# Aula 02 – Controlador OpenFlow

MATE18 – Oficina de OpenFlow/SDN
Universidade Federal da Bahia

Italo Valcy <italovalcy@ufba.br>

12 de julho de 2017

# Licença de uso e atribuição

Todo o material aqui disponível pode, posteriormente, ser utilizado sob os termos da:

**Creative Commons License:**
**Atribuição - Uso não comercial - Permanência da Licença**

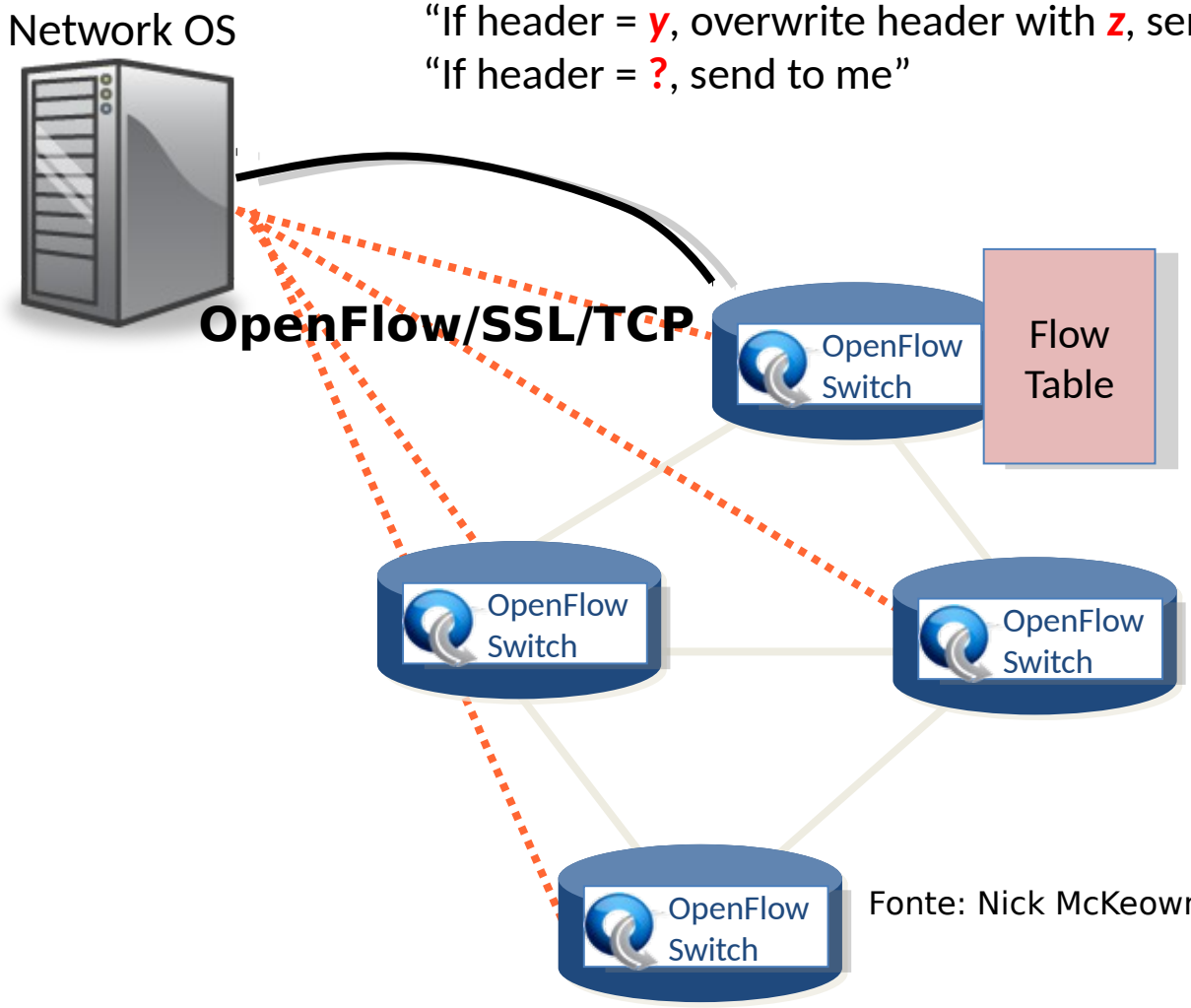

**http://creativecommons.org/licenses/by-nc-sa/3.0/**

# 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

"If header = **x**, send to port 4"
"If header = **y**, overwrite header with **z**, send to ports 5,6"
"If header = **?**, send to me"




