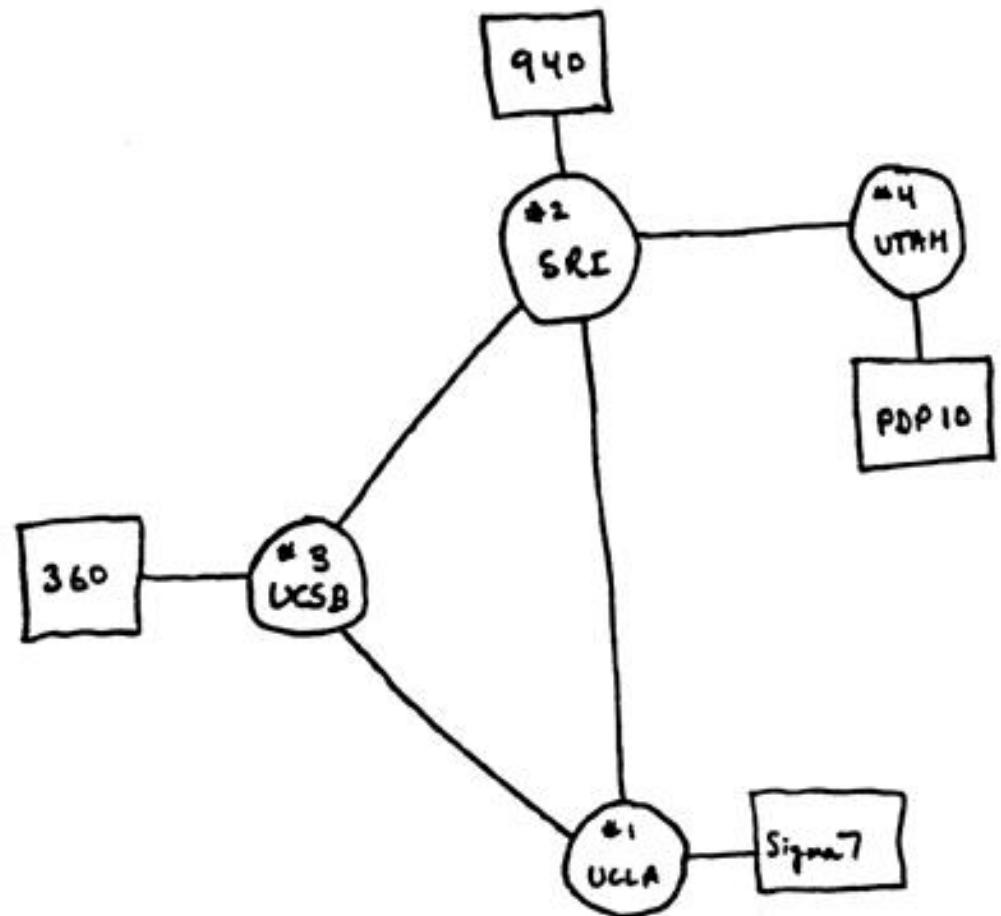
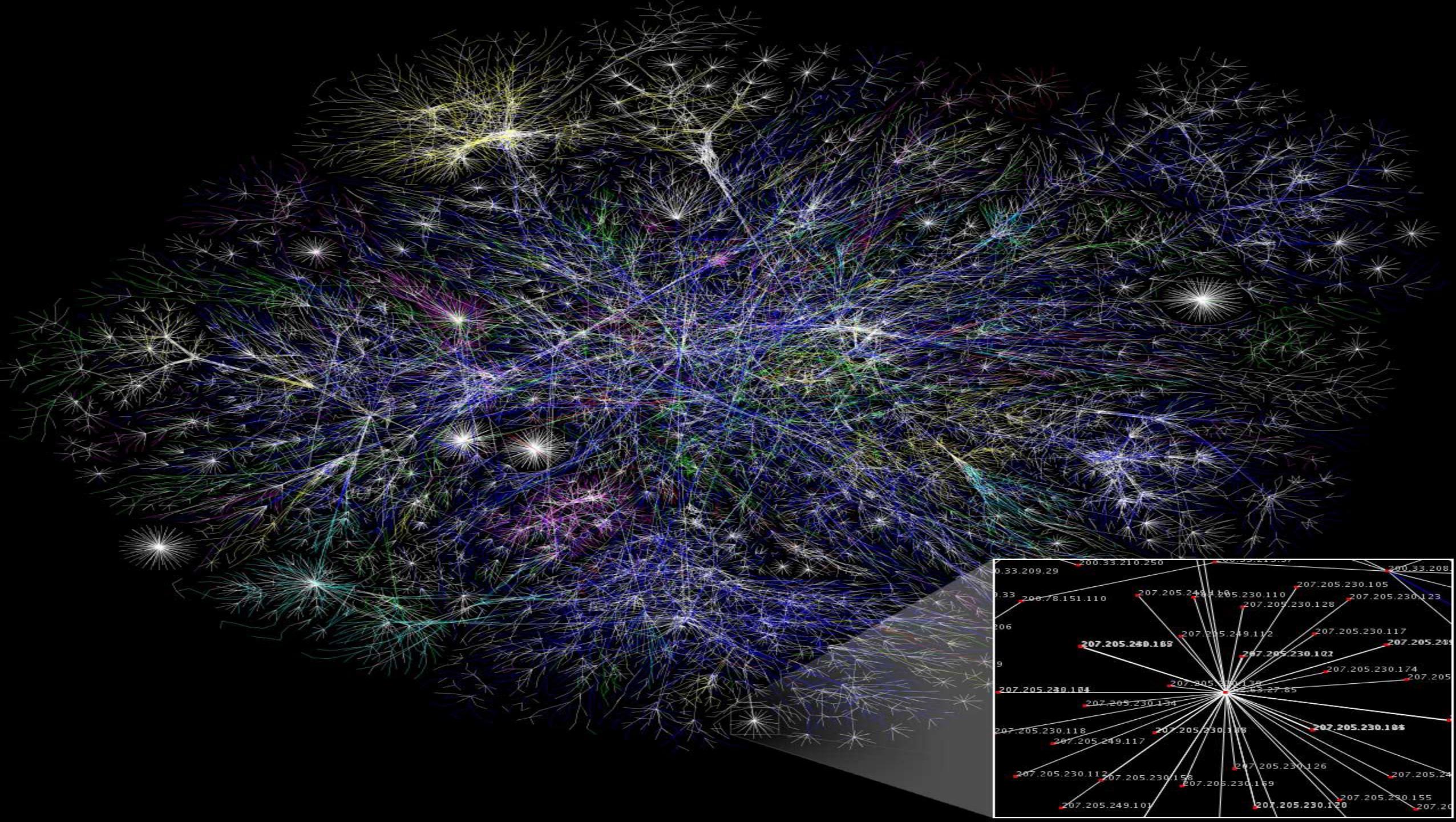


# SDN





- Firewall
  - Switch
  - Router
  - Protocolo
- S
- Serviços

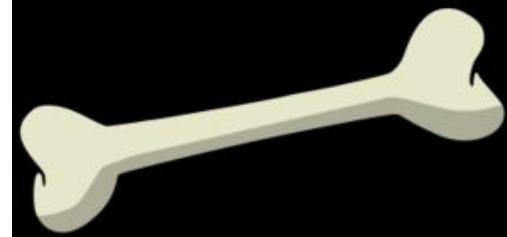


A Internet é sem dúvida, um grande sucesso...mas esse sucesso se tornou um grande problema para sua evolução!

“A Internet está ossificada”

Muito investimento depende do seu funcionamento  
Economia de escala inviabiliza novas tecnologias

Diversas soluções já foram propostas, mas tiveram alcance limitado.



- Uma nova forma de olhar para o problema: dotar novamente o núcleo da rede de recursos de programação.
  - Sane (2006), Ethane (2007): controle centralizado de regras de firewall em uma rede corporativa
  - OpenFlow (2008): extensão do princípio como uma forma de se programar os elementos de comutação da rede

# SDN

---

Software Defined Networking

The logo consists of a blue sphere with a white, curved, ribbon-like pattern wrapped around it, resembling a stylized 'Q'. To the right of the sphere, the word "OpenFlow" is written in a large, bold, sans-serif font. The first two words, "Open" and "Flow", are in blue, while "Flow" has a grey shadow effect.

OpenFlow



- SDN permite a inovação por conta própria;
- A mudança de paradigma;
- A era do Sistema Operacional de Rede;
- A inovação permite uma proposta de valor.

[Data Center](#)[Disaster Recovery](#) | [High Performance Computing](#) | [Infrastructure Management](#)[Sustainable IT](#) | [Virtualization](#)[Home](#) > [Data Center](#) > [Virtualization](#)[News](#)

## VMware to acquire OpenFlow pioneer Nicira for \$1.26 billion

Nicira helped define the emerging field of software-defined networking

By **Joab Jackson**

July 23, 2012 05:26 PM ET 

---

IDG News Service - Continuing its push to virtualize all aspects of the data center, VMware is acquiring software-defined networking firm Nicira for \$1.26 billion, the companies announced Monday.

"I believe we have the same opportunity to do for networking what we've already done for servers and many other parts of the data center," Steve Herrod, VMware's CTO, wrote in a [blog post](#) about the deal.

Open Networking Foundation, 2011 (Stanford + UC Berkeley)  
Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and  
Yahoo!

## Open Networking Summit

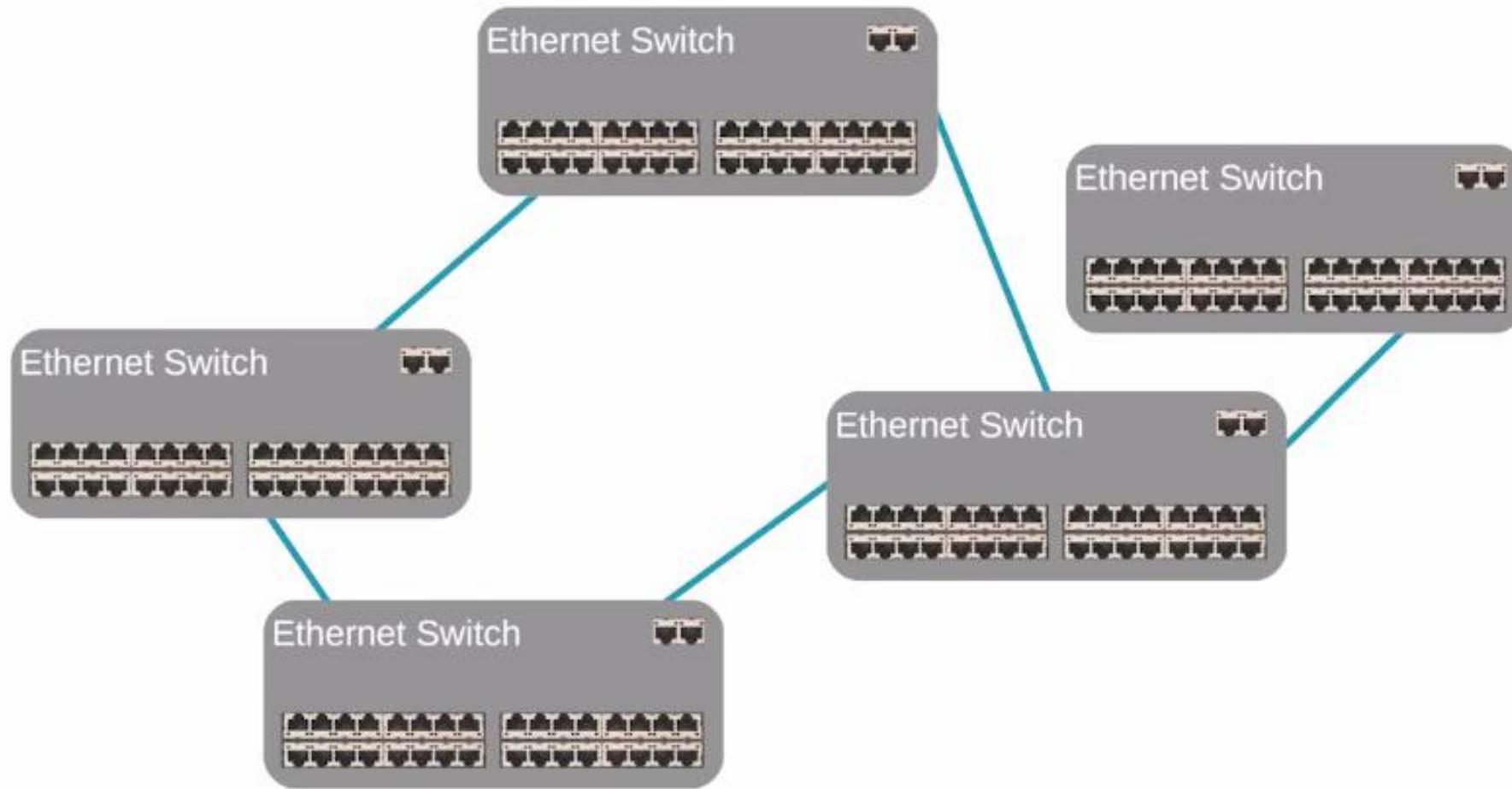
Open Networking Research Center, 2012  
CableLabs, Cisco, Ericsson, Google, Hewlett-Packard, Huawei,  
Intel, Juniper, NEC, NTT DOCOMO, Texas Instruments, Vmware.



# O que vamos aprender.

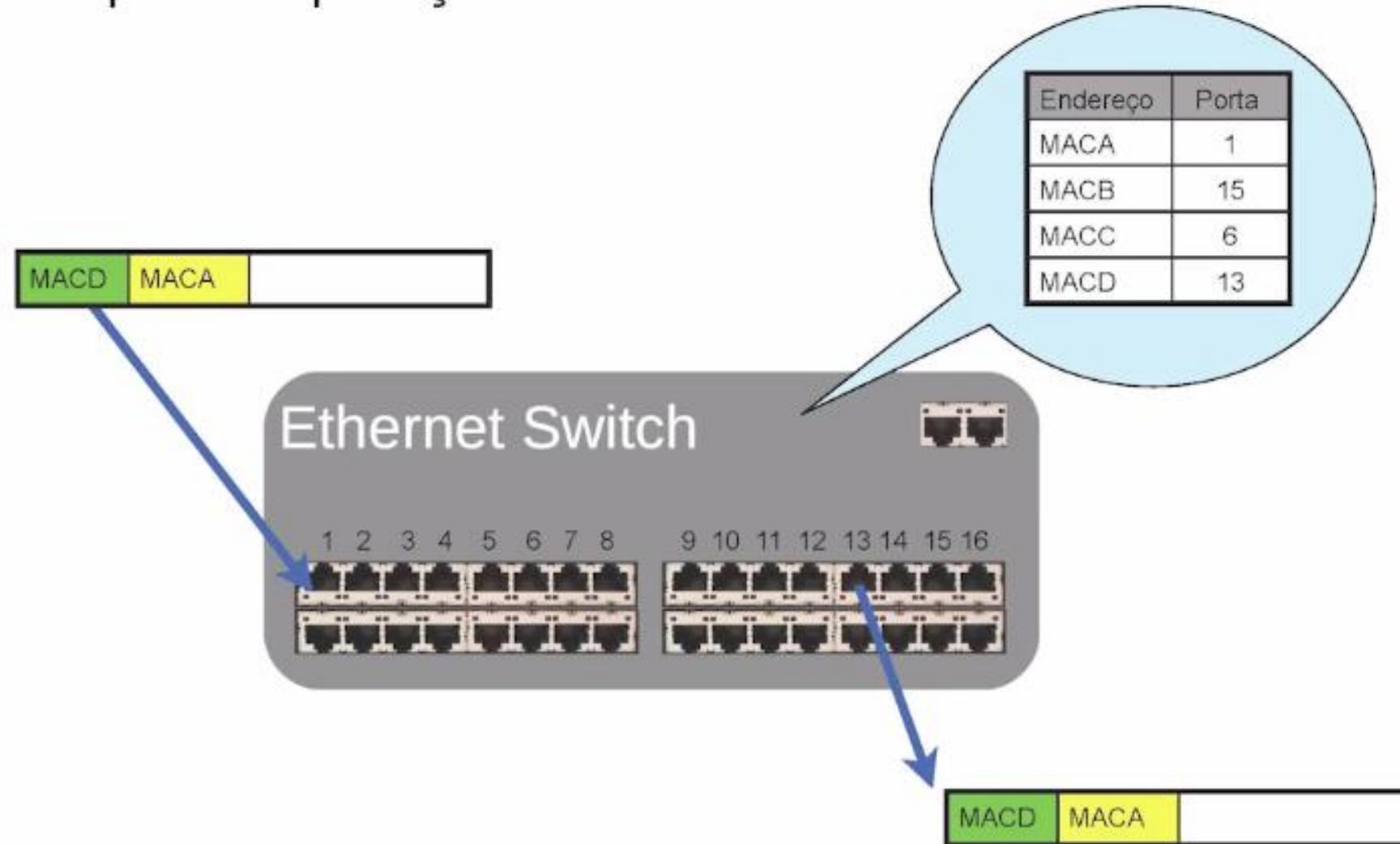
- Apresentar os elementos principais de SDNs;
- Exemplificar diversos modelos de programação possíveis;
- Introduzir um controlador em mais detalhes;
- Listar alguns exemplos de aplicação de SDNs;
- Discutir desafios de pesquisa.

