

## Laboratório (Switch Ethernet e Tecnologia P4)

André Madureira

Leobino Sampaio

26 de Outubro de 2019

## Sumário

<b>1</b>	<b>Máquina Virtual</b>	<b>1</b>
<b>2</b>	<b>Switch Ethernet</b>	<b>2</b>
<b>3</b>	<b>Análise dos Códigos Fonte</b>	<b>7</b>

# 1 Máquina Virtual

Para realizar a parte prática deste trabalho, vocês devem primeiro configurar a máquina virtual abaixo, conforme os seguintes passos:

1. Baixe a máquina virtual disponível no link <https://mega.nz/#F!ow5i0aSC!e-LDfLc4cwh1oNC4yrwMkg>
2. Abra o VirtualBox
3. Abra o Menu “Arquivo -> Importar Appliance” (vide figura 1)
4. Escolha o arquivo da Máquina Virtual
5. Inicie a Importação

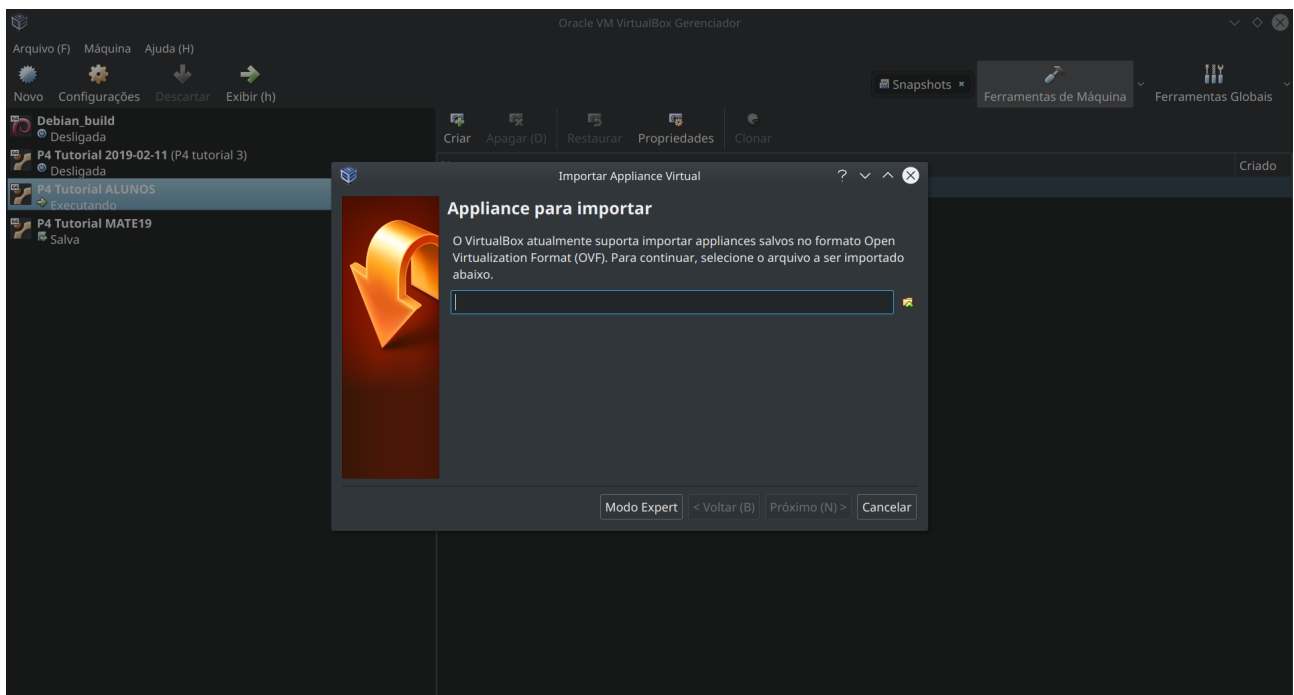


Figura 1: Janela de Importação de Appliance do VirtualBox

## 2 Switch Ethernet

Nesta etapa do trabalho, veremos o funcionamento de um switch ethernet. Para isso, siga os seguintes passos:

