$ mv java-te-sr/flow/flow_catalogue.json.out OSHI-SR-
pusher/out_flow_catalogue.json
user@OSHI-VM:~$ cd /home/user/workspace/sdn-te-sr-tools/OSHI-SR-pusher/
user@OSHI-VM:~$ rm sr_vlls.json
user@OSHI-VM:~$ ./sr_vll_pusher.py --controller 10.255.245.1:8080 --add
```

Passo 9: Agora é possível realizar o testes de conectividade no VLL, por exemplo executar o comando ping entre o CER1 e CER9.

Iniciar o Wireshark para captura de pacotes no nó OSHI PE (pe02 e pe06):

```
root@OSHI-VM:~$ wireshark &
```

- Verificar a continuidade fim a fim do serviço (ping do CER1 para o CER9).
- Capturar os pacotes OpenFlow do controlador Ryu trocados com os nós OSHI. Verificar a versão do OpenFlow (versão 1.3).
- Verificar os segmentos no pe02.
- Verificar os rótulos MPLS no pe02, cr03, cr04, cr05 e pe06.

Passo 10: Apagar os circuitos virtuais e fechar os terminais, parar a emulação do Mininet.

Fechar os terminais : exit ou ctrl + D;

Parar a emulação na linha de comando:exit ou ctrl + D.

5.6. Engenharia de Tráfego através do OSHI-TE

Para implementação de engenharia de tráfego (*Traffic Engineering* - TE) a ferramenta OSHI utiliza um aplicativo em Python (em protótipo) que influencia as decisões do controlador Ryu através de uma API REST. Na entrada é necessário um arquivo de configuração no formato JSON que descreve as relações do tráfego, enquanto que a topologia e capacidade dos enlaces são obtidas da API REST do módulo de topologia e do módulo do switch habilitado com OpenFlow (*OpenFlow Capable Switch* - OFCS), que provê a topologia e velocidade das portas. A implementação do TE é dividida em três partes: a obtenção das entradas, algoritmos baseados em heurísticas e a instalação de regras. O último passo é alcançado pela API REST ``Flowentry/Add'' do módulo OFCS, que permite implantar as regras nos switches OpenFlow. A Figura 5.11 ilustra este protótipo.

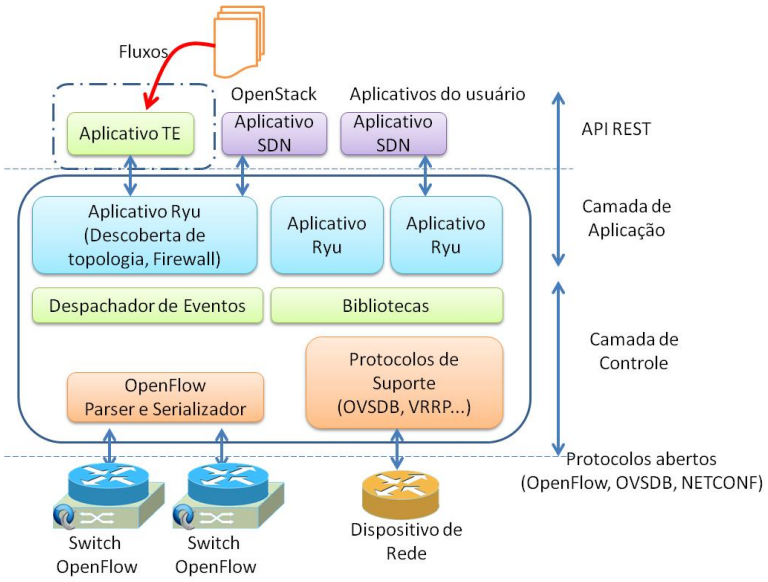


Figura 5.11 – Arquitetura de implementação do protótipo de TE.