# Rede Local



# Rede Local (Tabela Encaminhamento)

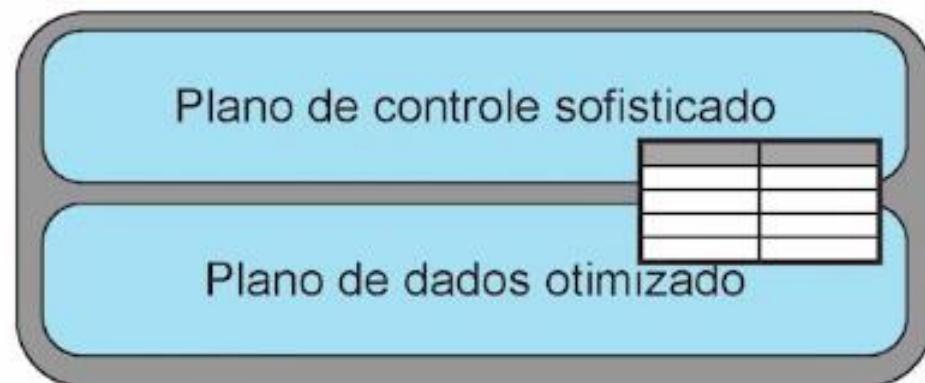
Exemplo de operação de um *switch* Ethernet



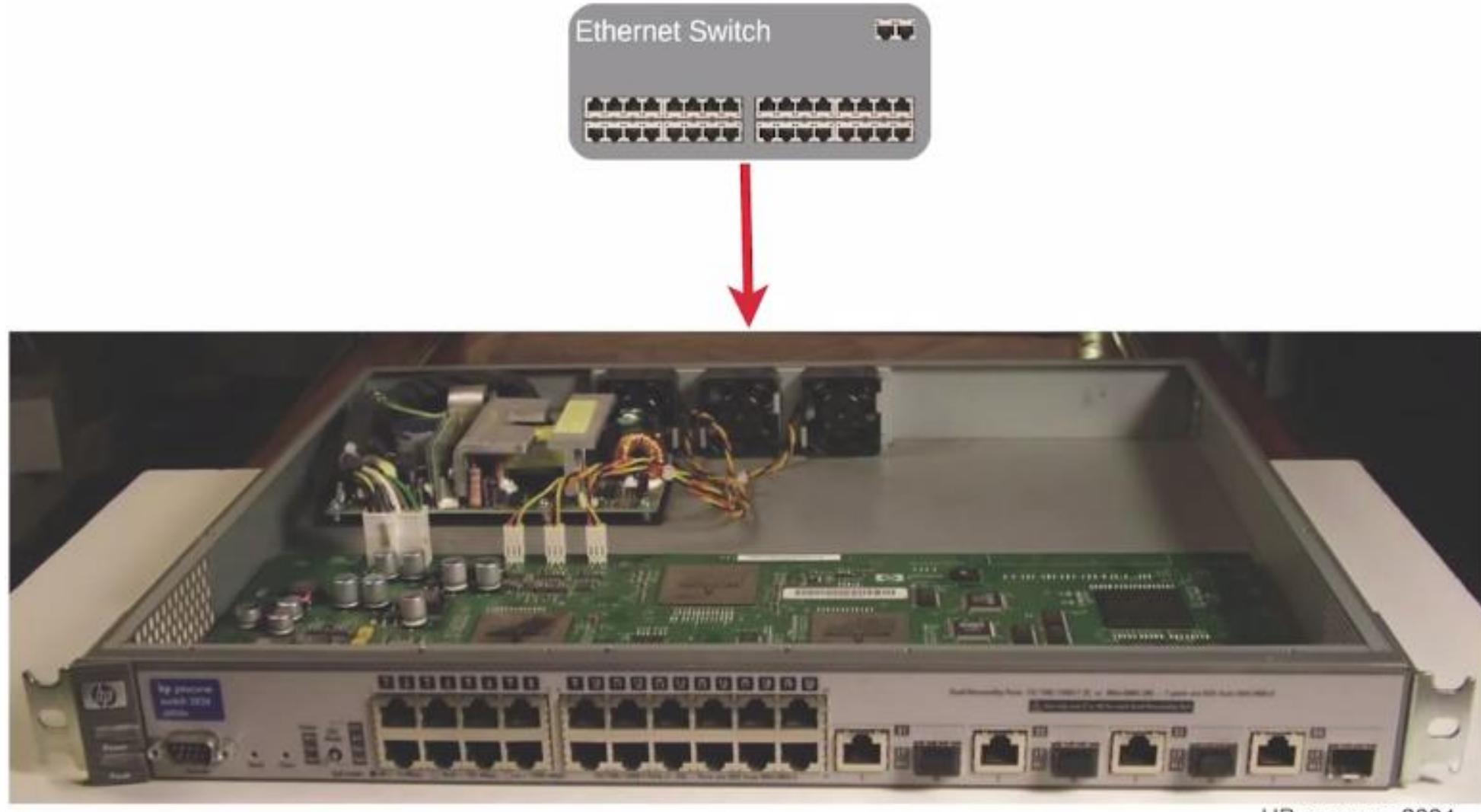
# Organização do Switch/Router

## Organização de um switch/roteador

- Plano de dados: o caminho de cada pacote
  - pacote chega
  - tabela de encaminhamento é consultada – e atualizada
  - pacote sai
- Plano de controle: tudo que acontece “por trás”
  - atualizações da tabela de encaminhamento
  - manutenção de estatísticas

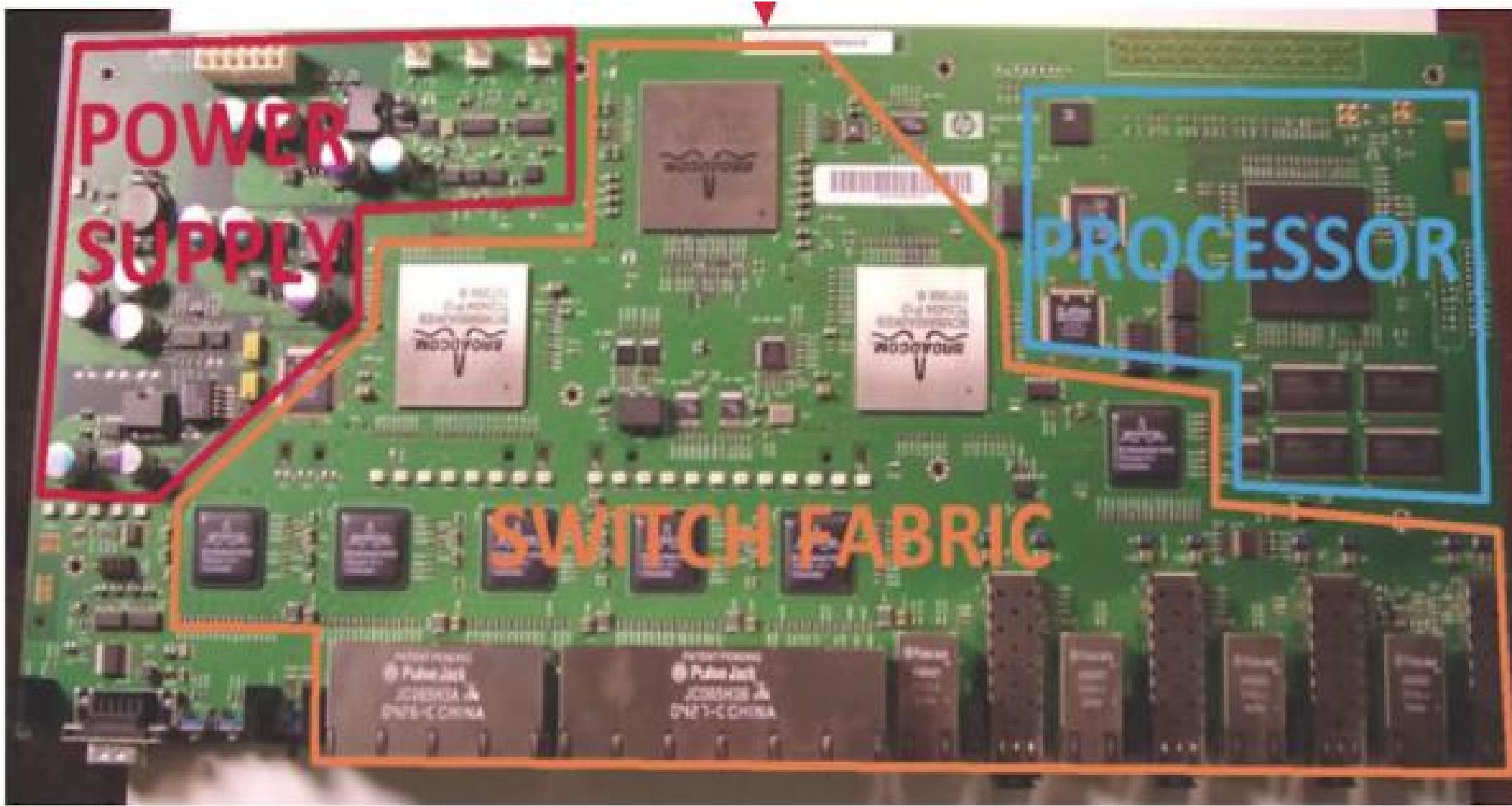


# Switch por dentro

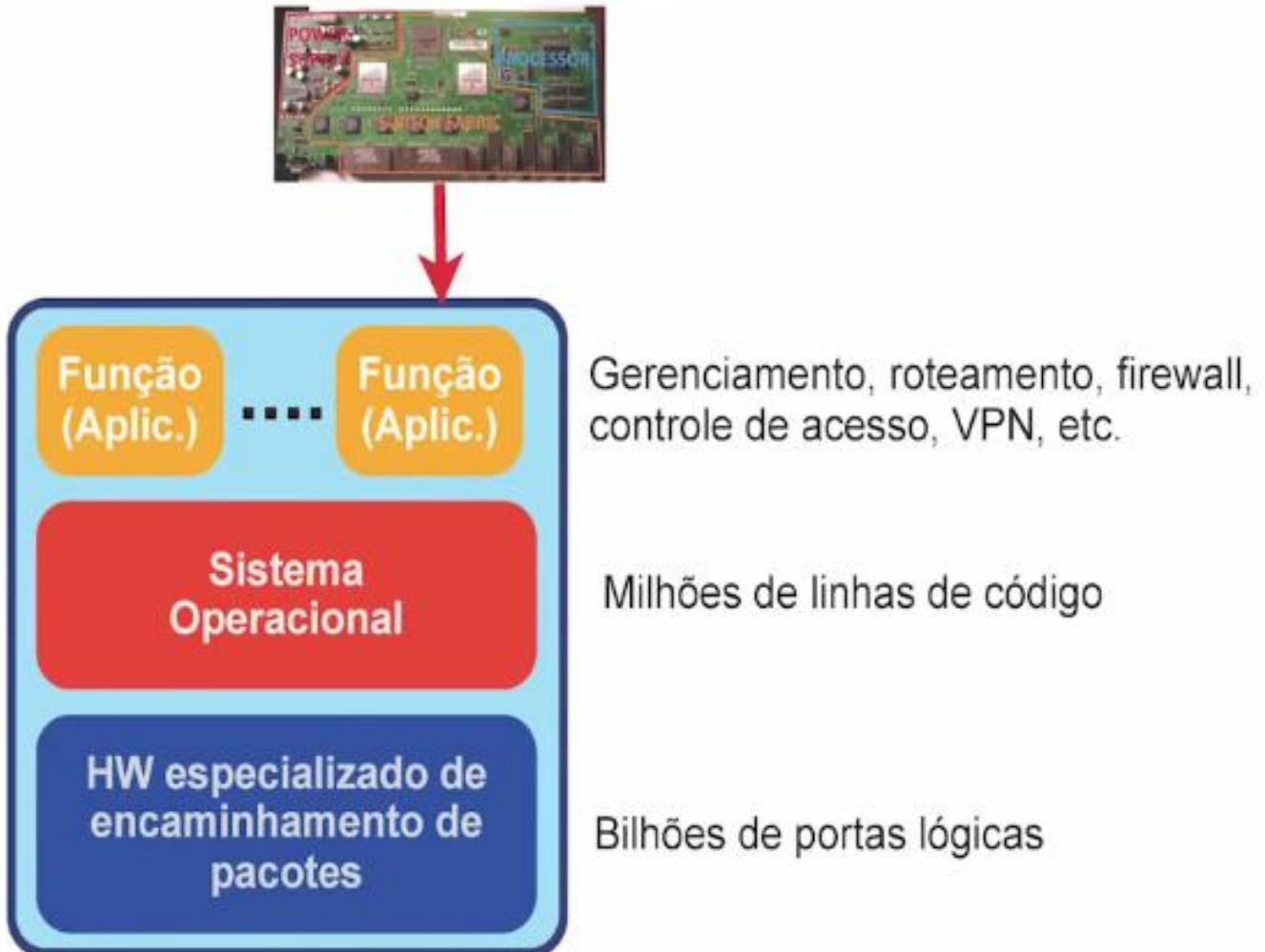


HP procurve 2824

# Switch por dentro



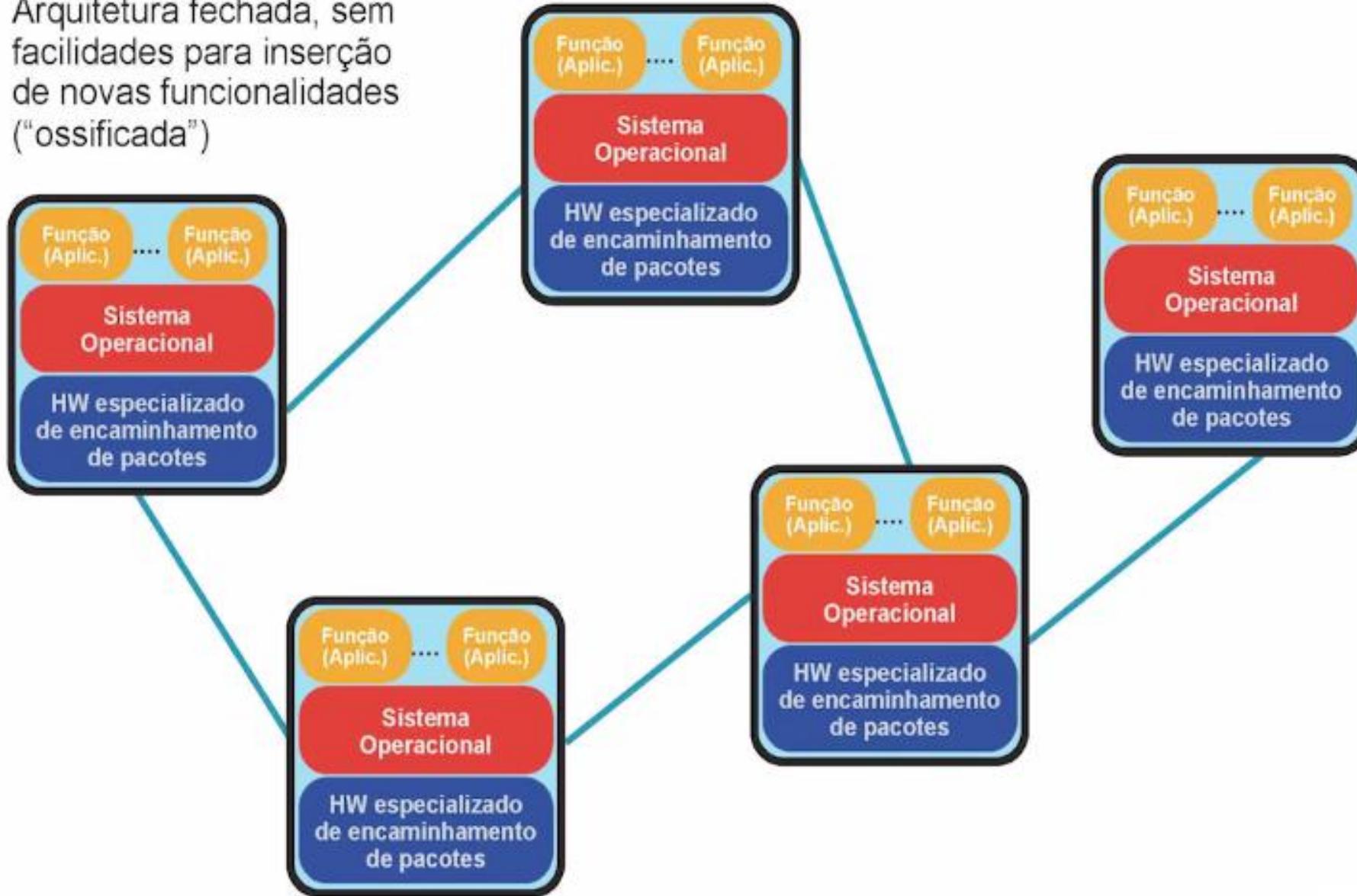
# Switch por dentro



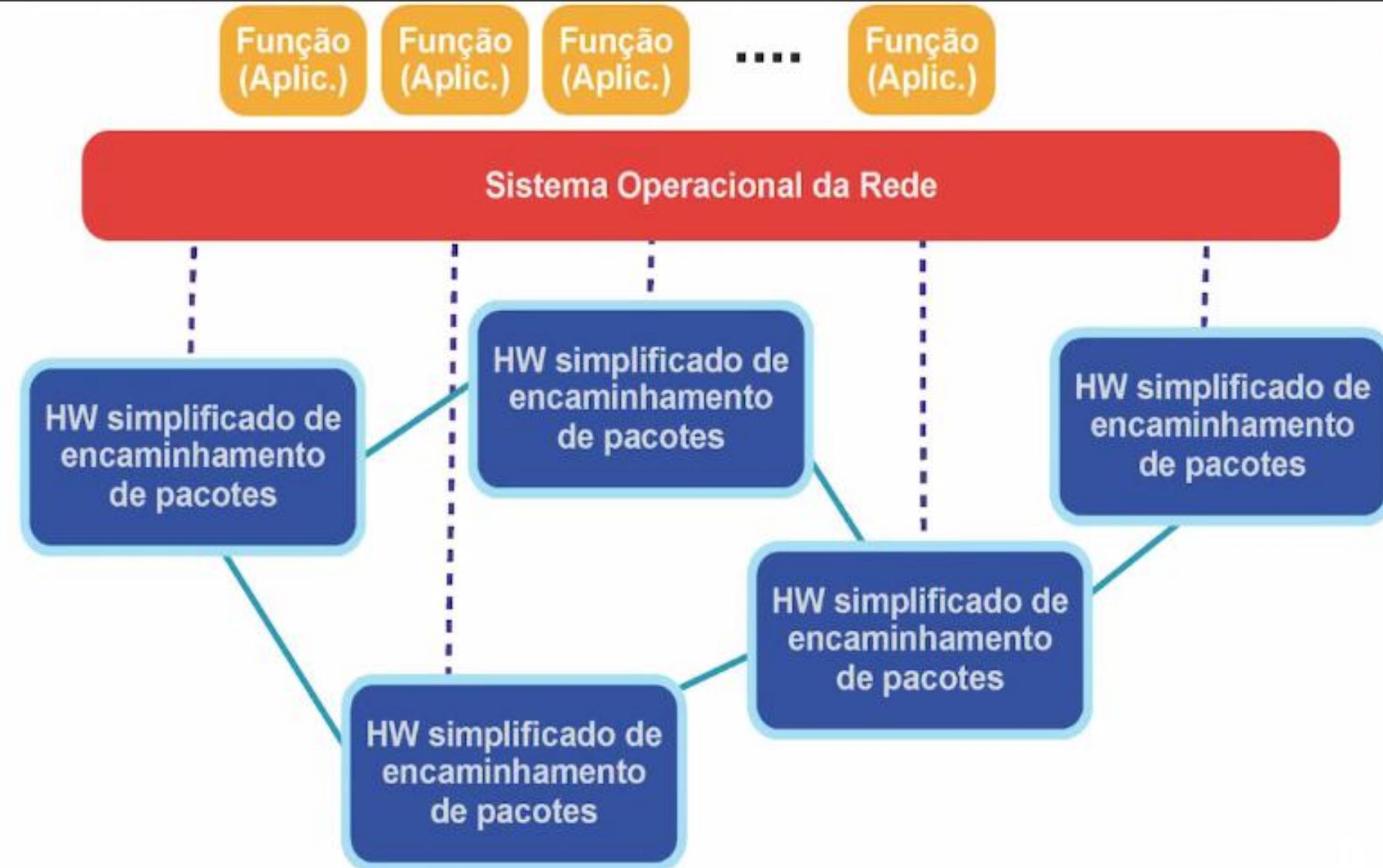
- Muitas funções complexas embutidas na infraestrutura