1. Inicie a máquina virtual
2. Faça o login usando o usuário **p4** e senha **p4** (vide figura 2)

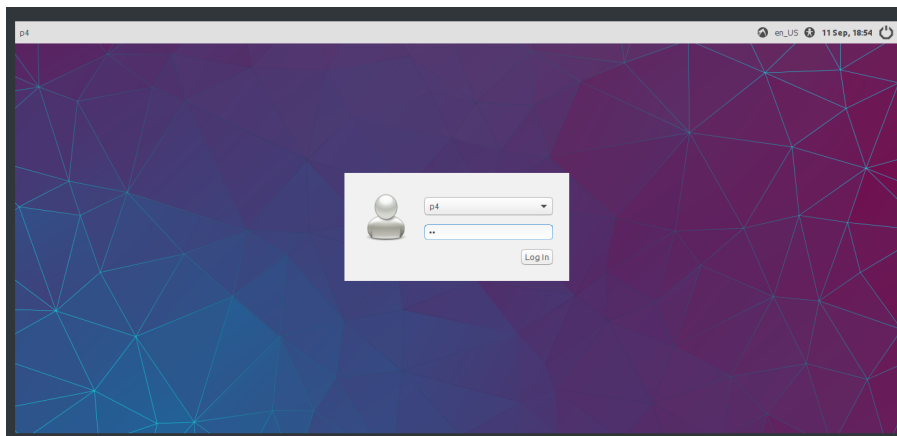
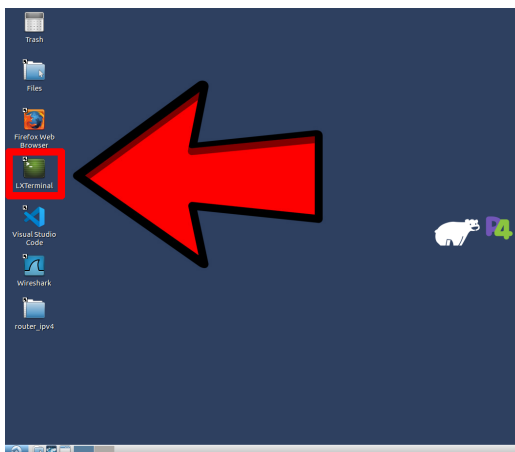
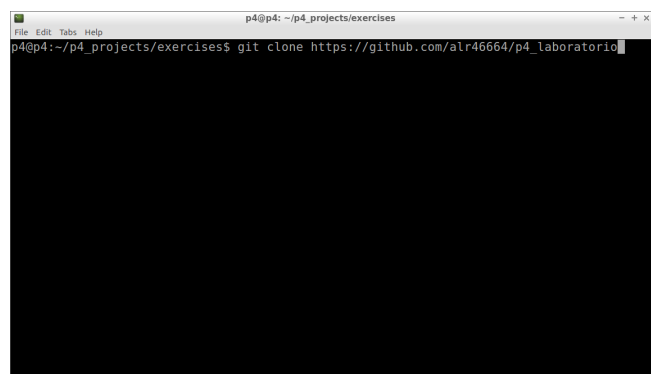


Figura 2: Login na máquina virtual

3. Abra o programa **LXTerminal** que está na área de trabalho (vide figura 3a)
4. Digite os comandos (vide figura 3b):  
`git clone https://github.com/alr46664/p4_laboratorio`  
`cd p4_laboratorio`  
`bash install.sh`



(a) Terminal Linux



(b) Comando **git**

Figura 3: Download do Repositório do GitHub

5. Na Área de Trabalho deverá aparecer a pasta **p4\_laboratorio**, abra a pasta (vide figura 4a)
6. Em seguida abra o arquivo **vscode.code-workspace** (vide figura 4b)
7. No Visual Studio Code, execute o comando **./run.sh** no terminal (vide figura 5). Este comando abrirá o ambiente de simulação Mininet, onde realizaremos testes com o switch ethernet.

