

MININET

ASSIS TIAGO DE OLIVEIRA FILHO

ASSIS.TIAGO@GPRT.UFPE.BR

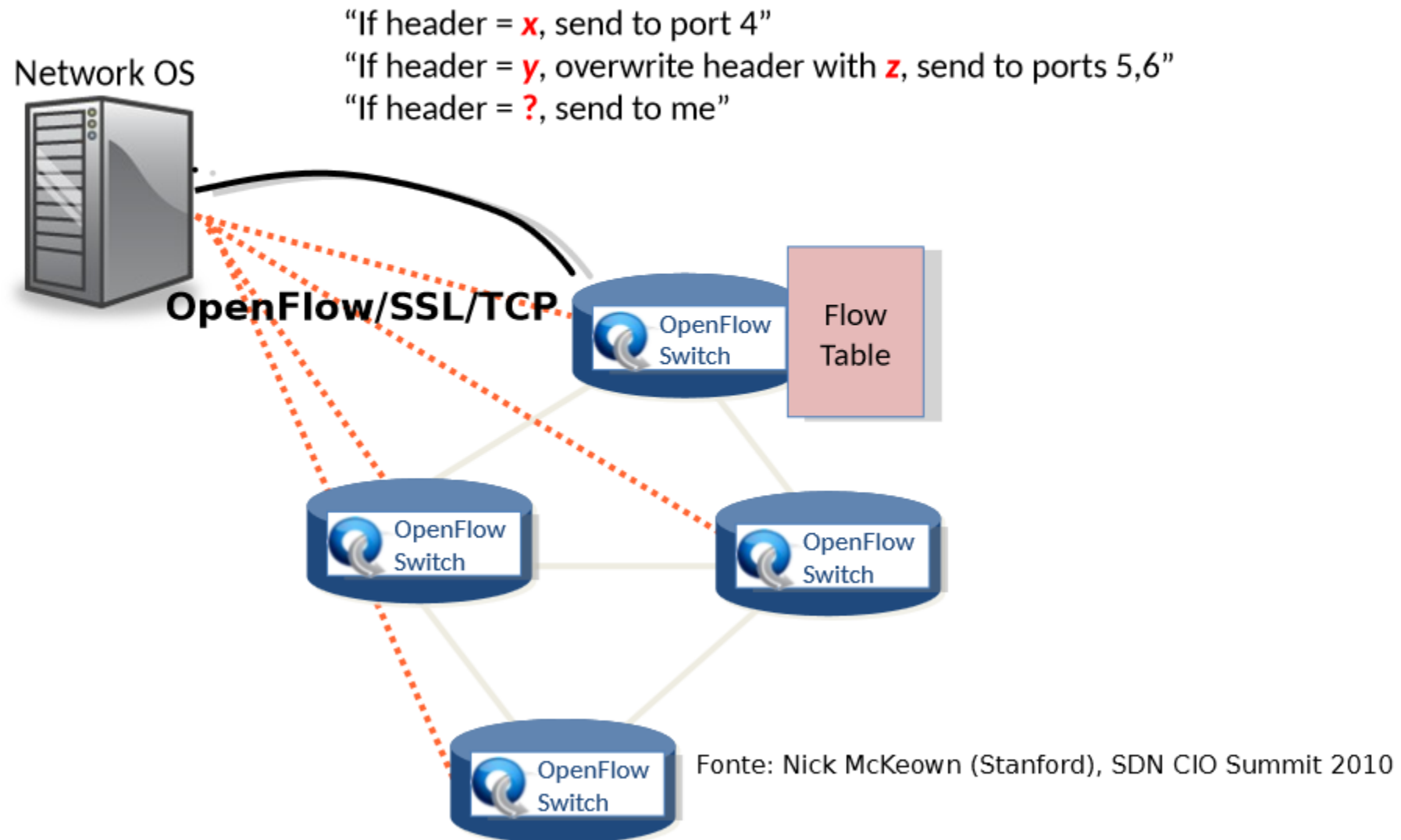
ATOF@CIN.UFPE.BR

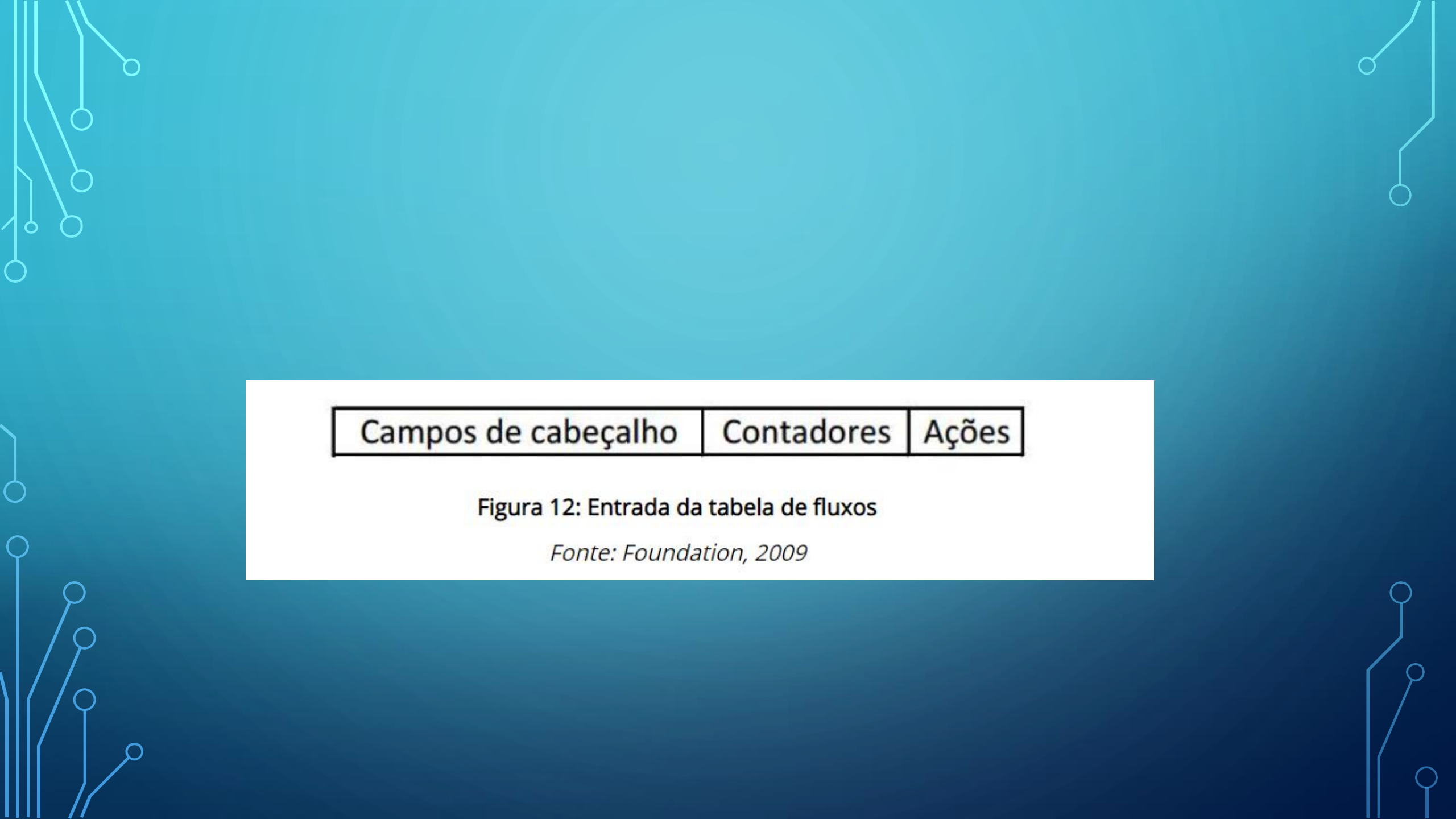


Agenda

- ▶ Aula 01: conceitos de OpenFlow, prática de captura de pacotes, alteração do datapath de forma pró-ativa
- ▶ **Aula 02: controlador OpenFlow, exemplos, APIs, bibliotecas e aplicações de apoio**
 - Exercício
- ▶ Aula 03: construção de aplicação L2 multi-switch com Ryu/Mininet
- ▶ Aula 04: prática com switches reais, conceitos de slices

Como OpenFlow funciona





| Campos de cabeçalho | Contadores | Ações |
|---------------------|------------|-------|
|---------------------|------------|-------|

Figura 12: Entrada da tabela de fluxos

Fonte: Foundation, 2009

COMO OPENFLOW FUNCIONA

- Cada entrada na tabela de fluxos é associada a uma ou mais ações. Se para uma determinada entrada na tabela não houver uma ação especificada, os pacotes desse fluxo serão descartados.