# Componentes da Rede Atual

Arquitetura fechada, sem facilidades para inserção de novas funcionalidades ("ossificada")



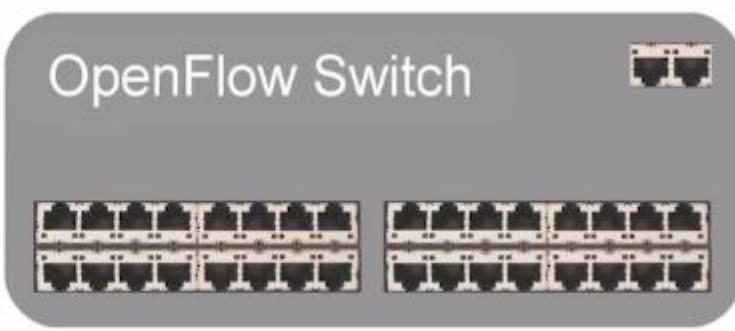
# Proposta SDN



# Organização Comutador Openflow

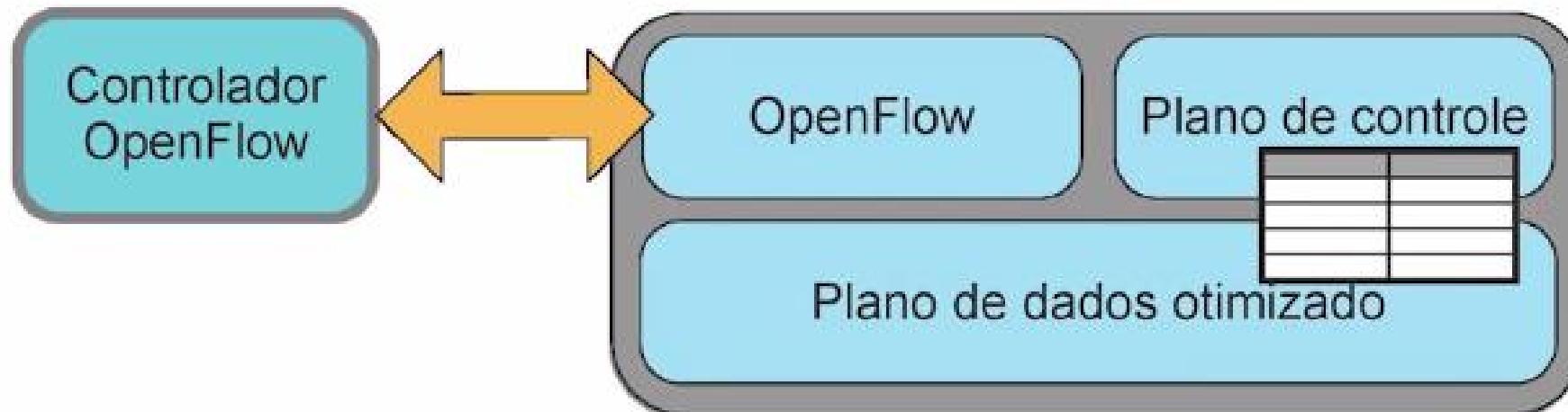
**HW simplificado de  
encaminhamento de  
pacotes**

=

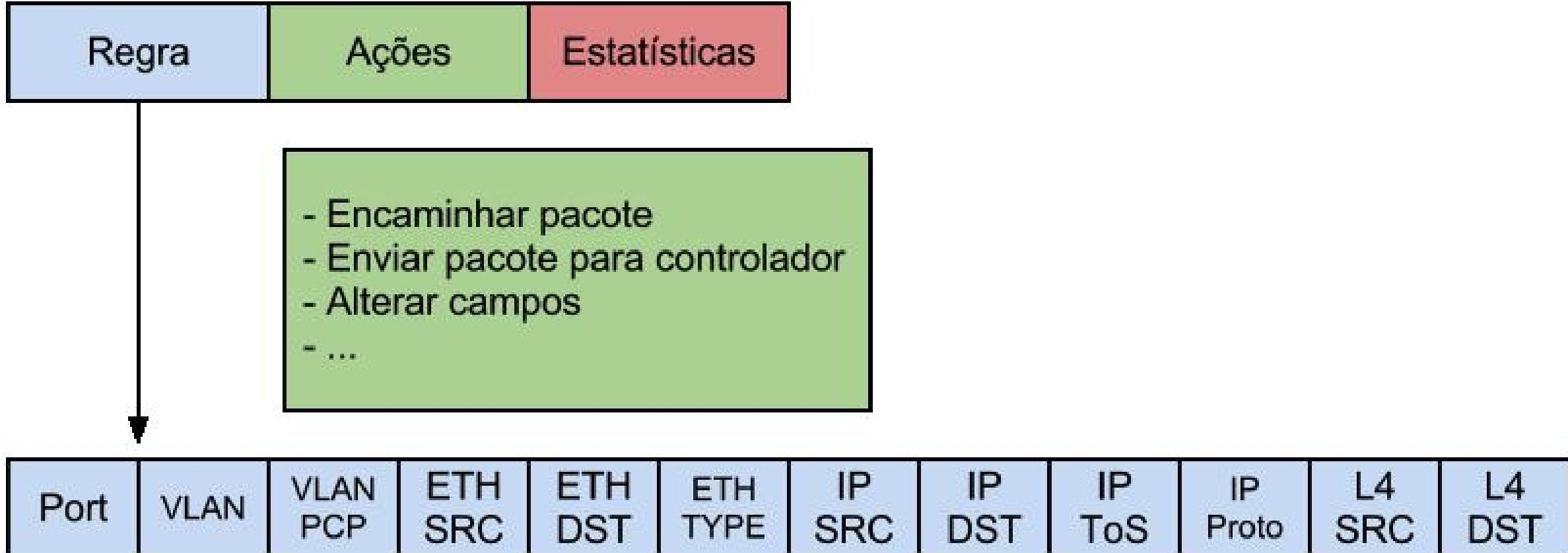


# Organização Comutador Openflow

- Plano de dados: o caminho de cada pacote
  - pacote chega
  - tabela de encaminhamento é consultada – e atualizada
  - tabela dita como o pacote deve ser tratado
- Plano de controle: só acesso à tabela de encaminhamento
- OpenFlow: API de acesso à tabela



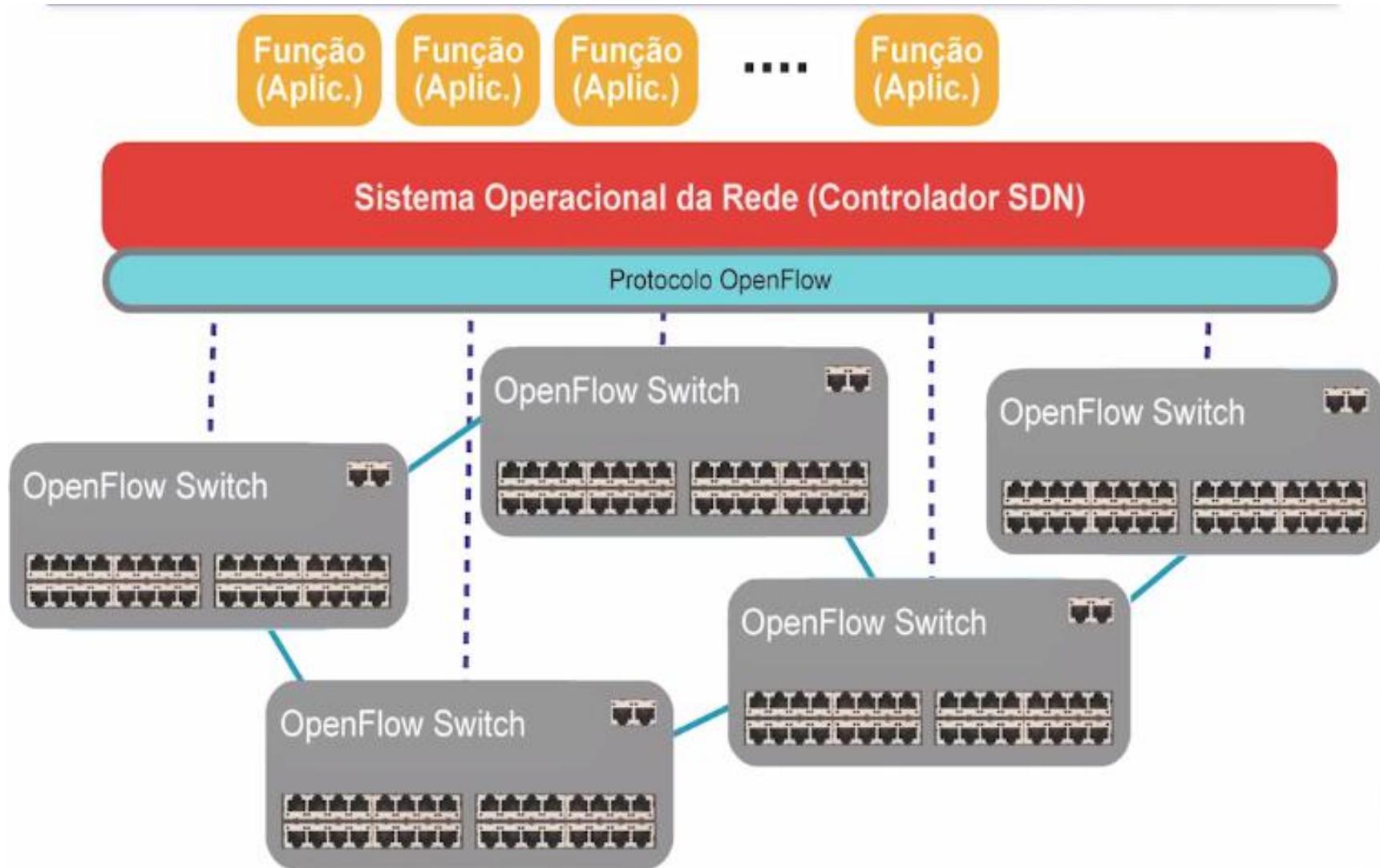
# OpenFlow: Entradas da Tabela de Encaminhamento.



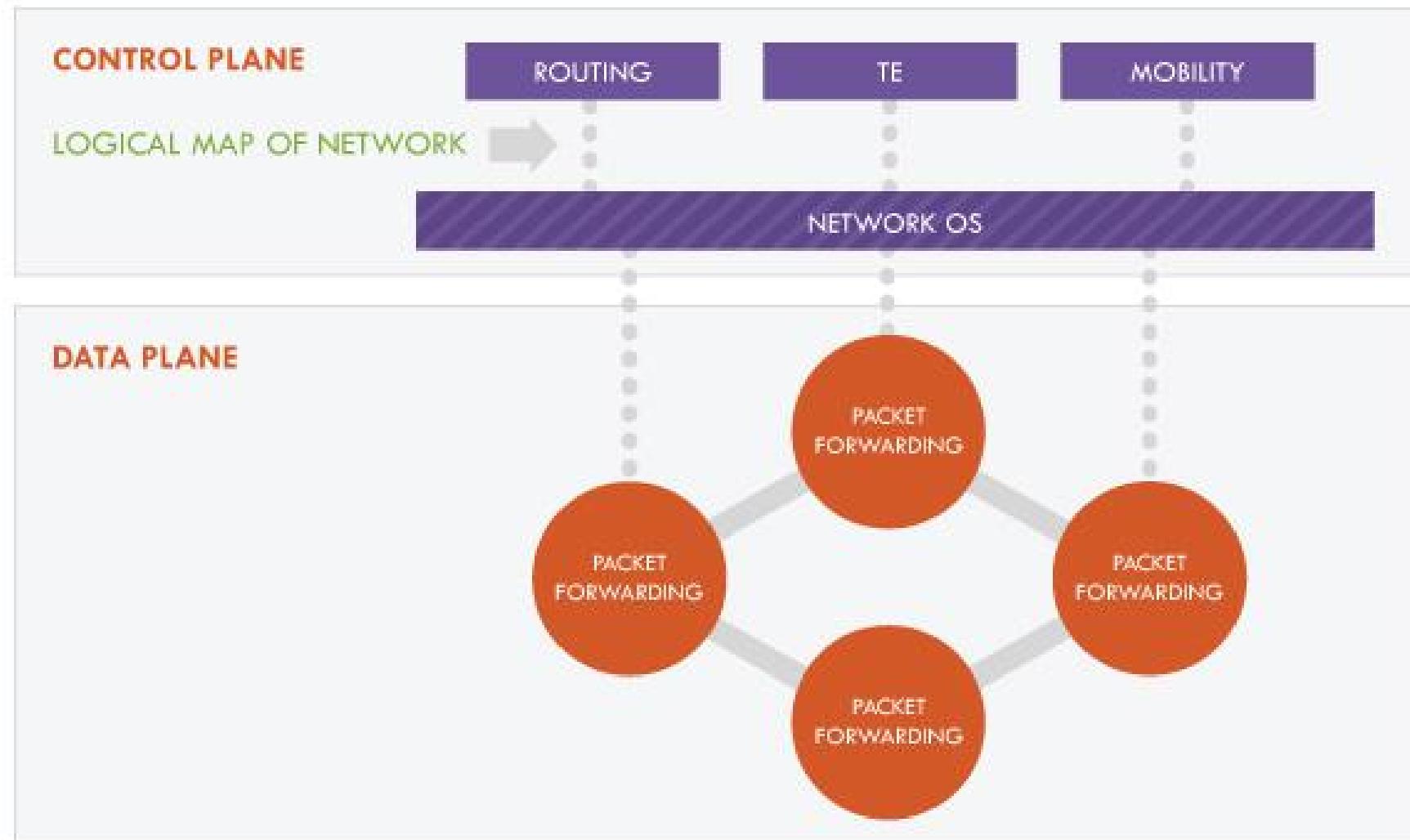
# OpenFlow: Tabela de Encaminhamento.

PORT	VLAN	MAC Src	MAC Dst	Eth Type	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Ação
*	*	*	00:1f...	*	*	*	*	*	*	Porta 6
p3	v11	00:20...	00:1f...	0800	1.2.3.4	5.6.7.8	4	1765	80	Porta 6
*	*	*	*	*	*	*	*	*	22	Drop
*	*	*	*	*	*	5.6.7.8	*	*	*	Porta 6
*	v11	00:1f...	*	*	*	*	*	*		Portas 6,7,8