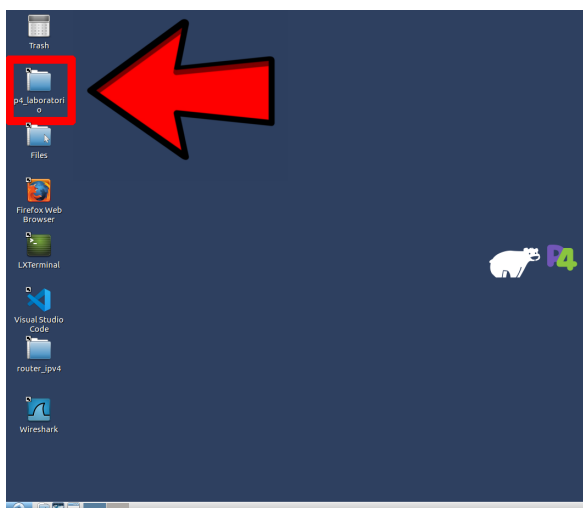
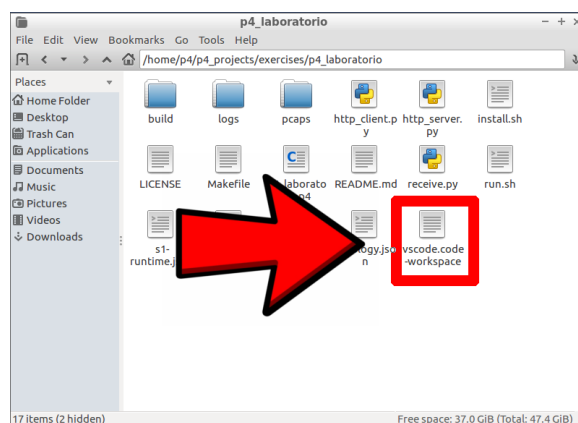
(a) Pasta **p4\_laboratorio**(b) Arquivo **vscode.code-workspace**

Figura 4: Pasta do Projeto Laboratório

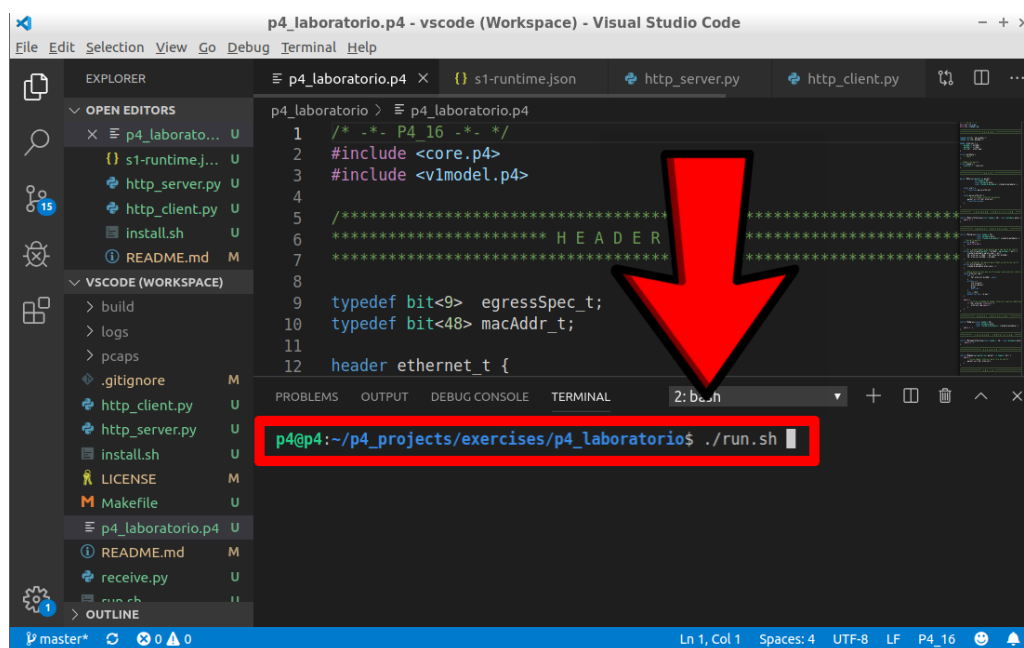


Figura 5: Ambiente do Visual Studio Code

8. O prompt **mininet>** aparecerá na sua tela. Nesse prompt, digite o comando **xterm h1 h2** (vide figura 6). Este comando abre dois terminais, que pertencem a duas máquinas virtuais que o Mininet criou (**h1** e **h2**). Estas máquinas estão conectados ao switch ethernet descrito no arquivo **p4.laboratorio.p4**. Dessa forma, podemos testar esse switch ao enviarmos pacotes de **h1** para **h2** e vice-versa.

```

mininet> xterm h1 h2

```

To view a switch log, run this command from your host OS:  
`tail -f /home/p4/p4_projects/exercises/router_ipv4/logs/<switchname>.log`

To view the switch output pcap, check the pcap files in /home/p4/p4\_projects/exercises/router\_ipv4/pcaps:  
for example run: `sudo tcpdump -xxx -r s1-eth1.pcap`

To view the P4Runtime requests sent to the switch, check the  
corresponding txt file in /home/p4/p4\_projects/exercises/router\_ipv4/logs:  
for example run: `cat /home/p4/p4_projects/exercises/router_ipv4/logs/s1-p4runtime-requests.txt`

Figura 6: Ambiente Mininet

9. Dois terminais aparecerão na tela. No terminal **h2**, digite o comando **wireshark**. O Wireshark aparecerá na sua tela. Com o Wireshark aberto, aperte o botão Capturar Pacotes (vide figura 7).
10. Ainda no terminal **h2**, digite o comando **./http\_server.py 8888** (vide figura 8).
11. No terminal **h1**, digite o comando **./http\_client.py 8888 http://10.0.1.2/**, conforme mostra a figura 8.