ABAIXO SEGUEM OS TIPOS DE AÇÕES:

- Encaminhamento
 - Obrigatório
 - ALL - Envia o pacote para todas as interfaces, exceto a interface de entrada;
 - CONTROLLER - Encapsula e envia o pacote para o controlador;
 - LOCAL - Envia o pacote para a pilha de rede local;
 - TABLE - Realiza ações na tabela de fluxos;
 - IN_PORT - Envia o pacote para a porta de entrada;
 - Opcional
 - NORMAL - Processa o pacote utilizando um encaminhamento tradicional;
 - FLOOD - Inunda o pacote, sem incluir a interface de entrada, levando em consideração o Spanning Tree.

ABAIXO SEGUEM OS TIPOS DE AÇÕES:

- Enfileirar (opcional) - Encaminha o pacote através de uma fila relacionada a uma porta;
- Descartar (obrigatória);
- Modificar campo (opcional):
 - Setar Vlan ID
 - Setar Vlan Priority
 - Separar o cabeçalho da Vlan
 - Modificar endereço MAC (\textit{Media Access Control}) de origem
 - Modificar endereço MAC de destino
 - Modificar endereço IP de origem
 - Modificar endereço IP de destino
 - Modificar ToS
 - Modificar a porta de transporte de origem
 - Modificar a porta de transporte de destino

O PROTOCOLO OPENFLOW SUPORTA TRÊS TIPOS DIFERENTES DE MENSAGENS:

- Controlador-Switch - Geradas pelo controlador para gerenciar e inspecionar o estado de um switch;
- Assíncronas - Geradas pelo switch para atualizar o controlador sobre eventos da rede e mudanças no estado do switch;
- Simétricas - Podem ser geradas tanto pelo controlador quanto pelo switch. São enviadas sem solicitação;

A FIGURA APRESENTA O FLUXOGRAMA REFERENTE A ENTRADA DE UM PACOTE EM UM SWITCH OPENFLOW.

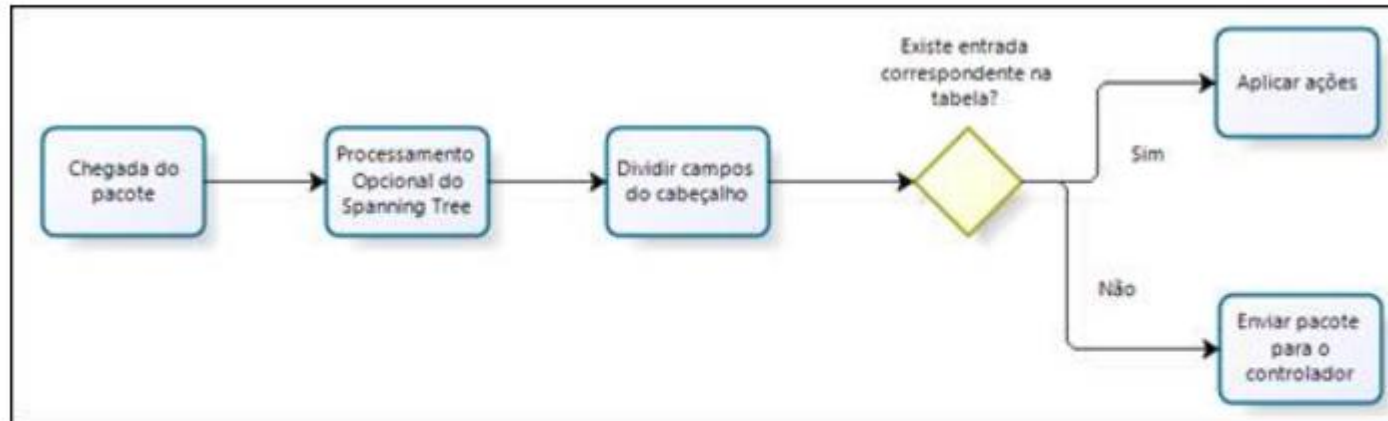


Figura 15: Fluxograma do processamento dos pacotes

Fonte: Foundation, 2009

O CANAL SEGURO

- Outra parte essencial da caracterização de um switch OpenFlow é o estabelecimento de um canal seguro com o controlador. Através desse canal, o controlador irá configurar e gerenciar o switch OpenFlow.