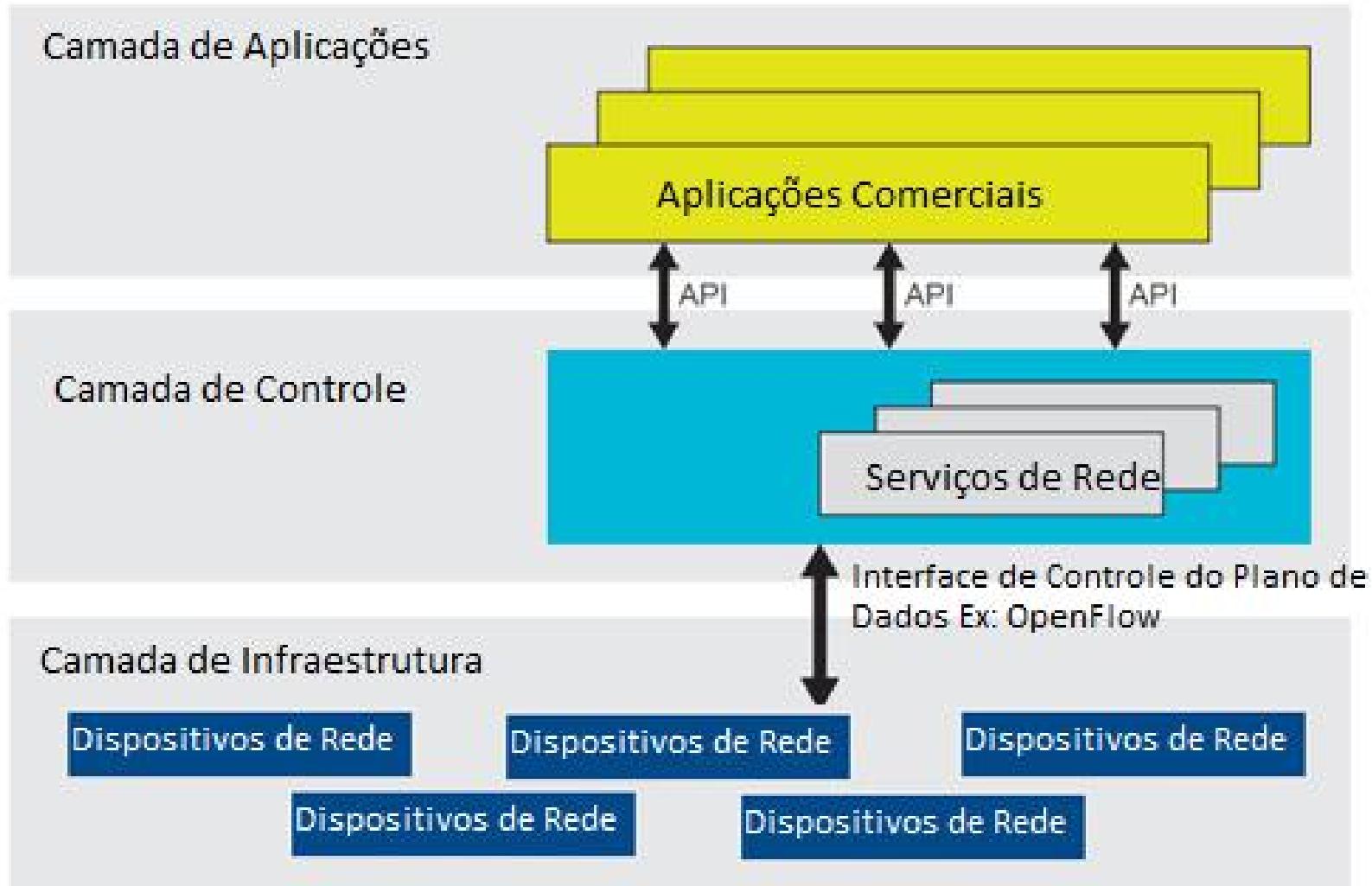
# Proposta SDN Atual



# Componentes de uma SDN



# Componentes de uma SDN



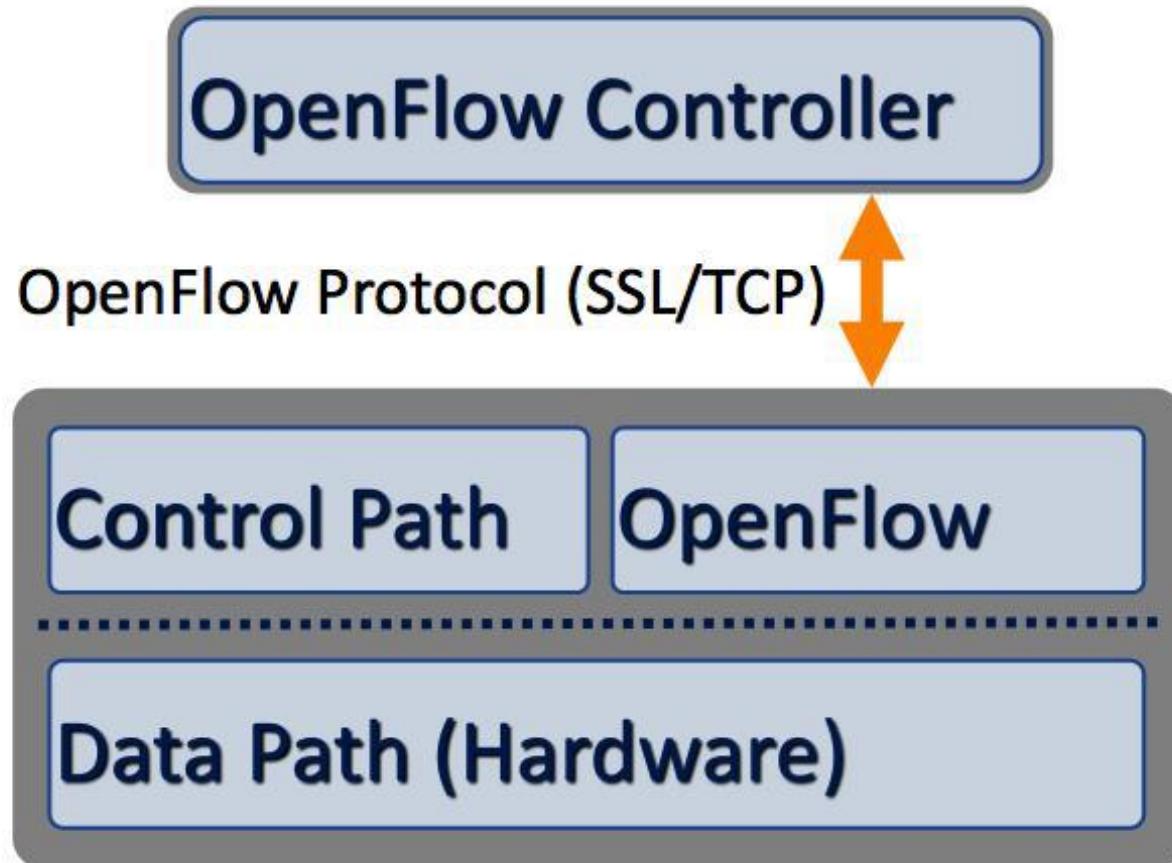
# Componentes de uma SDN

- ❑ Elementos de comutação
  - Plano de dados programável
- ❑ Divisor de recursos
  - Divisão do espaço de endereçamento
- ❑ Controlador (sistema operacional de rede)
  - Responsável pela visão global da rede
- ❑ Aplicações
  - Implementação da função de controle

# Elemento de comutação programáveis

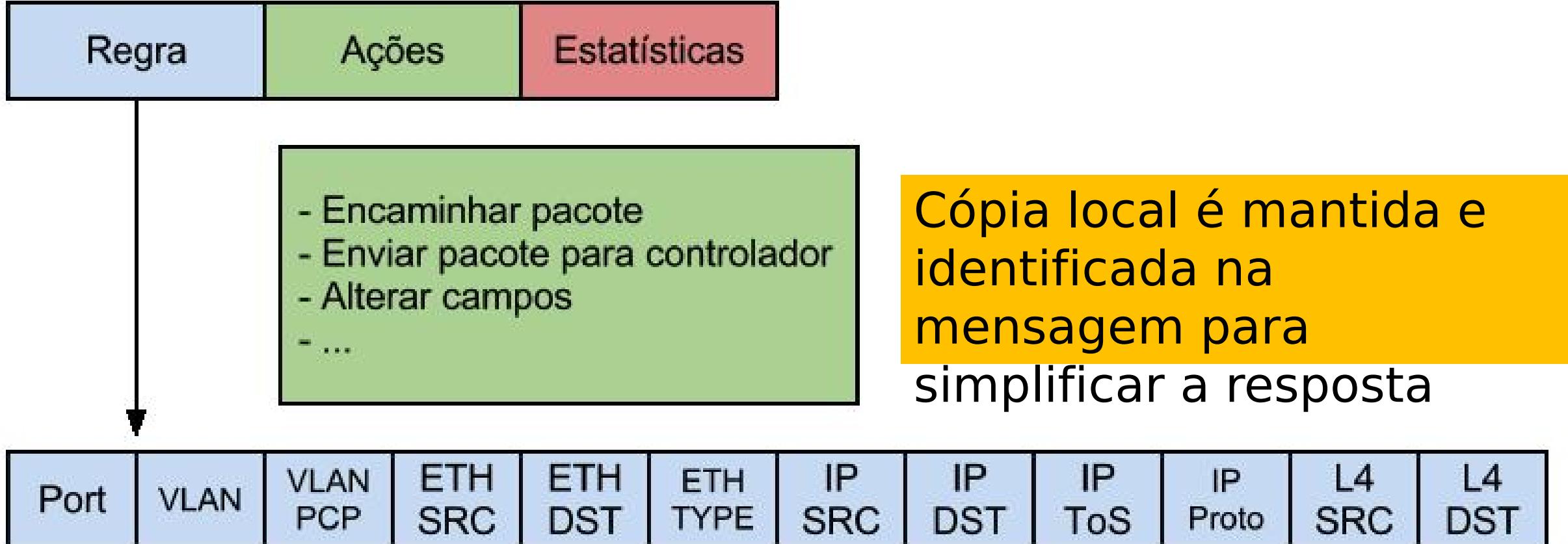
- Chaveamento de pacotes eficiente (HW).
- Interface de programação para preenchimento da tabela de encaminhamento
- OpenFlow é a interface de programação que tem ganhado mais atenção recentemente  
Mas outras opções são possíveis:  
Redes ativas (*active networks*)  
Chaveamento por labels (*a la* MPLS)

# OpenFlow



- Separação entre planos de dados e controle
- Protocolo de controle bem definido (seguro)
- Atuação através da tabela de fluxos

# OpenFlow: regras, ações e estatísticas.



# OpenFlow: Tabela de Fluxos.

# OpenFlow: implementações

- Versões: 1.4 (14/10/13);
- Switch OpenFlow em SW (aplicação);
- Implementação com NetFPGA (HW - <http://netfpga.org/>);
- Open vSwitch (kernel, eficiente);
- Open WRT (home, wireless);
- Implementações comerciais começam a surgir.

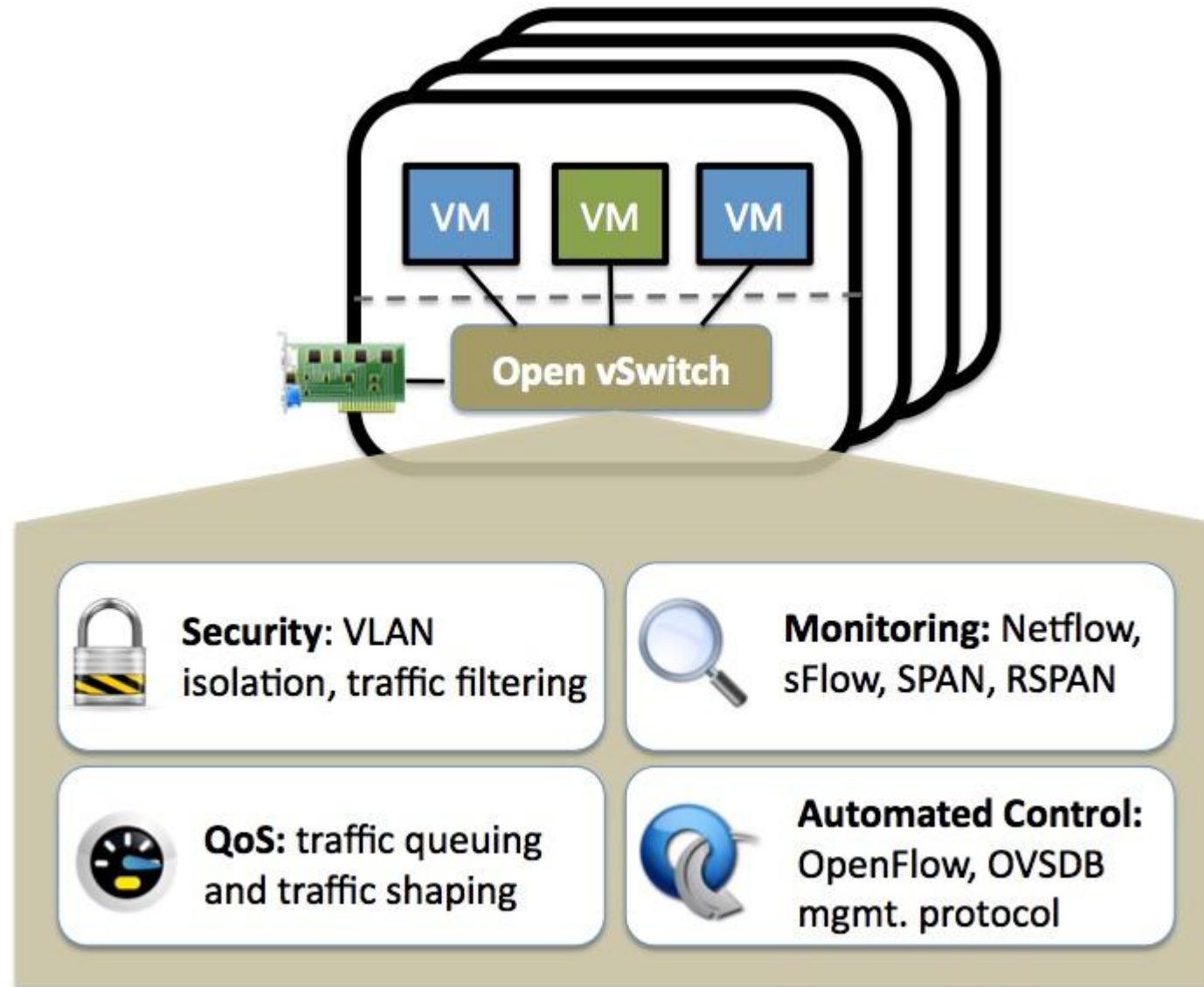
# Controladores SDN (SO da Rede)

- Comunicam-se com os elementos de comutação que constituem a Rede;
- Criam a visão única da rede (Grafos);
- Recebem dos elementos de comutação informações sobre pacotes que não casam com as regras já enviadas;
- Oferecem uma interface de programação para o desenvolvedor/adm de Rede;
- Questão: Nível de abstração e linguagem;

# Open vSwitch (<http://openvswitch.org/>)

- Desenvolvido para ambientes virtualizados;
- Alto desempenho em condições normais;
- Possível opção para caminho de migração sem alteração do hardware instalado;
- Problema de integração entre máquinas virtualizadas e máquinas físicas.

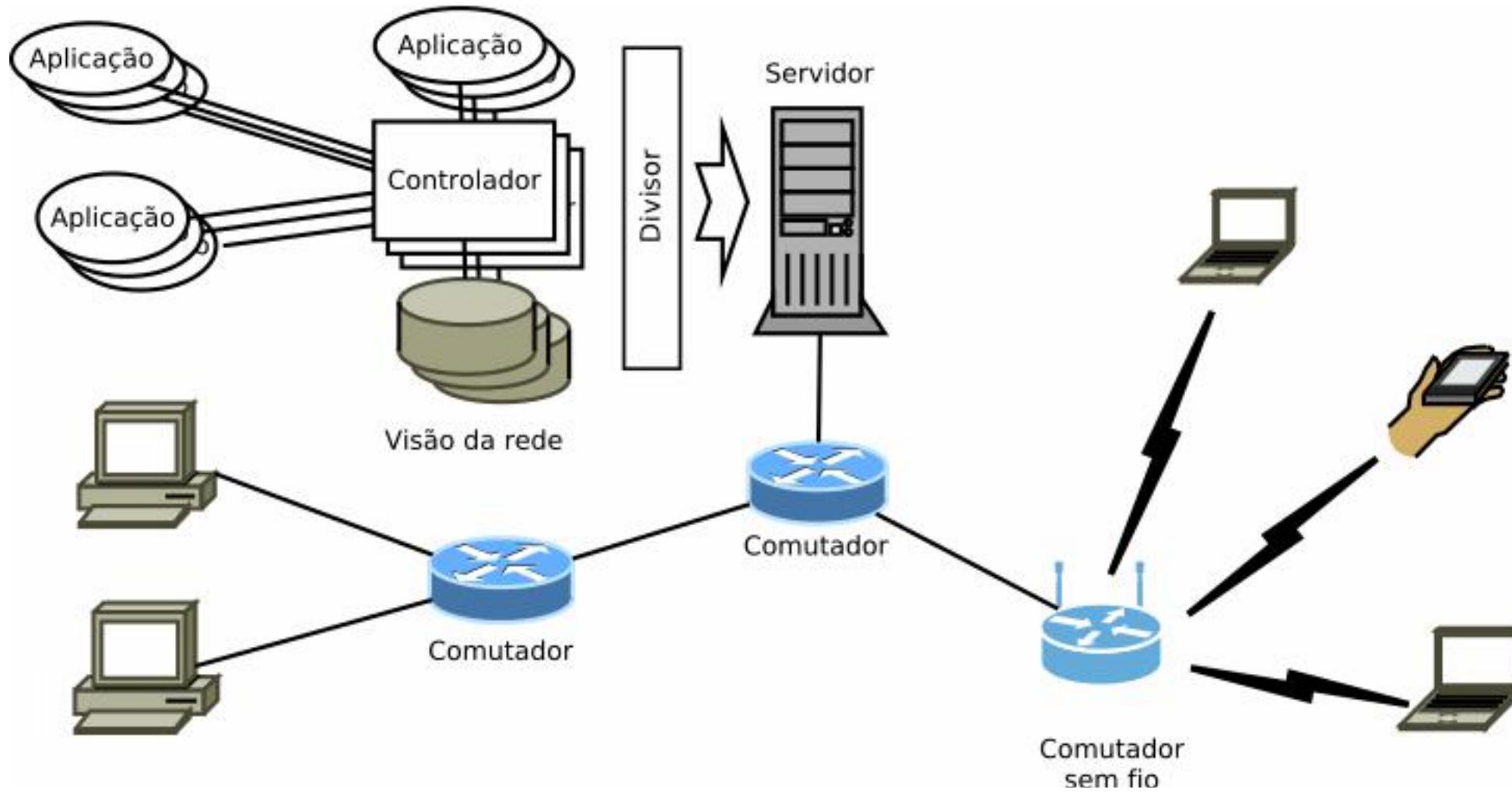
# Open vSwitch (<http://openvswitch.org/>)