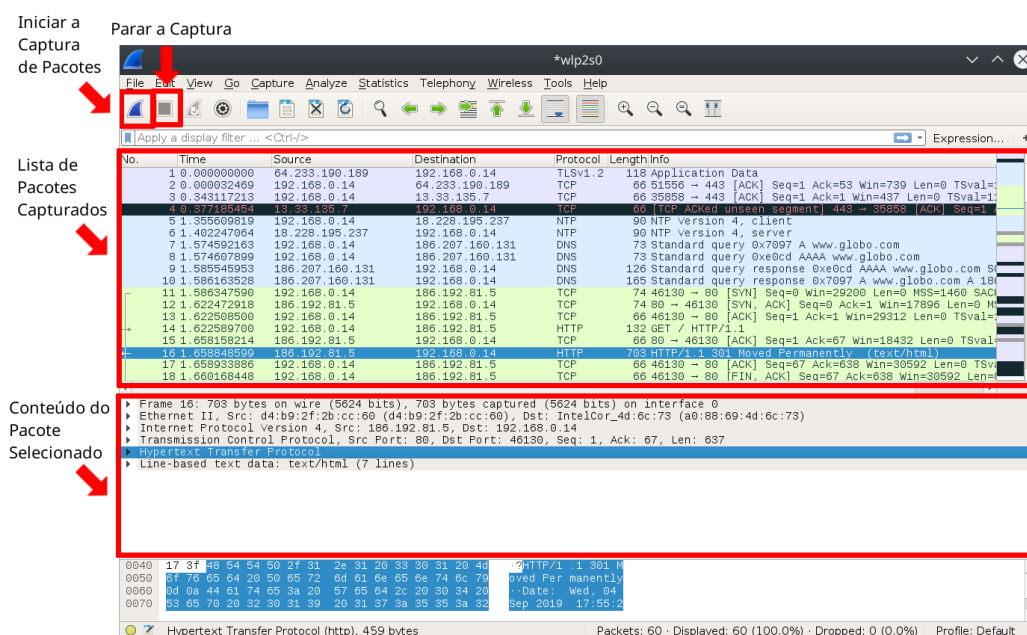
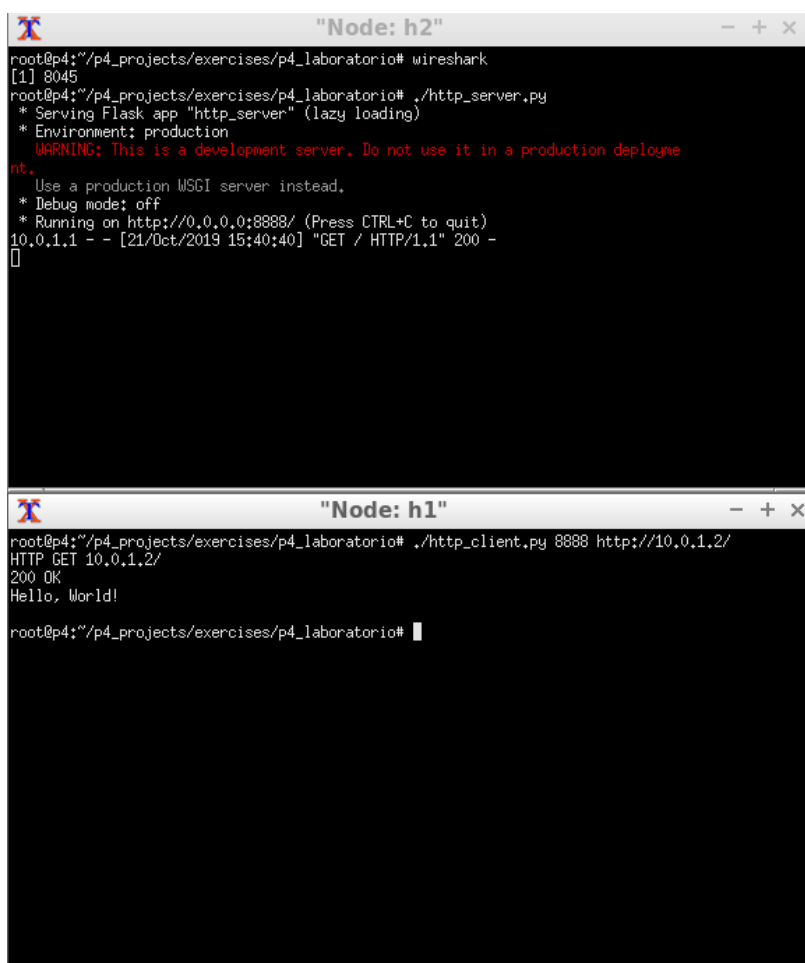