TIPOS DE MENSAGEM

- Controlador-Switch

- Características (Features) - O controlador requisita as características do switch. O switch deve responder com as características suportadas;
- Configuração (Configuration) - Usado para configurar ou solicitar configurações do switch;
- Modificação de estado (Modify-State) - Usado para adicionar, deletar e modificar a tabela de fluxos e para setar propriedades nas portas do switch;
- Leitura de estado (Read-State) - Coleta estatísticas;
- Envio de pacote (Send-Packet) - Utilizado para enviar pacotes por uma determinada porta do switch;
- Barreira (Barrier) - Usado para garantir que as dependências foram atendidas ou para receber notificações de operações finalizadas;

Tipos de mensagem OpenFlow

- ▶ Controller-to-switch (algumas):
 - **Features**: quais capabilities o switch suporta?
 - **Modify-state**: add/delete/modify flows na tabela de flows do switch
 - **Read-State**: obter estatísticas da tabela de flows, portas ou flows individuais
 - **Send-Packet**: usado pelo controller para enviar pacotes para uma porta do switch

TIPOS DE MENSAGEM

- Assíncrona
 - Entrada de pacotes (Packet-In) - Utilizado quando fluxos não classificados entram no switch.
 - Remoção de fluxo (Flow-Removed) - Mensagem enviada para o controlador, quando um fluxo é removido da tabela. Seja por Idle Timeout, Hard Timeout ou por uma mensagem de modificação da tabela de fluxos que delete a entrada em questão;
 - Estado da porta (Port-Status) - Mensagem enviada para o controlador sempre que há mudanças nas configurações das portas;
 - Erro (Error) - Notificações de erros;

Tipos de mensagem OpenFlow

- ▶ Asynchronous (algumas):
 - **Packet-in:** switch encaminha pacote (ou cabeçalho) para o controller se não houver uma entrada correspondente previamente instalada na tabela de flows
 - **Flow-removed:** quando o timeout do flow expirou e ele foi removido da tabela de flows
 - **Port-Status:** switch informa ao controller sobre mudanças na configuração do estado das portas
 - **Error:** informa ao controller sobre erros diversos

TIPOS DE MENSAGEM

- Simétrica
 - Hello - Mensagens trocadas entre o controlador e o switch quando uma conexão é estabelecida;
 - Echo - Mensagens usadas para identificação de latência, largura de banda e existência de conectividade;
 - Vendor - Provêem uma forma padrão para os switches OpenFlow oferecerem funcionalidades adicionais;

PARA MODIFICAR AS TABELAS DE FLUXOS DOS SWITCHES, O CONTROLADOR PODERÁ GERAR CINCO TIPOS DE MENSAGENS DIFERENTES.

```
enum ofp_flow_mod_command {  
    OFPFC_ADD,           /* New flow. */  
    OFPFC_MODIFY,        /* Modify all matching flows. */  
    OFPFC_MODIFY_STRICT, /* Modify entry strictly matching wildcards */  
    OFPFC_DELETE,        /* Delete all matching flows. */  
    OFPFC_DELETE_STRICT /* Strictly match wildcards and priority. */  
};
```

Figura 16: Tipos de mensagens para modificação da tabela de fluxos

Fonte: Foundation, 2009

Controlador OpenFlow

- ▶ O controlador OpenFlow se comunica com os switches através de um canal seguro
 - Objetivo: atualização da tabela de fluxo
 - A lógica é executada pelo controlador
- ▶ Fornece API (*Application Programming Interface*) para implementação de aplicações.
 - Mensagens OF são tratadas como eventos
- ▶ Diversas aplicações e bibliotecas de apoio: graph data structure, topology viewer, logging, link discovery, state sincronization, REST API, etc.