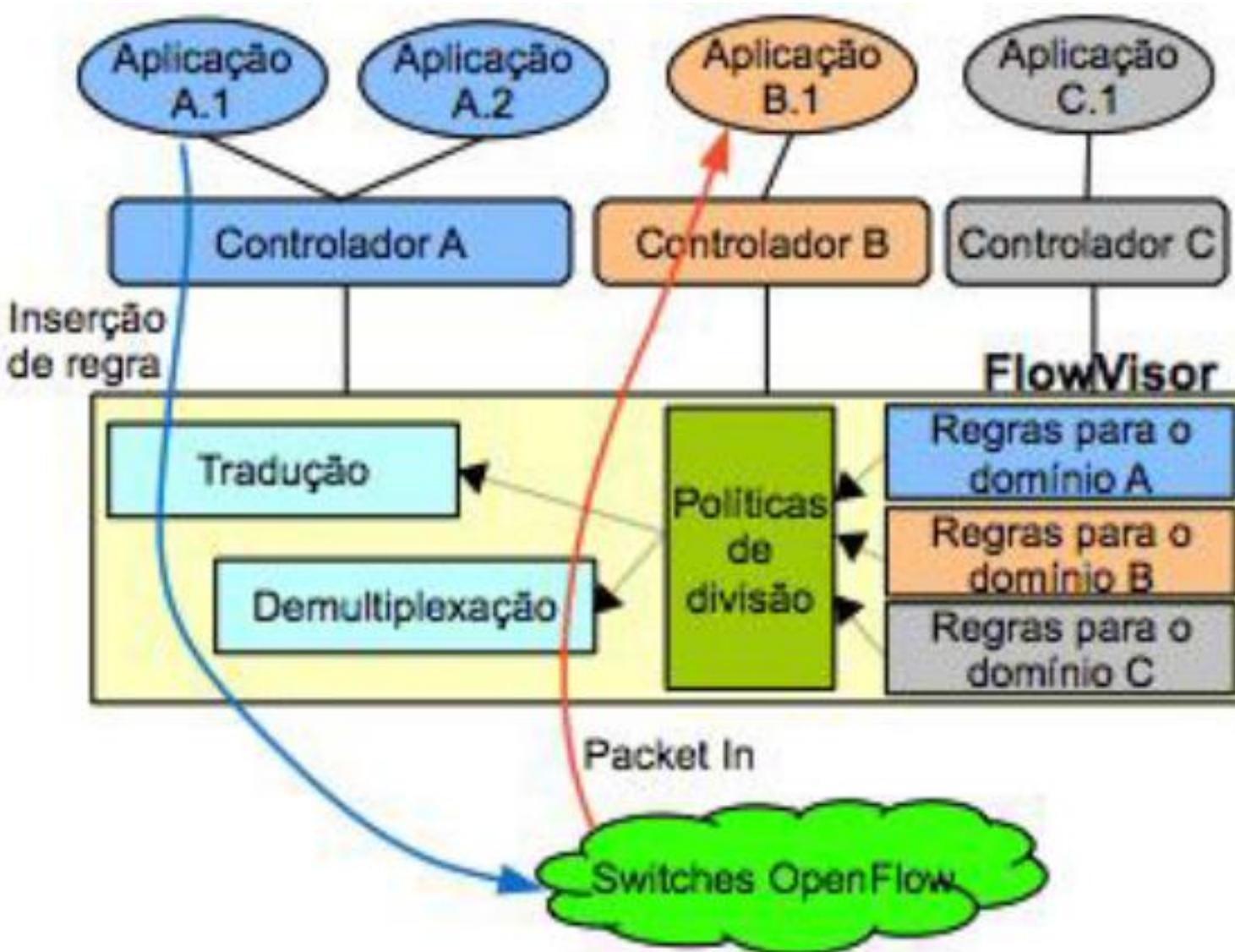
# Particionamento de recursos (FlowVisor)

- Proxy OpenFlow que permite dividir o espaço de endereçamento de pacotes
- Semelhante ao conceito de virtualização de máquinas (slices)
- Solução para o problema de compartilhamento de infra-estrutura entre projetos diferentes

# Particionamento de recursos (FlowVisor)



# Particionamento de recursos (FlowVisor)



# Controladores SDN (S.O. de rede)

- Comunicam-se com os elementos de comutação que constituem a rede
- Criam a visão única (grafo anotado) da rede
- Oferecem uma interface de programação para o desenvolvedor
- Questão: nível de abstração e linguagem

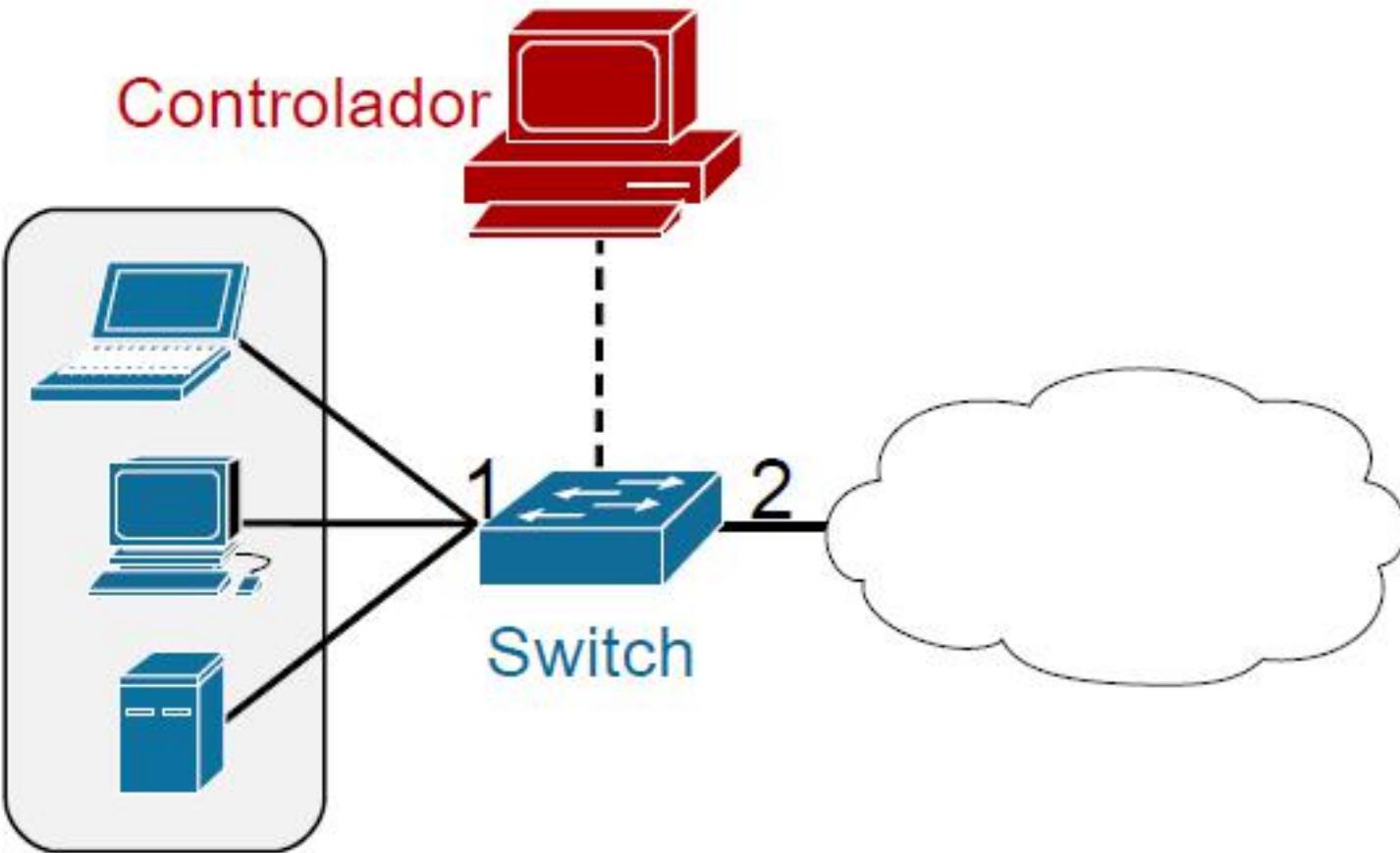
# NOX ([www.noxrepo.org](http://www.noxrepo.org))

- Primeiro controlador de SDN
- Desenvolvido pela Nicira em paralelo ao padrão OpenFlow (alta sinergia entre conceitos)
- Distribuído como GPL em 2008 (muito usado)
- Mantido pela comunidade de pesquisa

# NOX ([www.noxrepo.org](http://www.noxrepo.org))

- Linux
- C++ and Python (“cola” para módulos C++)
  - *Cooperative multithreading*
- Sistema de componentes
- Programação orientada a eventos
- Algumas aplicações distribuídas com o SW

# NOX - Exemplo Repetidor



Simples repetidor na rede

Encaminha pacotes do porta 1 para a 2 e vice-versa

# NOX - Exemplo Repetidor

## Programa NOX

```
def simple_repeater():
    # Repeat Port 1 to Port 2
    p1 = {IN_PORT:1}
    a1 = [(OFPAT_OUTPUT, PORT_2)]
    install(switch, p1, HIGH, a1)

    # Repeat Port 2 to Port 1
    p2 = {IN_PORT:2}
    a2 = [(OFPAT_OUTPUT, PORT_1)]
    install(switch, p2, HIGH, a2)
```

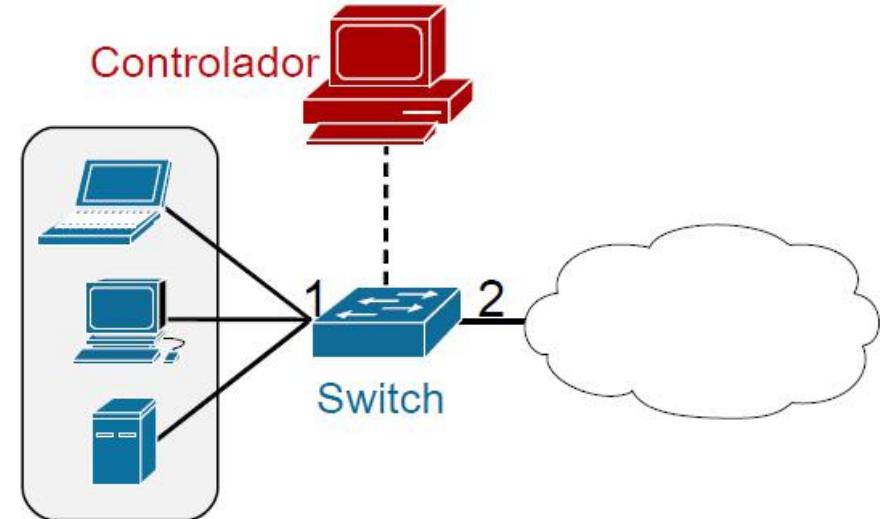


Tabela de Fluxos

Prioridade	Padrão	Ação	Contadores
HIGH	IN_PORT:1	OUTPUT:2	(0,0)
HIGH	IN_PORT:2	OUTPUT:1	(0,0)

# Trema (<http://trema.github.io/trema/>)

## Implementação de controladores

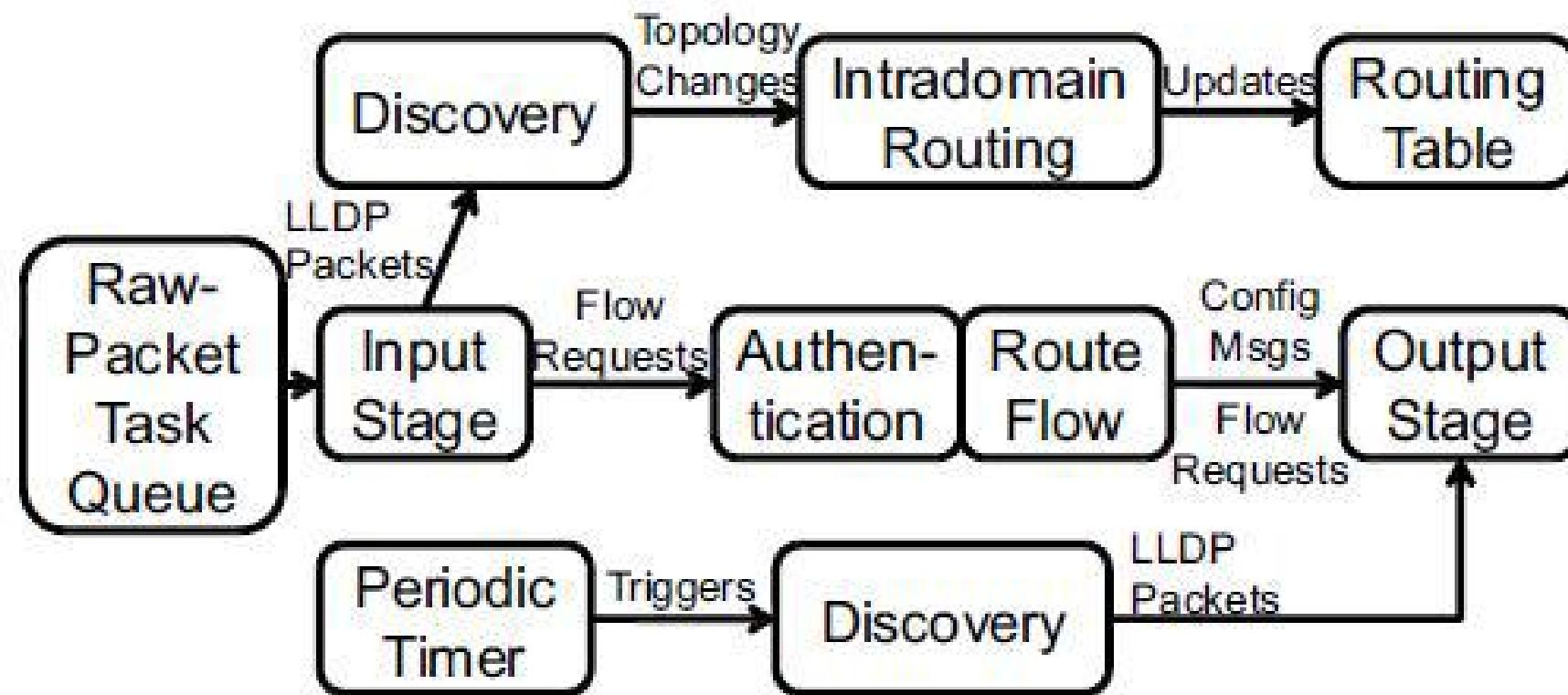
### OpenFlow

- Ruby e C
- Emulação

```
class MyController < Controller
  # packet_in handler
  def packet_in dpid, msg
    send_flow_mod_add(
      dpid,
      :match => ExactMatch.from(msg),
      :buffer_id => msg.buffer_id,
      :actions => ActionOutput.new(msg.in_port+1)
    )
  end
end
```

Concebido como Sistema Operacional de rede

- Tenta explorar o paralelismo dentro de uma única máquina



- Linguagem declarativa (datalog)
- Permite especificar políticas de gerência e segurança
- Especifica um conjunto de regras com critérios complexos para definir precedência e resolver conflitos

Exemplo: Fluxo de VoIP com limite máximo de jitter e latência

*latency( $\overline{Flow}$ , 100)  $\Leftarrow$  Prot = voip*

*jitter( $\overline{Flow}$ , 5)  $\Leftarrow$  Prot = voip*

*band( $\overline{Flow}$ , 3000)  $\Leftarrow$  Prot = 1233*

# Frenetic (<http://frenetic-lang.org/>)

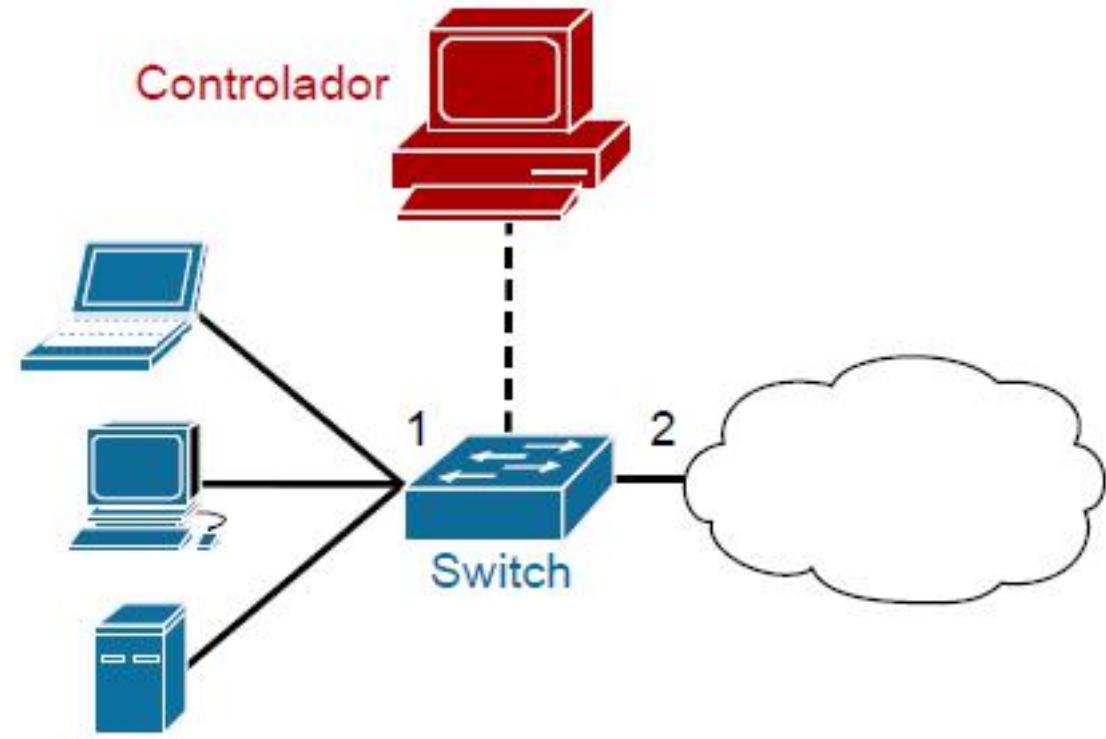
- Foco na programação de redes OpenFlow
- Oferece um nível de abstração mais alto
- Principais objetivos:
  - Permitir o desenvolvimento de consultas complexas às tabelas de fluxos (regras, estatísticas)
  - Simplificar a composição de regras
- Implementado sobre o NOX, em Python

# Frenetic comparação com NOX

Programa NOX do repetidor:

```
def simple_repeater():
    # Repeat Port 1 to Port 2
    p1 = {IN_PORT:1}
    a1 = [(OFPAT_OUTPUT, PORT_2)]
    install(switch, p1, HIGH, a1)

    # Repeat Port 2 to Port 1
    p2 = {IN_PORT:2}
    a2 = [(OFPAT_OUTPUT, PORT_1)]
    install(switch, p2, HIGH, a2)
```