Figura 7: Wireshark



The image shows two terminal windows. The top window, titled "Node: h2", shows the execution of the Flask application. The user runs 'wireshark' and then './http\_server.py'. The output shows the server starting on port 8888 and receiving a GET request from 10.0.1.1. The bottom window, titled "Node: h1", shows the execution of the HTTP client. The user runs './http\_client.py 8888 http://10.0.1.2/'. The output shows the client sending a GET request to 10.0.1.2 and receiving a 200 OK response with the text 'Hello, World!'.

```
root@p4:~/p4_projects/exercises/p4_laboratorio# wireshark
[1] 8045
root@p4:~/p4_projects/exercises/p4_laboratorio# ./http_server.py
* Serving Flask app "http_server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8888/ (Press CTRL+C to quit)
10.0.1.1 - - [21/Oct/2019 15:40:40] "GET / HTTP/1.1" 200 -

root@p4:~/p4_projects/exercises/p4_laboratorio#

root@p4:~/p4_projects/exercises/p4_laboratorio# ./http_client.py 8888 http://10.0.1.2/
HTTP GET 10.0.1.2/
200 OK
Hello, World!

root@p4:~/p4_projects/exercises/p4_laboratorio#
```

Figura 8: Terminais XTERM e Comandos

12. **IMPORTANTE:** Perceba que o servidor HTTP que estamos utilizando está rodando na porta 8888 e não na porta padrão do HTTP (porta 80).
13. Agora iremos testar o servidor HTTP usando o Firefox. Para isso, no terminal **h1**, digite o comando **firefox**. No Firefox, digite a URL **10.0.1.2:8888**. Deve aparecer no Firefox uma resposta como a que consta na figura 9.
14. Ao executar os comandos acima, veremos na tela a comunicação entre um Cliente HTTP e um Servidor, bem como os pacotes enviados e recebidos nessa comunicação (veja os pacotes capturados pelo Wireshark), conforme figura 10.