Pontos de atenção

- ▶ Linguagem de programação (possui ligação direta com a performance do controlador);
- ▶ Curva de aprendizado;
- ▶ Quantidade de usuários e comunidade de suporte;
- ▶ Bibliotecas e Aplicações de apoio
- ▶ Versão do OF
- ▶ Foco: Southbound/Northbound API; Educação, pesquisa ou produção?

Objetos de estudo

► POX:

- Suporta apenas a versão 1.0 do OpenFlow
- Python
- Largamente utilizado e suportado, curva de aprendizagem suave
- Desvantagem: Baixa performance

► Ryu:

- Suporta OpenFlow 1.0, 1.1, 1.2, 1.3 e extensões da Nicira;
- É um Framework para desenvolvimento de aplicações SDN, ao invés de um controlador monolítico
- Diversos componentes: openstack, snort, REST, Topology manager, HA

POX Estrutura básica

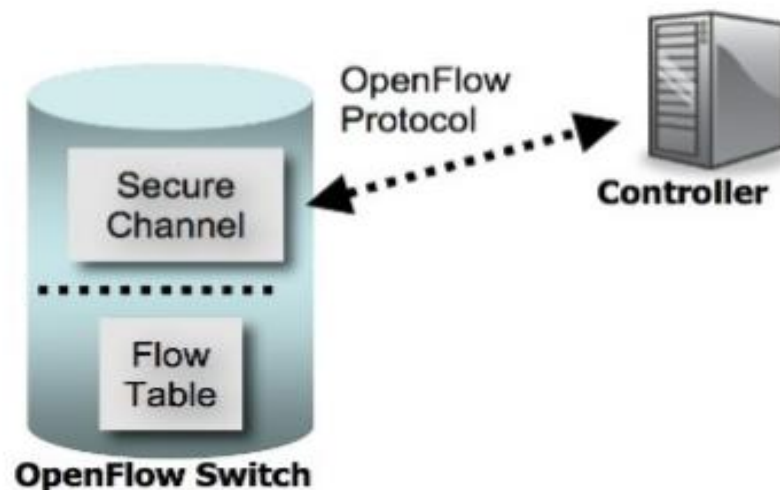
```
1 # ext/myfirstapp.py
2
3 # Importar as bibliotecas
4 from pox.core import core
5 import pox.openflow.libopenflow_01 as of
6 from pox.lib.revent import *
7
8 log = core.getLogger()      # logging
9
10 class myfirstapp (EventMixin):
11     switches = {}
12
13     def __init__(self):
14         self.listenTo(core.openflow)
15
16     def _handle_ConnectionUp (self, event):
17         log.debug("Connection UP from %s", event.dpid)
18         myfirstapp.switches[event.dpid] = event.connection
19
20     def _handle_PacketIn (self, event):
21         pass
22
23 def launch ():
24     core.openflow.miss_send_len = 1024
25     core.registerNew(myfirstapp)
```

Executando o POX

- ▶ `cd pox/`
- ▶ `python pox.py --verbose myfirstapp py \`
`log --no-default --file=/tmp/mylog.log`
 - `--verbose` → Exibe o modo debug do controlador
 - `openflow.of_01 --port=6634` → componente OF1.0
 - `log` → componente de logging
 - `py` → console python após iniciar o controlador

API do POX

- ▶ Criando uma mensagem entre controlador e o switch:
 - `msg = of.ofp_flow_mod()`



API do POX

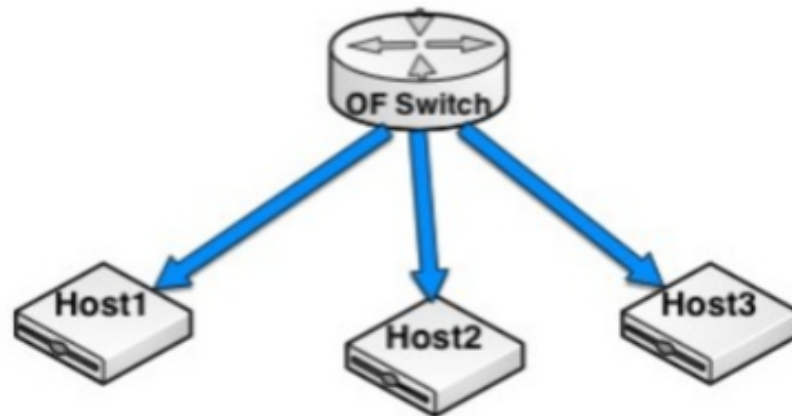
- ▶ Opções de casamento:
 - **match.in_port** → porta de entrada
 - **match.dl_src** → endereço MAC de origem
 - **match.dl_dst** → endereço MAC de destino
 - **match.dl_vlan** → ID da VLAN
 - **priority** → prioridade do Flow
 - **hard_timeout** → duração máxima do Flow no switch (em segundos)
 - **soft_timeout** → duração do Flow sem tráfego no switch (em segundos)