Fonte: Nick McKeown (Stanford), SDN CIO Summit 2010

# 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 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

# 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.

# Exemplos de controladores

- NOX/POX
- Ryu
- Floodlight
- Kytos
- ONOS
- OESS
- OpenDayLight
- RouteFlow

# 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)
```

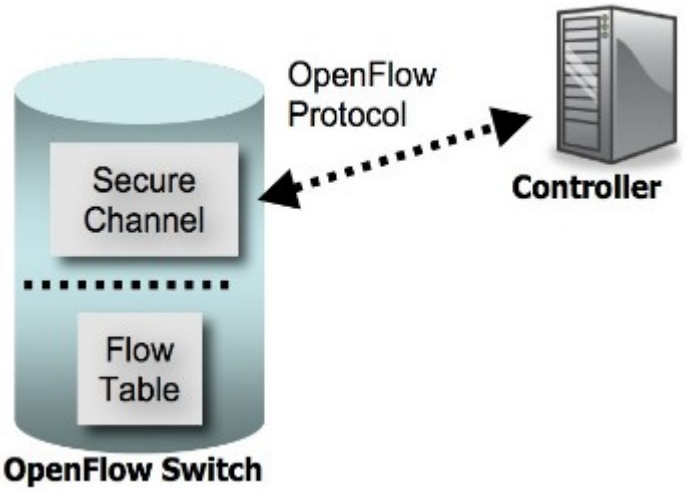
# Executando o POX

- 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

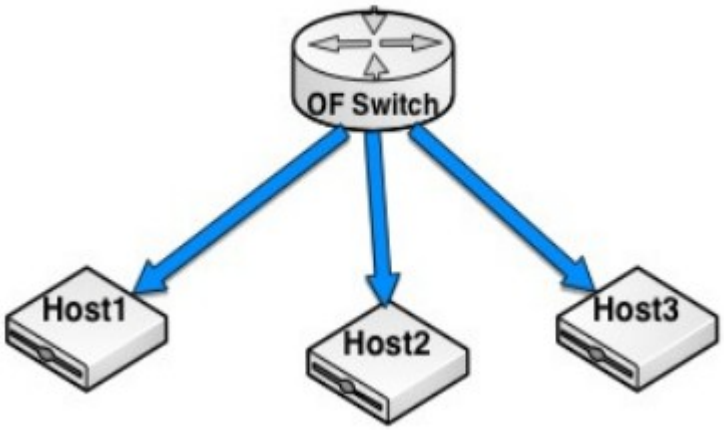
- Opções de casamento:
  - **match.in_port** → porta de entrada
  - **match.dl_src** → endereço MAC de origem