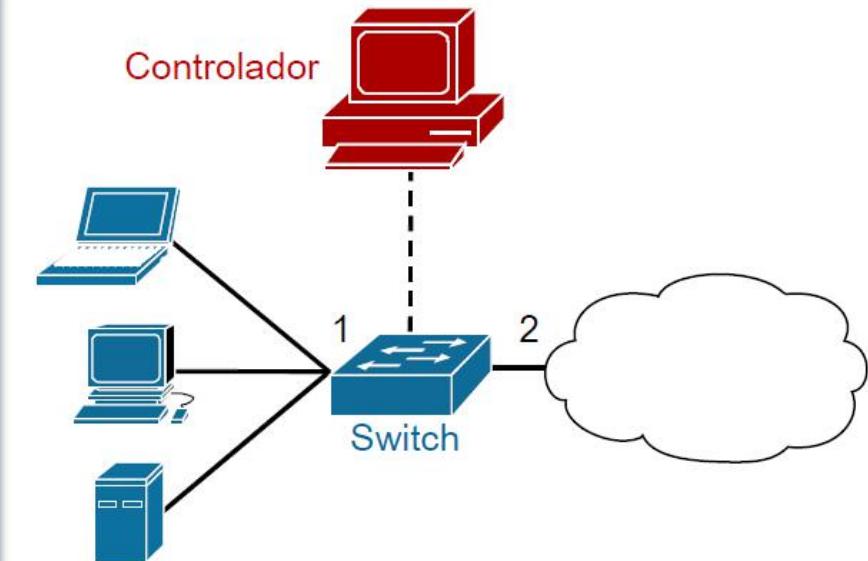
Para se acrescentar um monitor de tráfego Web,  
o código deve ser alterado e integrado

# Frenetic comparação com NOX

## Programa NOX do repetidor com monitor

```
def switch_join(switch):
    pat1 = {inport:1}
    pat2 = {inport:2}
    pat2web = {in_port:2, tp_src:80}
    install(switch, pat1, DEFAULT, None, [forward(2)])
    install(switch, pat2web, HIGH, None, [forward(1)])
    install(switch, pat2, DEFAULT, None, [forward(1)])
    query_stats(switch, pat2web)

def stats_in(switch, xid, pattern, packets, bytes):
    print bytes
    sleep(30)
    query_stats(switch, pattern)
```



# Frenetic comparação com NOX

## Programa Frenetic

```
# Static repeating between ports 1 and 2
def repeater():
    rules=[Rule(inport:1, [forward(2)]),
           Rule(inport:2, [forward(1)])]
    register(rules)
```

**Repeater**

```
# Monitoring Web traffic
def web_monitor():
    q = (Select(bytes) *
         Where(inport:2 & tp_src:80) *
         Every(30))
    q >> Print()
```

**Monitor**

**Repeater + Monitor**

```
# Composition of two separate modules
def main():
    repeater()
    web_monitor()
```

# Onix (Google, Nec, Yahoo)

- Controlador que distribuir o estado da rede em múltiplos controladores distribuídos;
- Network Information Base (NIB)
  - Estrutura de dados que representa a visão global da rede através de um grafo
  - Aplicações podem ler e escrever na NIB
  - Automaticamente atualiza switches e controladores

# Onix (Google, Nec, Yahoo)

- Código fechado, apenas descrito no artigo;
- Network Information Base (NIB)
  - Estrutura de dados que representa a visão global da rede através de um grafo
  - Aplicações podem ler e escrever na NIB
  - Automaticamente atualiza switches e controladores
- Ênfase: escalabilidade e confiabilidade

# Onix (Google, Nec, Yahoo)

- Controle da rede é feito alterando-se a NIB
  - Alterações se transformam em comandos para os elementos de comutação
- Escalabilidade:
  - NIB é dividida entre controladores
- Confiabilidade (reliability):
  - NIB é replicada (mestre-escravo)
- Consistência:
- Algoritmos de consenso em casos extremos

# Click (<http://www.read.cs.ucla.edu/click/click>)

Compartilha o objetivo de permitir que equipamentos de rede sejam programáveis.

- Enfatiza a modularidade.
- Permite criar módulos de processamento de pacotes customizados.
- Porém, foca exclusivamente em switches de software (módulo do kernel do Linux).
- OpenFlowClick: interface OF para o Click.

# Floodlight (<http://www.projectfloodlight.org/floodlight/>)

Controlador OF para redes corporativas baseado na linguagem Java, licença Apache.

- Apoiado pela Big Switch Networks
- Módulos principais são escritos em Java ou Python
  - Módulos exportam serviços
- Permite integrar com redes não openflow.
- Compatível com a ferramenta de simulação Mininet

# Simple Network Access Controle (SNAC)

- Controlador OpenFlow que utiliza uma interface web.
- Provê uma ferramenta de monitoramento gráfico mas não é um ambiente de programação genérica
- <http://archive.openflow.org/wp/deploy-production-controllersetup/#SNAC>

# Simple Network Access Controle (SNAC)

Policy Manager

user: admin Logout

## Overview

Network Overview ▶

Switches	Uptime: 1 day 16 hours 52 minutes 53 seconds
Hosts	CPU Load: 1%
Users	Flows/sec: 7
Locations	
Groups	
Network Events Log	

**Server Information**

Switches:	3 / 3 / 0
Locations:	18 / 18 / 0
Hosts:	17 / 22 / 4
Users:	1 / 3 / 0

**Entity Counts (Active/Total/Unregistered)**

**Policy Statistics**

Total Drops:	47698
Total Rules:	15

Top 5 Switch Ports by Tx Bandwidth

Port	Tx Bandwidth (KB/second)
Built-in:grasshopper:eth2	~72
Built-in:grasshopper:eth5	~20
Built-in:grasshopper:eth4	~18
Built-in:bumblebee:eth2	~10
Built-in:bumblebee:eth3	~10
Built-in:ladybug:eth5	~8

Internal Network | 6 Active Admins | Version: 0.3.0+build43

# NEC Programmable Flow

Produto comercial

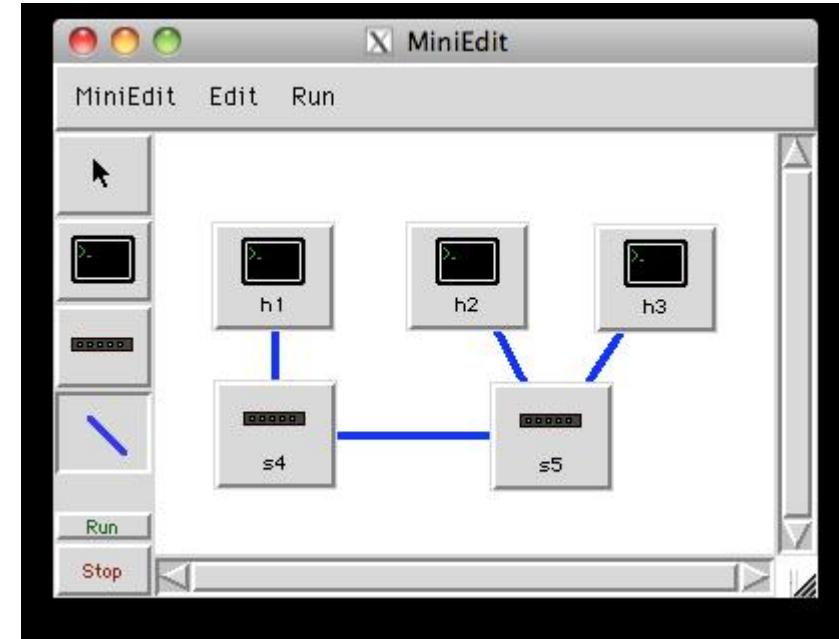
- Ferramenta centralizada de monitoramento
- Switches PF5420 e PF5820:
  - compatíveis com OpenFlow

**UNIVERGE PF5820**



# Mininet

- Ferramenta de simulação de SDN
- Permite a rápida prototipação de grandes redes utilizando apenas um computador



# POX

- NOX se tornou o controlador mais popular
- A maior parte dos usuários usam só Python
- Mas essa não era o objetivo inicial
- **Observação 1:** NOX é difícil de instalar e manter
- Um número considerável de usuários tem problemas para compilar e executar o NOX
- Montagem relativamente complexa, com um número significativo de dependências

# POX

- Observação 2: a API Python não era ideal para aplicações completas
  - Afinal, o plano era usá-la só para integração
  - Mas muitas aplicações foram desenvolvidas inteiramente em Python
  - A integração com Python adicionou um custo considerável de manutenção e instalação
    - Desnecessário para quem quer apenas C++

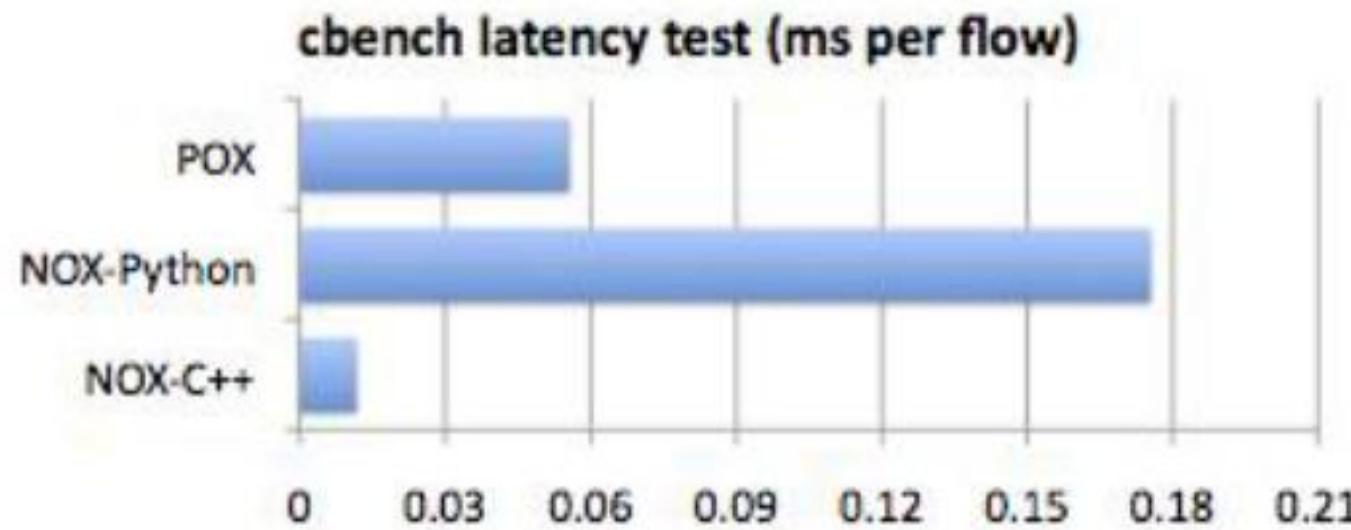
# POX

- Facilidade de instalação é importante
  - Dependências devem ser escolhidas com cuidado
- Escolha uma linguagem
  - Integrar duas ou mais é difícil... “e ninguém quer”
- Remoção de Python do NOX simplifica tudo
  - E torna-o melhor para quem só quer C++
- Criação de um controlador em Python permite aproveitar as boas idéias e torná-las simples

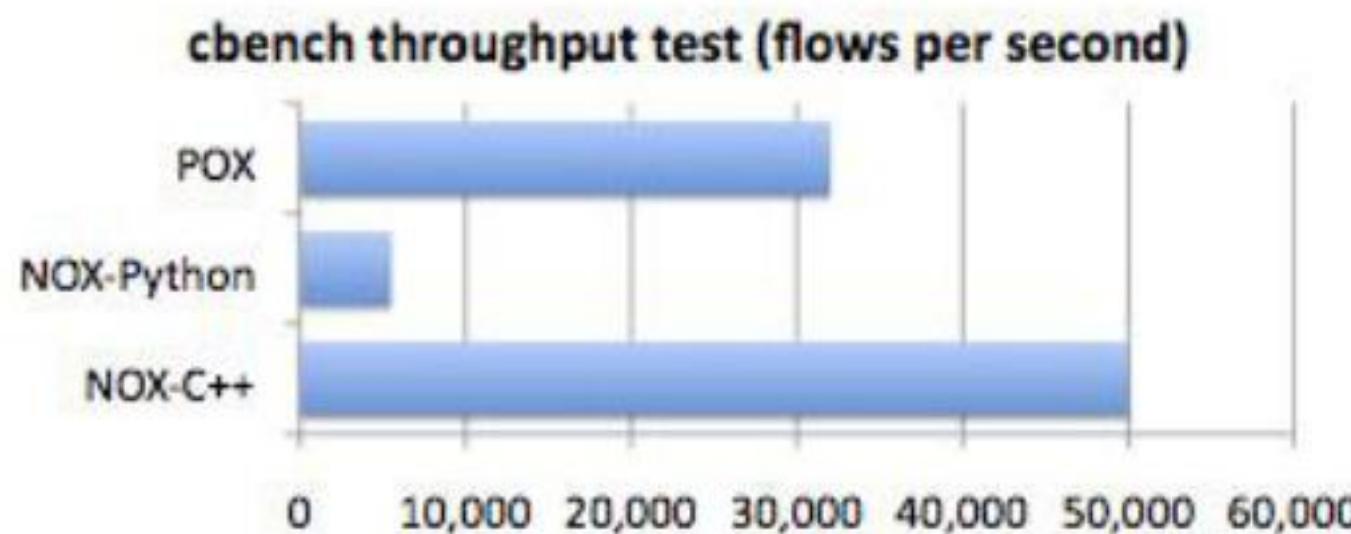
# POX

- Nova plataforma em Python puro
- Poucas dependências
- Aproveita as qualidades do NOX
- Instalação fácil e flexibilidade no desenvolvimento
- Bom para pesquisa, não para desempenho
- Mas melhor que NOX com Python!

# POX



Fonte: Murphy Mccauley



# POX

- Dependências: apenas Python (2.7)!
- Herda elementos do NOX
  - cooperative multithreading, componentes
  - montagem e desmontagem de pacotes
- Interface OF bastante melhorada

# POX principais módulos

- log: mensagens de erro, aviso e registro em geral
- lib: bibliotecas auxiliares, como:
- threads
- manipulação de endereços
- processamento de pacotes
- openflow: implementação do protocolo
- core.py: criador do sistema

# POX orientação a eventos

- Qualquer módulo pode se registrar para ser informado de eventos e pode criar novos eventos para disparar
- Eventos básicos: packetIn e timers

# POX

Exemplo simples: construir um switch

1. git clone <http://noxrepo.org/git/pox>
2. cd pox
3. vim ext/switch.py
4. ./pox.py switch

# Aplicações

- Controle de acesso, segurança
- Gerência de redes
- Redes domiciliares
- Gerência de energia
- Comutador virtual distribuído
- Roteador expansível (*scale-out*)
- Datacenters multi-usuários

# Aplicações - Controle de Acesso

- Contexto que originou OF (Ethane)
- Implementação de políticas fica mais simples com a visão global
- Visão de alto nível permite disassociar pessoas de identificadores de hardware
- Substituição de middle-boxes
- Políticas podem se adaptar a cenários mais abrangentes
- “A pode acessar B se estiver na rede X, mas não se o acesso vier dos pontos de trabalho na área Y”

# Aplicações - Gerência de Rede

A visão global oferece uma forma simples de agregar informações sobre todos os eventos relevantes para a administração da rede

A interface de programação dos switches oferece uma forma simples para atuar sobre o sistema

A escolha do nome “NIB” não parece aleatória...

Há uma relação forte com a área de gerência

# Aplicações - Redes Domiciliares

O que está acontecendo na minha rede?

Padrões de tráfego doméstico podem ser mais reveladores para deteção de malware

O controlador da rede pode exportar informações para um provedor de serviços de gerência e configuração

A visão de todos os fluxos cria uma oportunidade para entender os padrões de acesso dos moradores e detectar desvios

# Aplicações - Gerência de Energia

Técnicas de posicionamento e distribuição de carga se beneficiariam de um maior conhecimento do que passa pela rede

Migração de VMs pode ser usado para distribuir carga. SDNs podem simplificar o processo de migração

# Aplicações - Comutador Virtual Distribuído

- Em datacenters, o ambiente virtualizado pode permitir o uso de switches virtuais como base de instalação da SDN
- A visão global da rede podem ser usada para criar a abstração de um switch único interligando as VMs de um usuário
- Isolamento de tráfego pode se tornar mais simples

# Aplicações - Roteador Expansível

- Roteadores de borda de alta capacidade têm custo bastante elevado
- Um conjunto de switches controlados por uma SDN podem exportar a visão de um comutador único
- Um conjunto de rotas podem ser transformado em regras de encaminhamento entre switches que sejam agrupada para substituir o roteador

# Aplicações - Datacenter Multiusuários

Um switch virtual pode ser usado como abstração para interligar as máquinas de um cliente

Cada usuário (tenant) pode ser levado a enxergar a rede como um único switch que interliga suas máquinas

# Desafios

- Visão da rede/abstrações
- Virtualização
- API para programação dos switches
- Especificação de aplicações
- Depuração
- Distribuição para desempenho e tol. falhas

# Desafios – Visão da rede

Qual a visão da rede que deve ser oferecida pelo controlador da SDN?

Qual a melhor abstração para cada tipo de aplicação que pode existir sobre uma rede?

# Desafios – Virtualização

FlowVisor oferece uma forma de dividir e isolar os recursos da rede, mas há outras

Pode-se traçar um paralelo entre as diversas formas de virtualização de servidores, em diferentes níveis da arquitetura

# Desafios - API para programação

OpenFlow é apenas uma forma de se oferecer tal API

Que outros tipos de interface poderiam ser criados?

É necessário que todos os nós exportem a mesma funcionalidade, ou pode-se trabalhar com modelos baseados em borda?

# Desafios – Especificação de aplicações

Qual a forma mais adequada de se programar um controlador de rede?

Frenetic e FML, por exemplo, usam modelos bem diferentes

Técnicas de representação de conhecimento de outras áreas poderiam ser úteis? Semiótica, sistemas especialistas, etc...

# Desafios - Distribuição para desempenho

A noção da visão global centralizada é apenas uma forma simples de se encarar o problema

Como distribuir a visão da rede se aspectos de desempenho e confiabilidade são importantes?