API do POX

- ▶ Algumas Actions:
 - `actions.append(of.ofp_action_output(port = 2))`
 - `actions.append(of.ofp_action_output(port = of.OFPP_ALL))`
 - `actions.append(of.ofp_action_vlan_vid(vlan_vid = 50))`

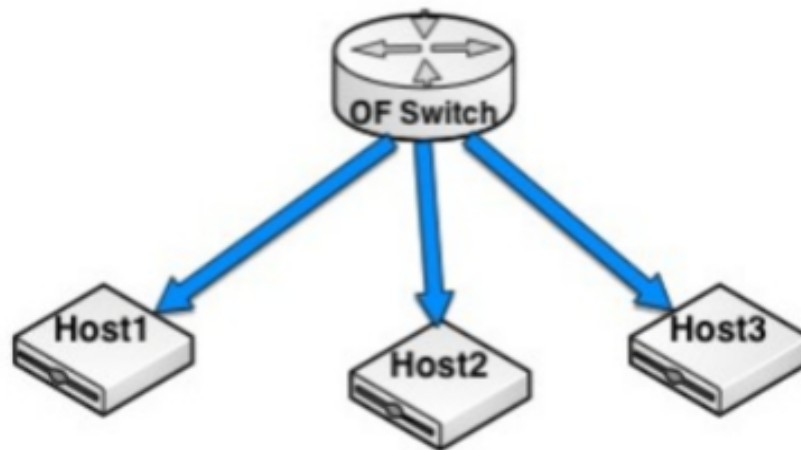
API do POX

- ▶ Enviando mensagens OpenFlow:
 - **connection.send(msg)**



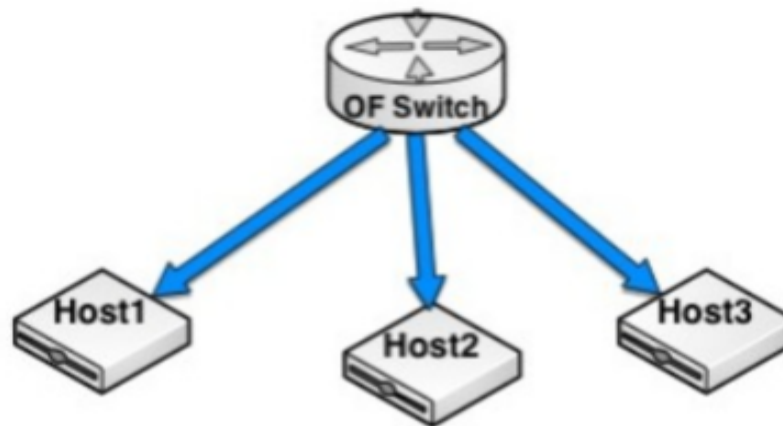
API do POX

- ▶ Tratando eventos:
 - **event.connection** → endereço do switch
 - **event.port** → porta do switch que gerou o evento



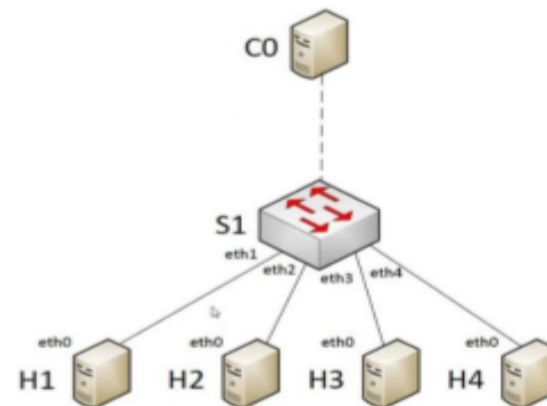
API do POX

- ▶ Parse dos eventos:
 - **packet = event.parsed**
 - **packet.src** → MAC address de origem
 - **packet.dst** → MAC address de destino