  - **match.dl_dst** → endereço MAC de destino
  - **match.dl_vlan** → ID da VLAN
  - **priority** → prioridade do Flow
  - **hard_timeout** → duração máxima do Flow no switch (em segundos)
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

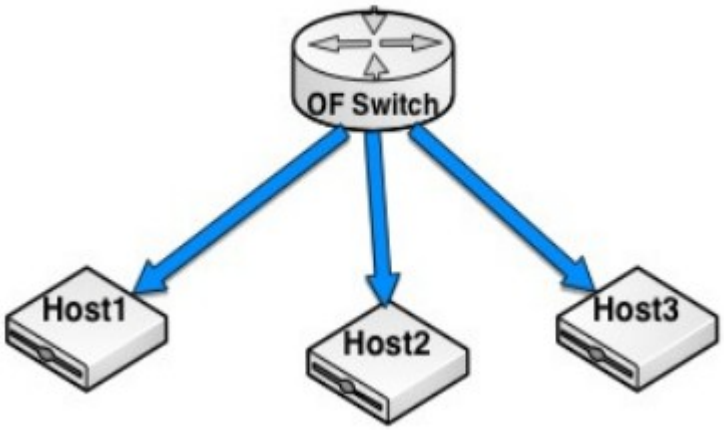
# API do POX

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



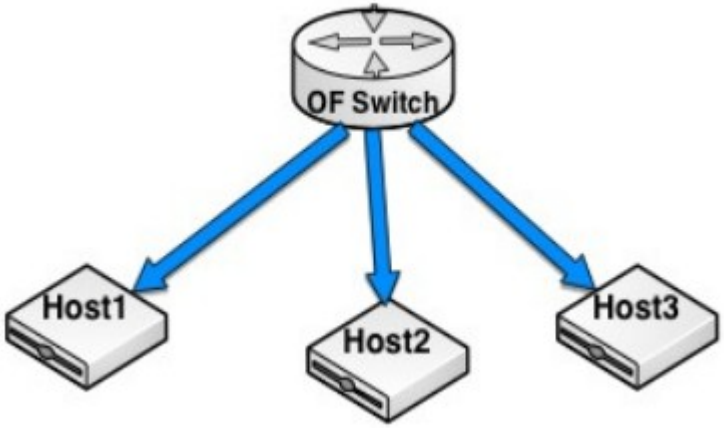
# API do POX

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



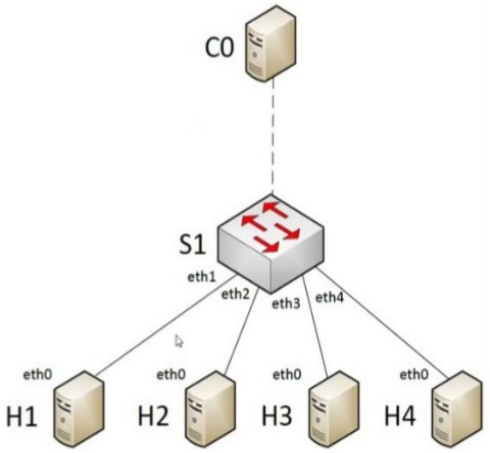
# API do POX

- 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**

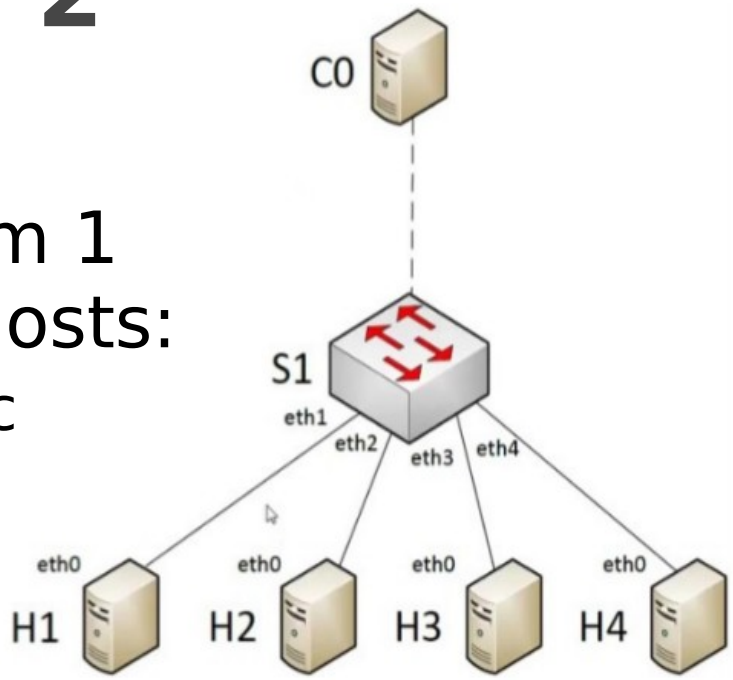


# Exercicio 1