Figura 9: Teste com Firefox

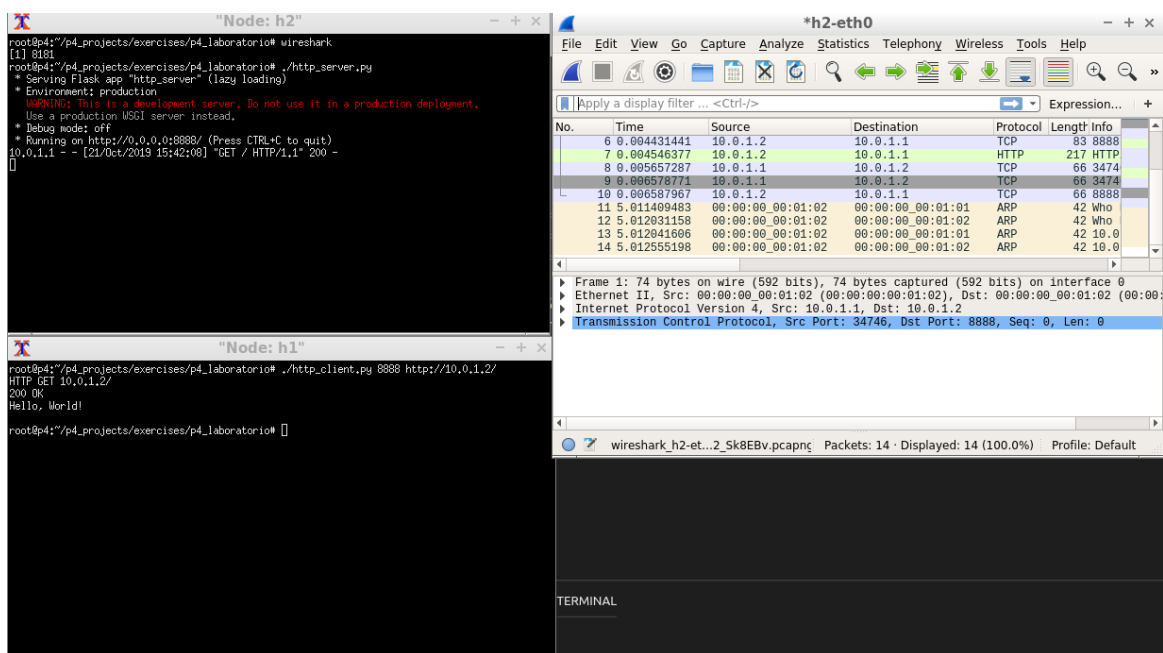


Figura 10: Teste com Wireshark

15. Agora, ao invés de utilizar o `http_client.py`, utilize o cliente HTTP que você construiu no Trabalho 01 da disciplina. Execute seu cliente no terminal `h1` e observe se seu cliente HTTP funciona de maneira parecida com o `http_client.py`. **IMPORTANTE:** Modifique a porta do seu cliente HTTP para a porta 8888.



### 3 Análise dos Códigos Fonte

1. Nesta parte do trabalho iremos analisar os códigos fonte dos programas utilizados neste laboratório. Ao abrirmos o arquivo **p4\_laboratorio.p4**, percebemos que o P4 não compreende o protocolo IP, porém, ainda assim, consegue enviar e receber pacotes. Isso ocorre, pois o **p4\_laboratorio.p4** descreve o funcionamento de um switch ethernet. Um switch não compreende os protocolos de roteamento (camada 3), como o IP. Porém o switch consegue realizar o encaminhamento de pacotes através do endereço de destino da camada de enlace (camada 2).
2. Podemos perceber no Wireshark que, para uma aplicação HTTP funcionar, o TCP Handshake é utilizado. Além disso, percebemos que há a formação de uma conexão TCP (pacotes TCP SYN). Como o HTTP utiliza sockets TCP, o HTTP não precisa se preocupar com os serviços de envio confiável de dados, controle de congestionamento e afins. Tudo isso é gerenciado pelo protocolo TCP, dentro do sistema operacional.
3. Vale ressaltar que é da responsabilidade do protocolo de aplicação (no nosso caso, o HTTP) estabelecer como o Cliente e o Servidor devem se comunicar. Para isso, o HTTP utiliza os métodos GET, POST, HEAD e outros descritos na RFC 2616. Leia o código fonte **http\_server.py** para compreender melhor como o servidor responde a uma solicitação de um cliente HTTP.
4. O código **p4\_laboratorio.p4** define o que PODE SER executado (o P4 trabalha com **actions**) e COMO SERÁ executado (o P4 descreve a sequência de passos que uma **action** deve executar). Perceba que uma **action** é bastante semelhante a uma função de uma linguagem de programação convencional. Já a estrutura do P4 chamada de **table** pode ser vista como um **switch case** de uma linguagem de programação convencional, que escolherá qual **action** executar.
5. O arquivo **s1-runtime.json** configura as **tables** do P4 para determinar as condições necessárias para a execução de uma determinada **action**. Ou seja, o arquivo **s1-runtime.json** define O QUE SERÁ executado (**action**) e QUANDO SERÁ executado (chamamos isso de **match**). Assim, para o código **p4\_laboratorio.p4** funcionar, é necessário que o arquivo de configuração **s1-runtime.json** também esteja correto.
6. Compare o código **p4\_laboratorio.p4** com o **s1-runtime.json**. Tente entender como eles funcionam. Veja quais são as suas diferenças e como eles se complementam. Para facilitar o entendimento desses conceitos, pesquise sobre **Redes Definidas por Software**, também chamadas de **Software Defined Networks**.
7. **DICA:** Não tente utilizar o código do **http\_server.py** para o Trabalho Semestral. Tanto o **http\_server.py** como o **http\_client.py** utilizam FRAMEWORKS e BIBLIOTECAS para implementar o servidor e cliente HTTP. No trabalho semestral vocês devem implementar o cliente e servidor utilizando Sockets.
8. Quanto terminar de realizar seus testes, digite o comando **exit** no prompt **mininet>** para fechar o ambiente Mininet.