Exercicio 1

- ▶ Utilizar a aplicação “myfirstapp” para instalar, via console, fluxos de encaminhamento entre a porta 1 e 3 em uma topologia “single,4”
 - **python pox.py --verbose myfirstapp py log --no-default --file=/tmp/mylog.log**
 - **sudo mn --topo single,4 --mac --arp --controller remote**

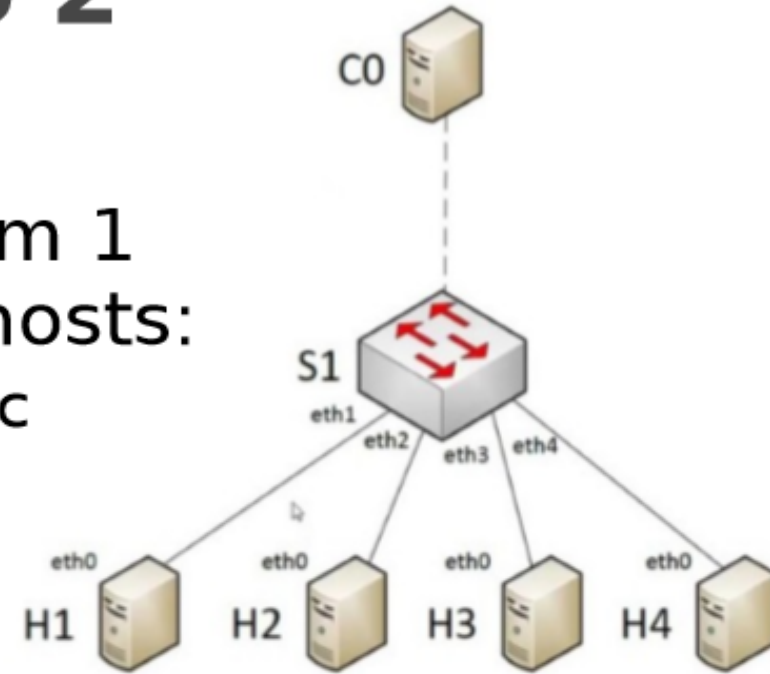


Exercicio 1

```
POX> import pox.openflow.libopenflow_01 as of
POX> from myfirstapp import myfirstapp
POX>
POX> msg = of.ofp_flow_mod()
POX> msg.match.in_port = 1
POX> msg.actions.append(of.ofp_action_output(port = 3))
POX> myfirstapp.switches[1].send(msg)
POX>
POX> msg = of.ofp_flow_mod()
POX> msg.match.in_port = 3
POX> msg.actions.append(of.ofp_action_output(port = 1))
POX> myfirstapp.switches[1].send(msg)
```


Exercício 2

- ▶ Criar topologia simples com 1 controlador, 1 switch e 4 hosts:
 - `sudo mn --topo single,4 --mac --arp --controller remote`
- ▶ Desenvolver código para refletir a FlowTable abaixo:

[illegible]

Exercício 2

```
1 # ext/aula2ex1.py
2
3 # Importar as bibliotecas
4 from pox.core import core
5 import pox.openflow.libopenflow_01 as of
6 from pox.lib.revent import *
7 from pox.lib.addresses import EthAddr, IPAddr
8 from pox.lib.util import dpidToStr
9
10 log = core.getLogger()      # logging
11
12 class aula2ex1 (EventMixin):
13     def __init__(self):
14         self.listenTo(core.openflow)
15
16     def _handle_ConnectionUp (self, event):
17         log.debug("Connection UP from %s", event.dpid)
18
19     def _handle_PacketIn (self, event):
20         packet = event.parsed
21         # Drop de todos os pacotes
22         msg = of.ofp_flow_mod()
23         msg.match.in_port = event.port
24         msg.match.dl_src = packet.src
25         msg.match.dl_dst = packet.dst
26         event.connection.send(msg)
27         log.debug("Drop packet sw=%s in_port=%s src=%s dst=%s" \
28                 % (event.dpid, event.port, packet.src, packet.dst))
29
30 def launch ():
31     core.openflow.miss_send_len = 1024
32     core.registerNew(aula2ex1)
```