```
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:

| Match | | | | | | | | | | | | | Ação |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prioridade | Porta de entrada | MAC de origem | MAC de destino | Tipo Ethernet | VLAN ID | VLAN Priority | IP de origem | IP de destino | ToS | Protocolo | Porta de origem | Porta de destino | Ação |
| - | 1 | - | - | - | - | - | - | - | - | - | - | - | drop |
| - | 2 | - | - | - | - | - | - | - | - | - | - | - | drop |
| - | 3 | - | - | - | - | - | - | - | - | - | - | - | drop |
| - | 4 | - | - | - | - | - | - | - | - | - | - | - | drop |

# Exercício 2

```python
# ext/aula2ex1.py

# Importar as bibliotecas
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.addresses import EthAddr, IPAddr
from pox.lib.util import dpidToStr

log = core.getLogger()     # logging

class aula2ex1 (EventMixin):
    def __init__(self):
        self.listenTo(core.openflow)

    def _handle_ConnectionUp (self, event):
        log.debug("Connection UP from %s", event.dpid)

    def _handle_PacketIn (self, event):
        packet = event.parsed
        # Drop de todos os pacotes
        msg = of.ofp_flow_mod()
        msg.match.in_port = event.port
        msg.match.dl_src = packet.src
        msg.match.dl_dst = packet.dst
        event.connection.send(msg)
        log.debug("Drop packet sw=%s in_port=%s src=%s dst=%s" \
              % (event.dpid, event.port, packet.src, packet.dst))

def launch ():
    core.openflow.miss_send_len = 1024
    core.registerNew(aula2ex1)
```

# Exercício 3

- Criar Flows de encaminhamento entre as portas 1 e 3. Os demais pacotes devem ser descartados.

| Match | | | | | | | | | | | | | Ação |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prioridade | Porta de entrada | MAC de origem | MAC de destino | Tipo Ethernet | VLAN ID | VLAN Priority | IP de origem | IP de destino | ToS | Protocolo | Porta de origem | Porta de destino | Ação |
| - | 1 | - | - | - | - | - | - | - | - | - | - | - | Output:3 |
| - | 2 | - | - | - | - | - | - | - | - | - | - | - | drop |
| - | 3 | - | - | - | - | - | - | - | - | - | - | - | Output:1 |
| - | 4 | - | - | - | - | - | - | - | - | - | - | - | drop |

# Exercício 3

```
19      def _handle_PacketIn (self, event):
20          packet = event.parsed
21          if event.port == 1:
22              msg = of.ofp_flow_mod()
23              msg.match.in_port = event.port
24              msg.actions.append(of.ofp_action_output(port = 3))
25
26              event.connection.send(msg)
27              log.debug("FlowMod in_port=1 action=output:3")
28          elif event.port == 3:
29              msg = of.ofp_flow_mod()
30              msg.match.in_port = event.port
31              msg.actions.append(of.ofp_action_output(port = 1))
32              event.connection.send(msg)
33              log.debug("FlowMod in_port=3 action=output:1")
34          else:
35              # Drop de todos os pacotes
36              msg = of.ofp_flow_mod()
37              msg.match.in_port = event.port
38              msg.match.dl_src = packet.src
39              msg.match.dl_dst = packet.dst
40              event.connection.send(msg)
41              log.debug("Drop packet sw=%s in_port=%s src=%s dst=%s" \
42                    % (event.dpid, event.port, packet.src, packet.dst))
```

# Exercício 3

- Desafio 1: como tratar a perda de pacotes do slowpath?
- Desafio 2: como encaminhar em modo flood? (hub)