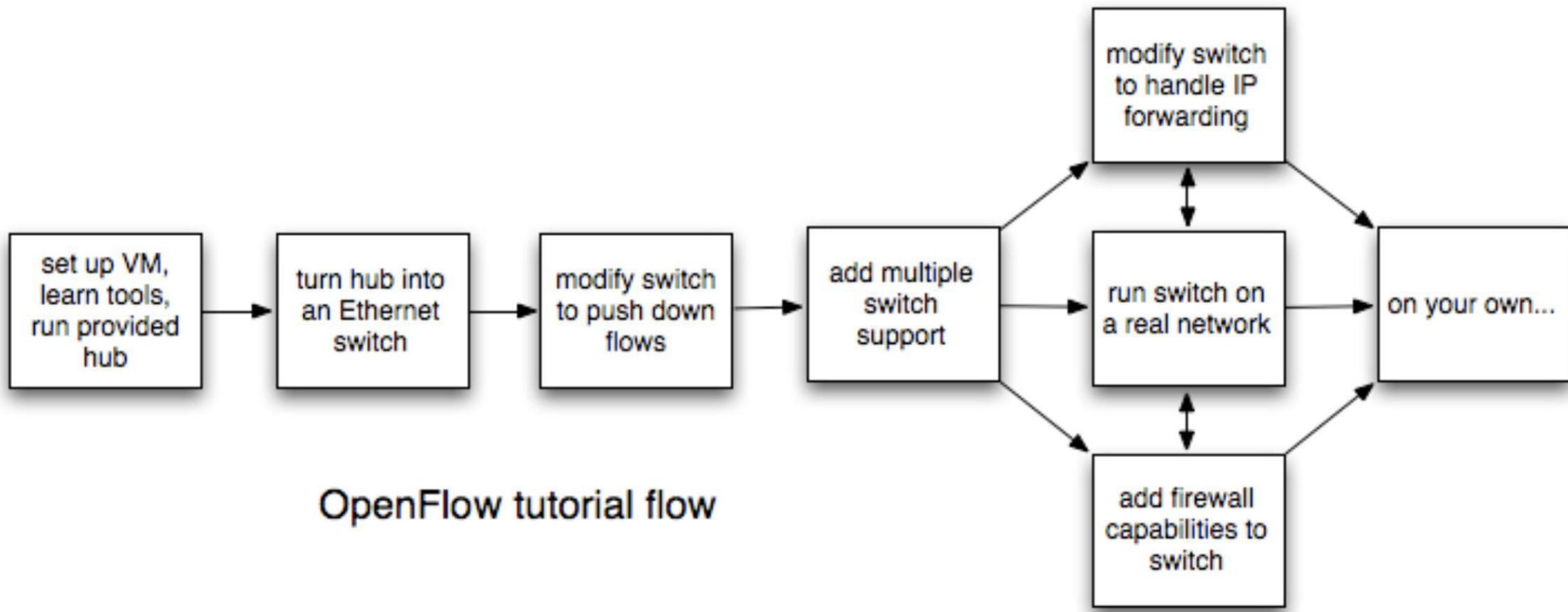
Ônix é apenas um ponto no espaço de soluções

# Observações Finais

- O interesse da indústria e da comunidade científica está apenas começando
- O caminho ainda não está completamente delineado
  - há oportunidades e riscos
- Sites úteis:

[noxathome.org](http://noxathome.org), [noxrepo.org](http://noxrepo.org), [openflow.org](http://openflow.org),  
[openwrt.org](http://openwrt.org),  
[openvswitch.org](http://openvswitch.org), [opennetworking.org](http://opennetworking.org),  
[opennetsummit.org](http://opennetsummit.org)

# Hands On - Mininet



# Mininet - Pré-requisitos

Putty

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

Xming - <http://sourceforge.net/projects/xming/>

Virtual Box/Vmware

Mininet - <https://github.com/mininet/mininet/wiki/Mininet-VM-Images>

# Mininet - Definições

O Mininet simulador de rede de código aberto é projetado para apoiar a investigação e educação no domínio dos sistemas de SDN. Mininet cria uma rede simulada que executa o software real sobre os componentes da rede, de modo que pode ser usado para testar de forma interativa o software de rede.

# Mininet - Definições

Mininet é projetado para criar redes virtuais SDN, consiste em um controlador OpenFlow, uma rede Ethernet fixa de múltiplos switches Ethernet habilitado para OpenFlow, e várias máquinas conectadas aos switches. Ele foi construído com funções que suportam a utilização de diferentes tipos de controladores e switches.

Nós podemos criar também cenários personalizados complexas usando a API do Python Mininet.

# Mininet - Network Namespaces

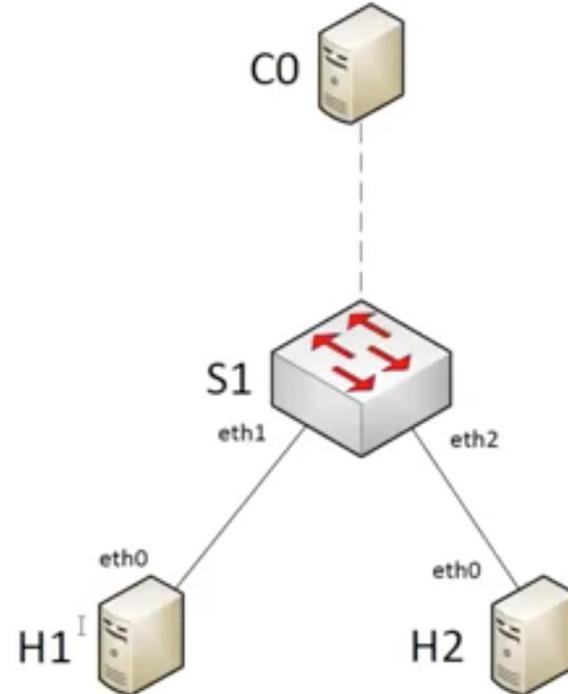
Mininet usa Linux Network Namespaces para criar nós virtuais na rede simulada. Esta é uma maneira leve e rápido para criar nós virtuais, mas ele não fornece máquinas virtuais totalmente separados e não é possível salvar as configurações em cada um dos nós virtuais após a simulação é desligado.

# Mininet - Passo a Passo

1. Instalar a máquina virtual
2. Efetuar o login/senha mininet/minet
3. Instalar o ambiente gráfico
  1. sudo apt-get update
  2. sudo apt-get install xinit <enviroment>
    1. lxde - compacto e rápido
    2. Flwm - pequeno porém primitivo
    3. Ubuntu-desktop - ambiente ubuntu completo
  3. sudo apt-get install xinit lxde

# Mininet - Start

## sudo mn



- Topologia (h1, h2, s1, c0)
- sudo mn
- nodes
- net
- dump
- pingall
- h1 ping h2 -c3

# Mininet

```
mininet> help
```

Documented commands (type help <topic>):

```
=====
```

```
EOF  gterm iperfudp nodes      pingpair    py    switch  
dpctl help link  noecho      pingpairfull quit  time  
dump intfs links   pingall     ports      sh    x  
exit iperf net     pingallfull px      source xterm
```

# Mininet

Exibir Nós: mininet>nodes

Exibir Link: mininet>net

Informações sobre todos os nós: mininet>dump

Executar um comando no host:

mininet>h1 ifconfig -a

mininet>h2 ifconfig -a

mininet>s1 ifconfig -a

mininet>c0 ifconfig -a

# Mininet

```
mininet>h1 ps -a
```

```
mininet>s1 ps -a
```

Testar a conectividade entre hosts

```
mininet>h1 ping -c 1 h2
```

# Mininet

Mininet>pingall (testa todos os links)

Executar um servidor web e um cliente simples

```
mininet>h1 python -m SimpleHTTPServer 80 &  
Mininet>h2 wget -O - h1  
Mininet>h1 ps -a  
Mininet>h1 kill python(pid)
```

Sair do mininet> exit

# Mininet

Mininet>mn -c (Caso o mininet trave)

Executar um teste de regressão

~\$ sudo mn --test pingpair

# Mininet - Topologia

--topo=TOPO

**minimal** - 1 switch e 2 hosts;

**single,X** - a single switch com X hosts attached;

**linear,X** - X switches lineares conectados com um host cada;

**tree,X** - a tree topology with X fanout

# Mininet - Topologia

**--switch=SWITCH**

Criar diferentes tipos de switches:

ovsk - Default Open vSwitch pre-instalado na VM

user - Switch executando software namespace (lento)

# Mininet - Topologia

**--controller=CONTROLLER**

ovsc - Default OVS Controller preinstalado

nox - Controlador NOX

remote - Não cria um controlador porém permite a controladores externos escutar a rede.

# Mininet - Topologia

--mac

set uma maneira mais simples de MAC addresses para os dispositivos

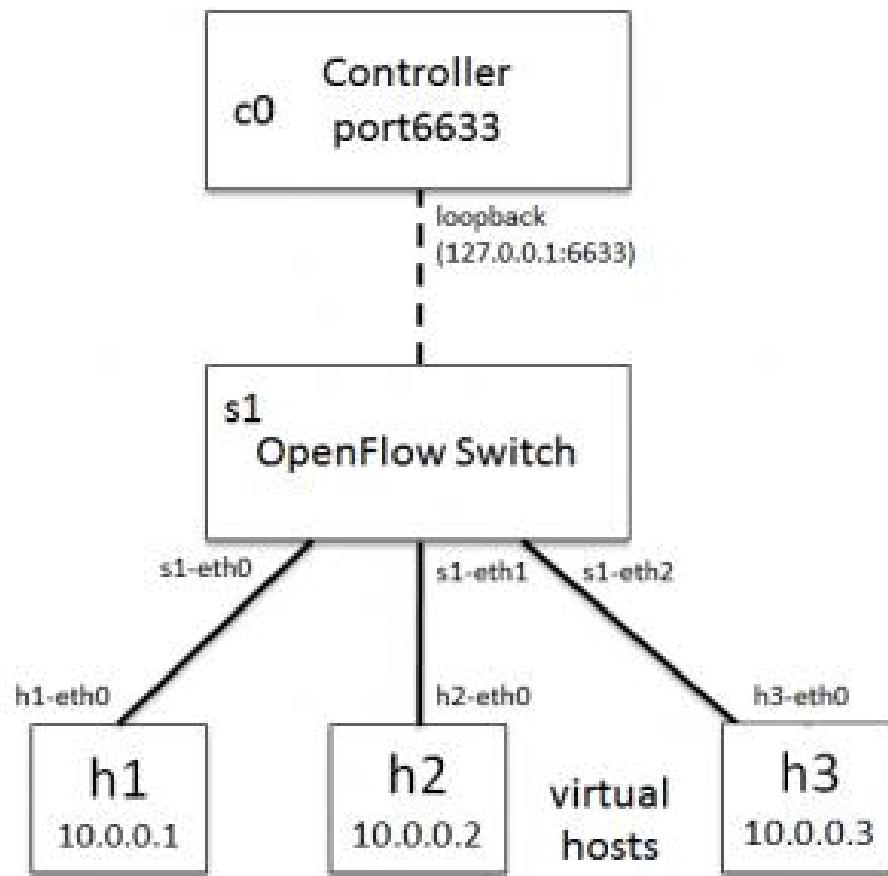
# Mininet - Topologia

```
~$ sudo mn --topo single,3
```

Desenhe a  
topologia

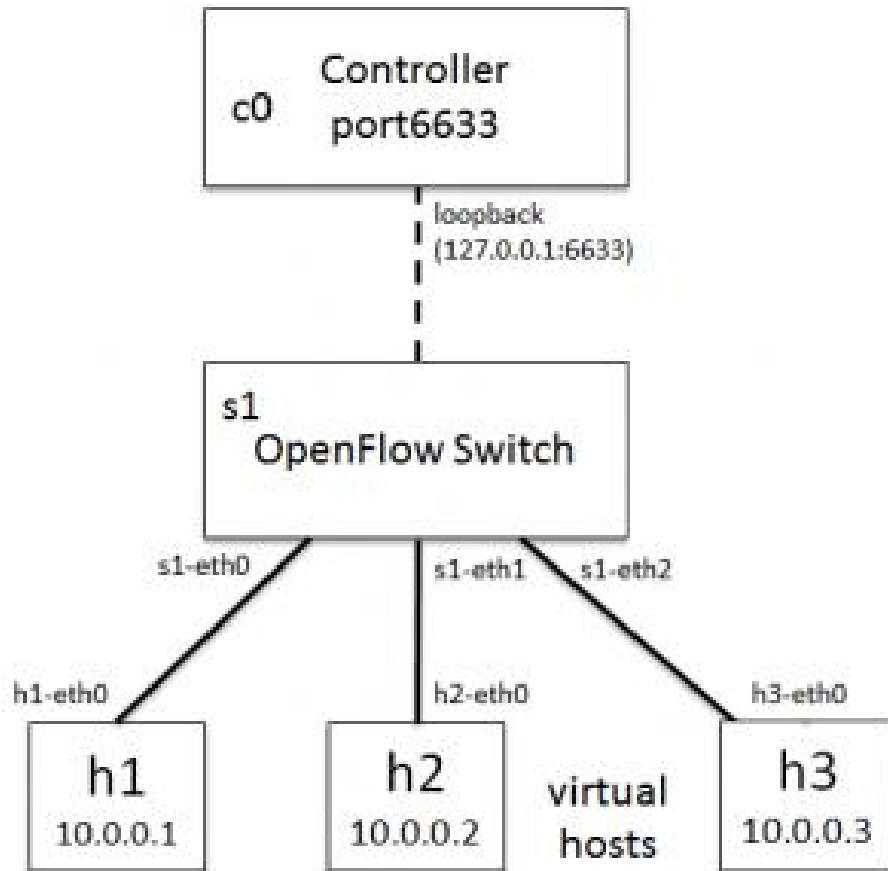
# Mininet - Topologia

```
~$ sudo mn --topo single,3
```



# Mininet - Topologia

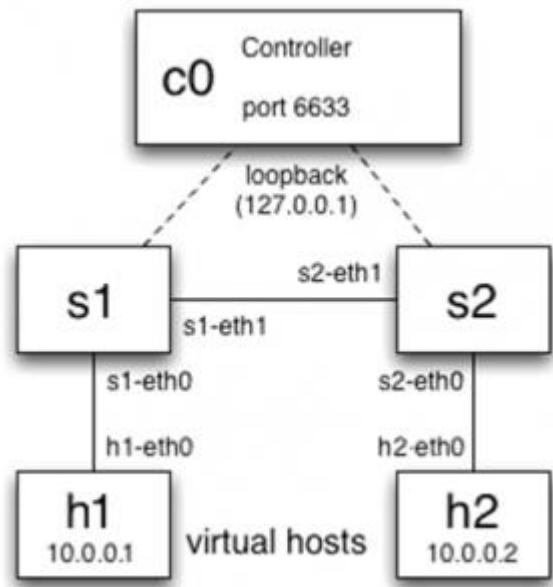
```
$ sudo mn --arp --topo single,3 --mac --switch ovsk --controller remote
```



- **-mac:** Auto set MAC addresses
- **-arp:** Populate static ARP entries of each host in each other
- **-switch:** ovsk refers to kernel mode OVS
- **-controller:** remote controller can take IP address and port number as options

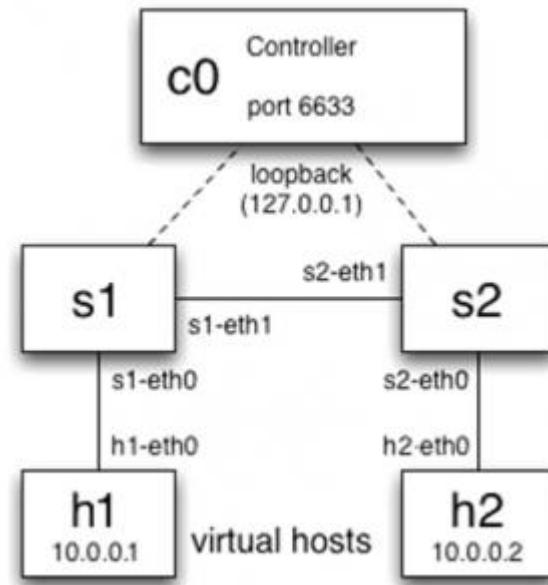
# Mininet - Topologia

```
$ sudo mn --topo linear
```



# Mininet - Topologia

```
$ sudo mn --topo linear --switch ovsk --controller remote
```



# Mininet - Topologia

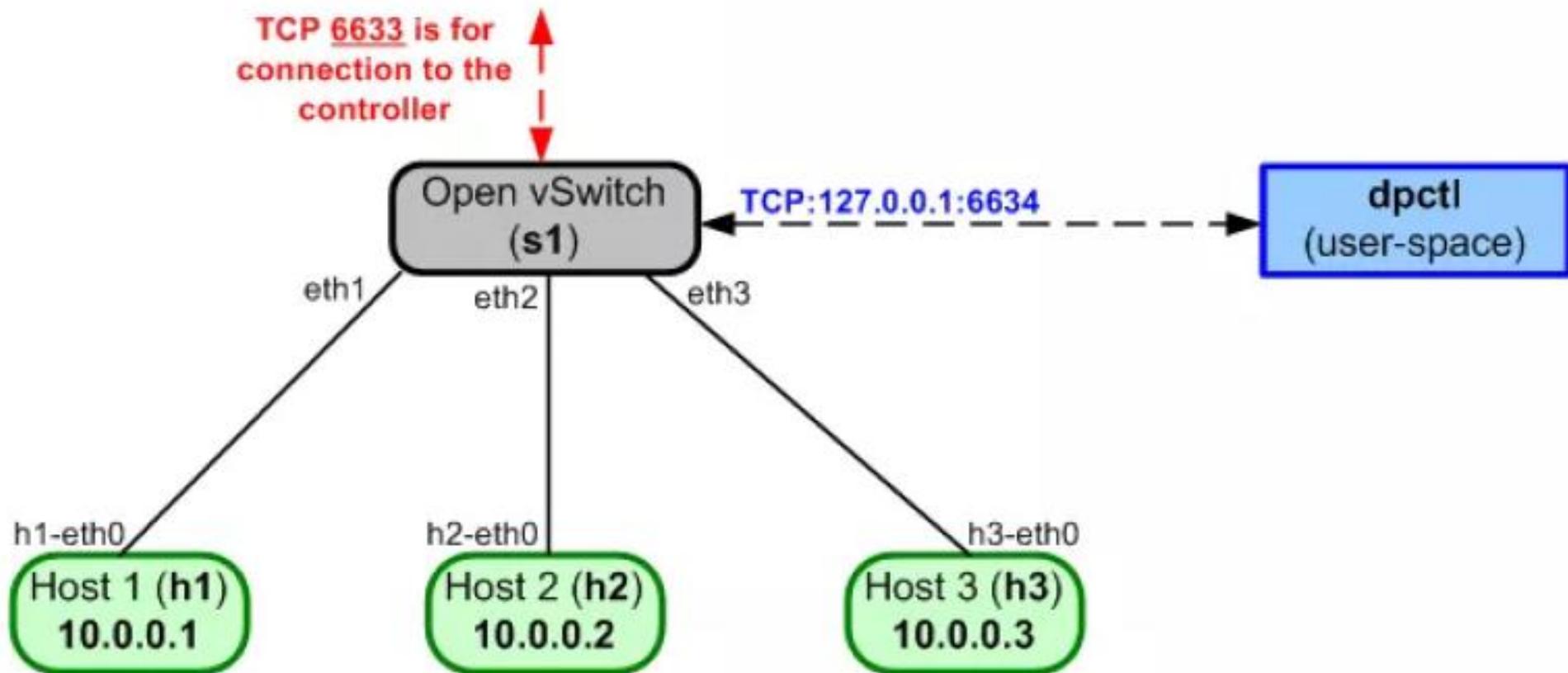
```
~$ sudo mn --topo single,3  
~$ sudo mn --topo linear,3  
~$ sudo mn --test pingall --topo single,3  
~$ sudo mn -test pingall --topo linear,4
```

# Mininet - Topologia

```
sudo mn --topo=single,3  
--mac --controller=remote
```

# Mininet - Topologia

```
sudo mn --topo=single,3 --mac --controller=remote
```



# Mininet - Variações do Link

```
~$ sudo mn --link tc,bw=10,delay=10ms
```

```
mininet> iperf
```

```
Mininet> h1 ping -c 10 h2
```

Se o atraso para cada link é de 10 ms, o tempo de ida e volta (RTT) deve ser de cerca de 40 ms, uma vez que a solicitação ICMP atravessa dois links (um para o interruptor, um para o destino) e a resposta ICMP atravessa duas ligações que vêm de volta .