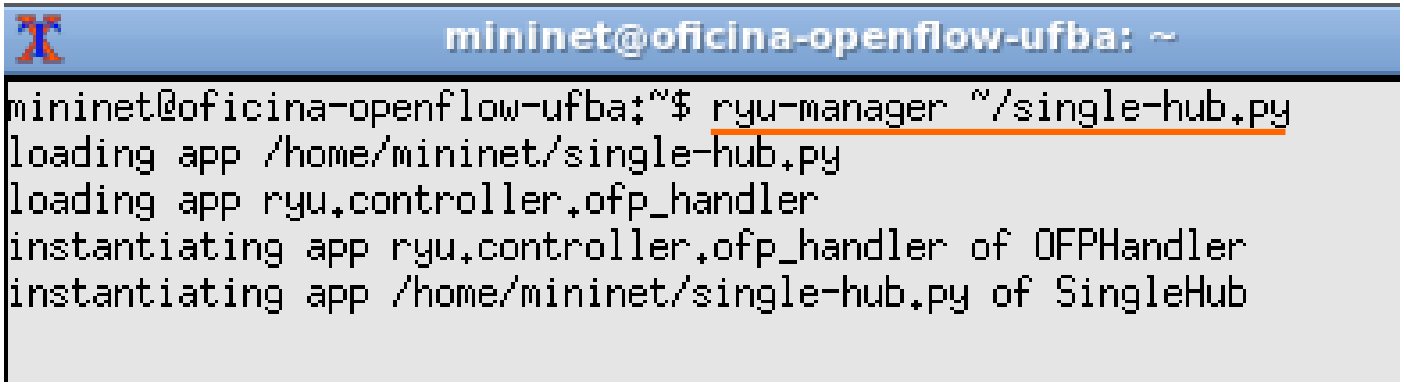
# Ryu

- Instalação de dependências:
  - sudo apt-get install -y libxslt1-dev  libffi-dev msgpack-python python-setuptools python-nose python-pip python-dev
  - sudo pip install ipaddr networkx bitarray netaddr oslo.config routes webob paramiko mock xml_compare pyflakes pylint debtcollector oslo.i18n rfc3986 greenlet tinyrpc ovs 'eventlet>=0.18.2,!=0.18.3,!=0.20.1,<0.21.0'
- Instalação do Ryu
  - git clone git://github.com/osrg/ryu.git
  - cd ryu; sudo python ./setup.py install

# Ryu Exemplo 4

```
1  # ~/single-hub.py
2  from ryu.base import app_manager
3  from ryu.controller import ofp_event
4  from ryu.controller.handler import MAIN_DISPATCHER
5  from ryu.controller.handler import set_ev_cls
6  from ryu.ofproto import ofproto_v1_0
7
8  class SingleHub(app_manager.RyuApp):
9      OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
10
11     def __init__(self, *args, **kwargs):
12         super(SingleHub, self).__init__(*args, **kwargs)
13
14     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
15     def packet_in_handler(self, ev):
16         msg = ev.msg
17         dp = msg.datapath
18         ofp = dp.ofproto
19         ofp_parser = dp.ofproto_parser
20
21         actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
22         out = ofp_parser.OFPPacketOut(
23             datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
24             actions=actions)
25         dp.send_msg(out)
```

# Ryu Exemplo 4



# Bibliotecas e Apps de apoio

- NetworkX
  - Estrutura de dados para armazenamento de informações da topologia de rede
- LLDP
  - Aplicação de apoio para descoberta de enlaces
- REST
  - Aplicação de apoio para Northbound API
- Topology Viewer
  - Aplicação de apoio para visualização da rede
- ...

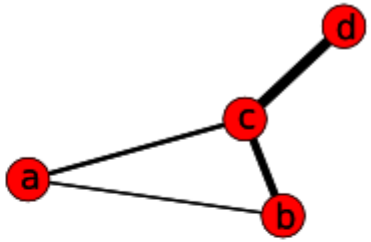
# NetworkX

- Estrutura de dados para representar muitos tipos de redes, ou seja, grafos
- Os nós podem ser qualquer objeto Python (hashable) e as arestas podem conter dados arbitrários
- Fácil instalar em muitas plataformas
- Muita documentação disponível

# NetworkX – Exemplo

- Usar algoritmo de Dijkstra para busca do melhor caminho em grafos com e sem pesos

```
>>> g = nx.Graph()
>>> g.add_edge('a','b',weight=0.1)
>>> g.add_edge('b','c',weight=1.5)
>>> g.add_edge('a','c',weight=1.0)
>>> g.add_edge('c','d',weight=2.2)
>>> print nx.shortest_path(g,'b','d')
 ['b', 'c', 'd']
>>> print nx.shortest_path(g,'b','d',weighted=True)
 ['b', 'a', 'c', 'd']
```



# NetworkX – Iniciando

- Importar a biblioteca:
  - import networkx as nx
- Você pode criar diferentes tipos de grafo:
  - g = nx.Graph()
  - d = nx.DiGraph()
- Adicionar os nós (um ou muitos):
  - g.add_node(1)
  - g.add_nodes_from([2 ,3])
  - g.add_node(4, time=’5pm’)
- Adicionar arestas:
  - g.add_edge(1,2)
  - g.add_edges_from([(1 ,2) ,(1 ,3)])

# NetworkX – Iniciando

- Iteração com vértices e aretas:

```
>>> g.add_edge(1,2)
>>> for node in g.nodes():
     print node, g.degree(node)
 1, 1
 2, 1

>>> g.add_edge(1,3,weight=2.5)
>>> g[1][2]['weight'] = 1.5
>>> for n1,n2,attr in g.edges(data=True):
     print n1,n2,attr['weight']
1, 2, 1.5
1, 3, 2.5
```

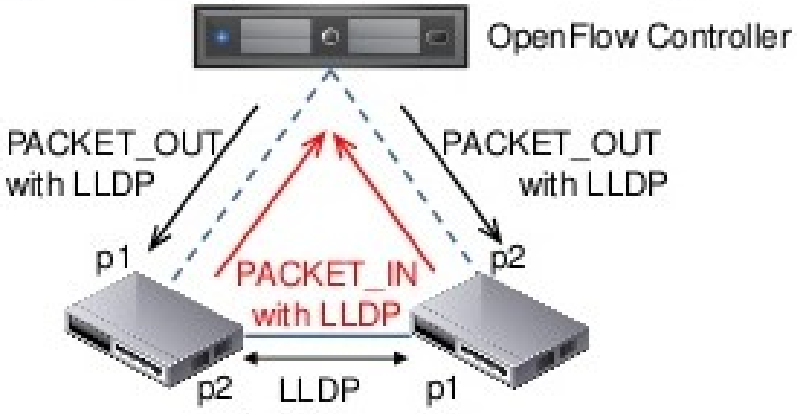
# LLDP

- Um desafio na construção de redes OF complexas ou dinâmicas é a descoberta e checagem da saúde dos links
  - Não confundir a topologia do mininet com visão do controlador
- Mecanismos ativos vs passivos
- LLDP – Link Layer Discovery Protocol
  - Padrão IEEE 802.1AB
  - Ethertype 0x88CC
- Pode incluir diversas informações do nó:
  - Nome, descrição da porta, VLAN, IP, MAC, features L2/L3, etc

# LLDP

- Mecanismo:
  - 1) regra no switch OF para enviar tráfego desconhecido ou LLDP ao controlador
  - 2) controlador envia PacketOut com LLDP em todas as portas do switch OF
  - 3) controlador captura os PacketIn equivalentes e constrói a topologia

| IDX | SRC | DST | SRC PORT | DST PORT |
|---|---|---|---|---|
| 153 | sw. A | sw. B | p2 | p1 |
| ... | ... | ... | ... | ... |
| 357 | sw. B | sw. A | P1 | p2 |



# Exercício

- Criar uma  topologia linear,3 no Mininet
- Desenvolver uma aplicação multi-hub.py
  - Topologia da rede em Grafo/NetworkX
  - LLDP para descoberta de links
  - Encaminhamento em modo “flooding”
- Testar interface REST do Ryu