# Mininet - Nível de Informação

```
~$ sudo mn -v debug
```

```
~$ sudo mn -v output
```

# Mininet - Personalizando a topo

```
~$ cd mininet/custom
```

```
~$ ls -l
```

```
~$ vim topo-2sw-2host.py
```

```
~$ sudo mn --custom ~/mininet/custom/topo-2sw-  
2host.py --topo mytopo --test pingall
```

# Mininet - Mac (simples)

```
~$ sudo mn
```

```
~$ h1 ifconfig
```

```
~$ sudo mn --mac
```

```
~$ h1 ifconfig
```

# Mininet - xterm

```
~$ sudo mn -x
```

```
~$ dpctl dump-flows tcp:127.0.0.1:6634
```

```
mininet>h1 ping 10.0.0.2
```

```
~$ dpctl dump-flows tcp:127.0.0.1:6634
```

# Mininet - outros switchs

```
~$ sudo mn --switch user --test iperf
```

```
~$ sudo mn --switch ovsk --test iperf
```

# Mininet - benchmark

```
~$ sudo mn --test none
```

Para colocar o switch em outro namespace.

```
~$ sudo mn --innamespace --switch user
```

# Mininet - Interpretador Python

```
~$ sudo mn
```

```
Mininet> py 'hello' + 'world'
```

Variáveis locais:

```
mininet>py locals()
```

Novo métodos e propriedades

```
Mininet>py dir(s1)
```

# Mininet - Interpretador Python

Ler a documentação dos métodos de um Nó.

```
mininet> py help(h1)
```

Acessando um método:

```
mininet>py h1.IP()
```

Link Up e Down

```
mininet>link s1 h1 down
```

```
mininet>link s1 h1 up
```

# Mininet - XTerm Display

```
mininet> xterm h1 h2
```

# Mininet - outros exemplos

## Diretório Exemplos

```
$ sudo ~/mininet/examples/sshd.py
```

```
$ ssh 10.0.0.1
```

```
$ ping 10.0.0.2
```

```
exit
```

# Mininet - Recapitulando

```
sudo mn
```

```
nodes,net,dump,xterm h1 h2 s1, pingall
```

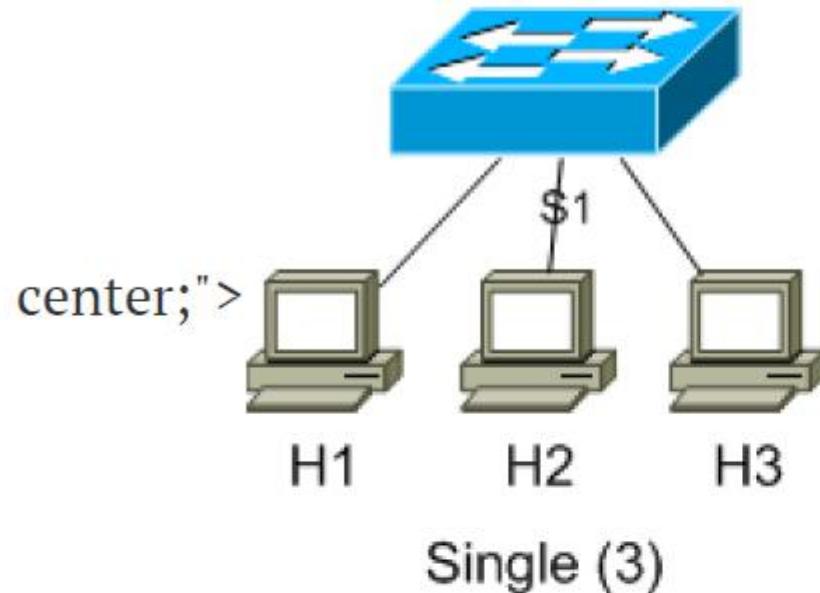
```
link h1 s1 down
```

```
h1 ping -c 1 h2
```

# Mininet - Recapitulando

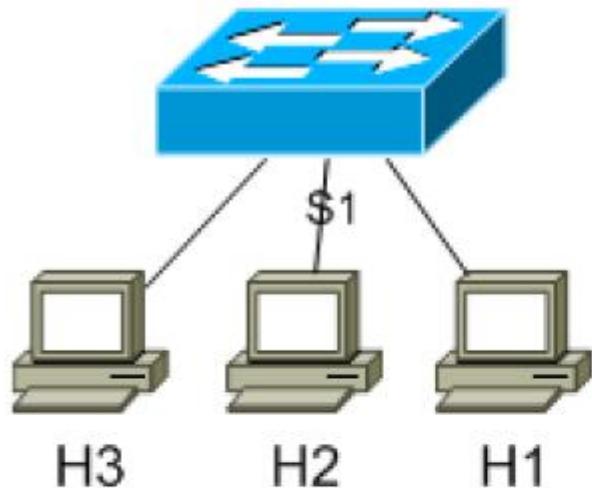
```
sudo mn --topo minimal
mininet> net
h1 h1eth0:s1-eth1
h2 h2eth0:s1-eth2
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0
```

# Mininet - Recapitulando



```
sudo mn --topo single,3
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
```

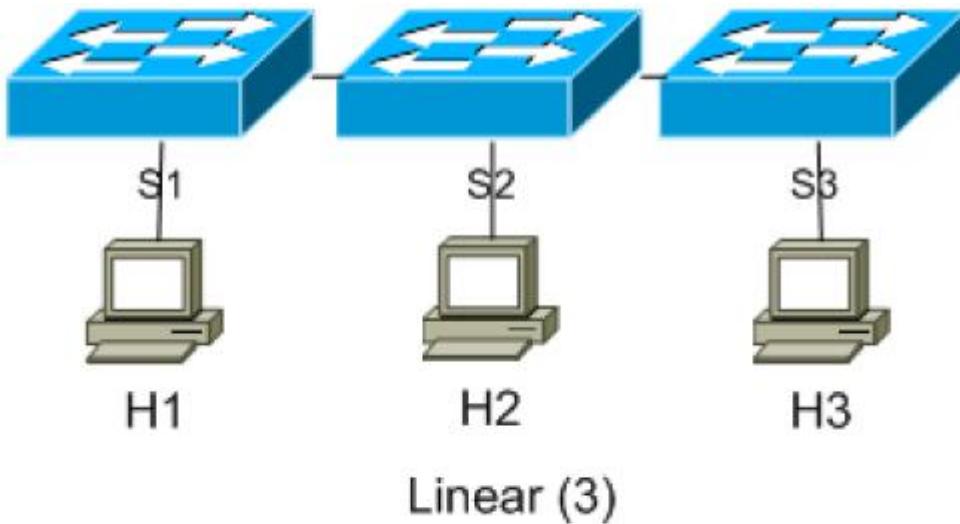
# Mininet - Recapitulando



Reversed (3)

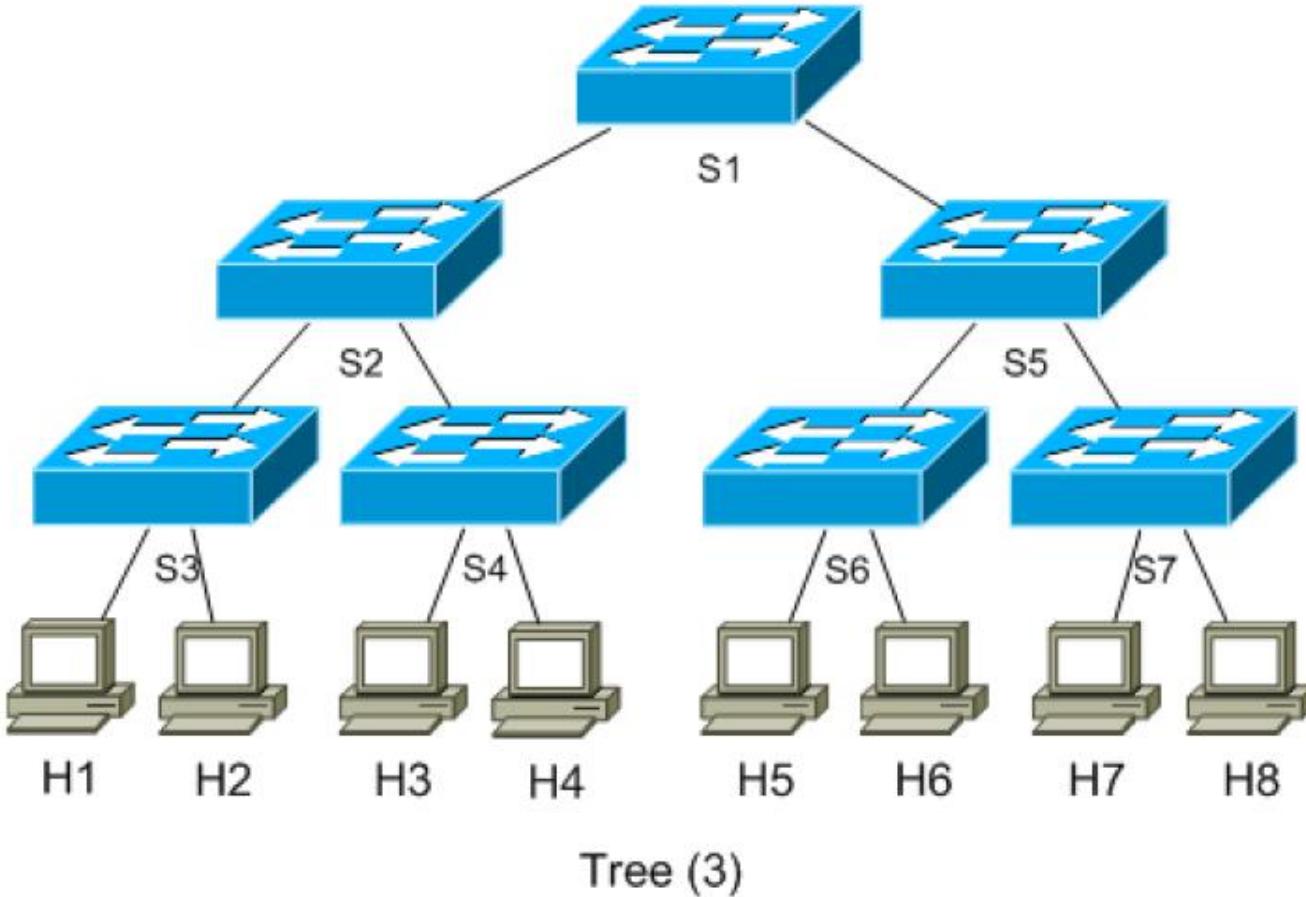
```
sudo mn --topo reversed,3
mininet> net
h1 h1-eth0:s1-eth3
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth1
s1 lo: s1-eth1:h3-eth0 s1-eth2:h2-eth0 s1-eth3:h1-eth0
```

# Mininet - Recapitulando



```
sudo mn --topo linear,3
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo: s3-eth1:h3-eth0 s3-eth2:s2-eth3
```

# Mininet - Recapitulando



```
sudo mn --topo tree,3
```

```
mininet> net
```

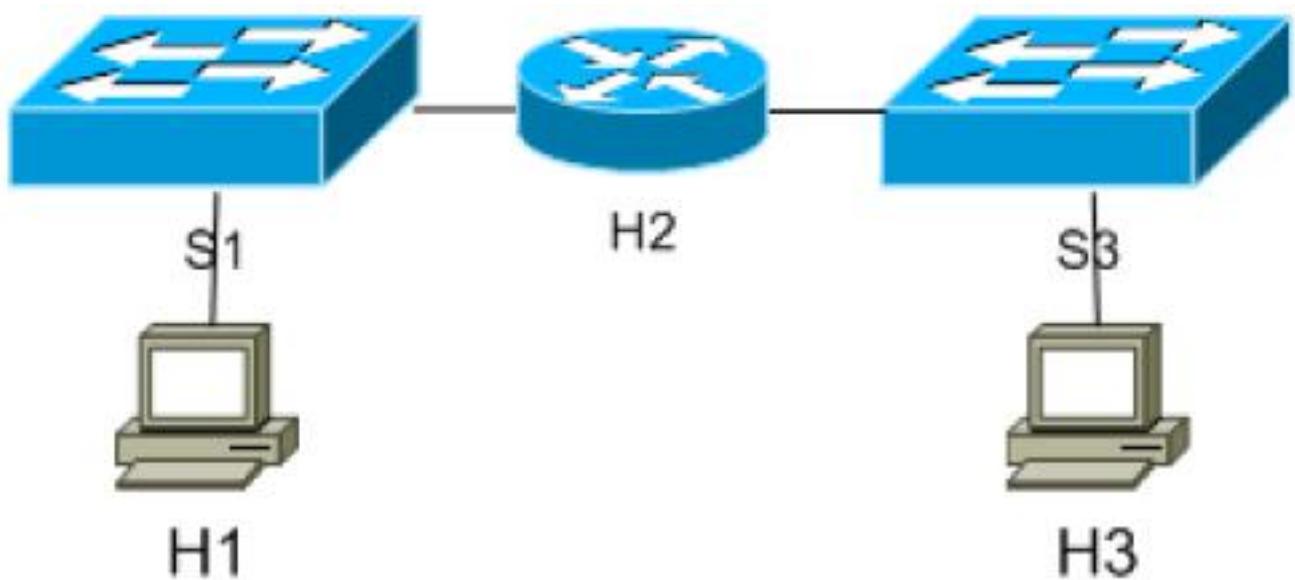
# Mininet - Miniedit

<https://github.com/mininet/mininet/blob/master/examples/miniedit.py>

# Mininet



# Mininet - Router



# Mininet - Codando

```
1  from mininet.topo import Topo
2
3  class Router_Topo(Topo):
4      def __init__(self):
5          "Create P2P topology."
6
7          # Initialize topology
8          Topo.__init__(self)
9
10         # Add hosts and switches
11         H1 = self.addHost('h1')
12         H2 = self.addHost('h2')
13         H3 = self.addHost('h3')
14         S1 = self.addSwitch('s1')
15         S2 = self.addSwitch('s2')
16
17         # Add Links
18         self.addLink(H1, S1)
19         self.addLink(H2, S1)
20         self.addLink(H2, S2)
21         self.addLink(H3, S2)
22
23     topos = {
24         'router': (lambda: Router_Topo())
25     }
```

# Mininet - Executando

```
sudo mn --custom router.py --topo router
```

# Mininet - Links

```
mininet> net  
h1 h1-eth0:s1-eth1  
h2 h2-eth0:s1-eth2 h2-eth1:s2-eth1  
h3 h3-eth0:s2-eth2  
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0  
s2 lo: s2-eth1:h2-eth1 s2-eth2:h3-eth0
```

# Mininet - Pingar

Mininet> h1 ping h2

Mininet> h2 ping h3

Mininet> h1 ping h3

# Mininet - configurando o roteamento

```
mininet> h1 ifconfig h1-eth0 192.168.12.1 netmask 255.255.255.0
mininet> h2 ifconfig h2-eth0 192.168.12.2 netmask 255.255.255.0
mininet> h2 ifconfig h2-eth1 192.168.23.2 netmask 255.255.255.0
mininet> h3 ifconfig h3-eth0 192.168.23.3 netmask 255.255.255.0
mininet> h1 route add default gw 192.168.12.2
mininet> h3 route add default gw 192.168.23.2
mininet> h2 sysctl net.ipv4.ip_forward=1
```

# Mininet – Pingar de novo

```
Mininet> h1 ping -c 1 h3
```

```
Mininet> h1 ping -c 1 192.168.23.3
```

# Mininet - Nova Topologia (exercício)

Host ----- switch ----- switch ----  
- host

# Mininet - Resposta

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
        "Create custom topo." # Initialize topology
        Topo.__init__( self )
        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )
        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

```
sudo mn --custom lab1.py --topo mytopo
```

```
*** TODO ***
nodes
net
dump
h1 ifconfig -a
h1 ping -c 3 h2
pingall
xterm h1 h2
h1> ping -c 3 10.0.0.2
h2> iperf -s
h1> iperf -c 10.0.0.2
exit
```

# Lab 2 - Controle do Switch

dpctl: utilitário em linha de comando para enviar uma mensagem Openflow para um Switch

# Lab 2 - Controle do Switch

\$ dpctl dump-flows tcp:15.200.124.107:6633

Fluxo Instalado, Regras, Timeout,Ações

Pacotes e bytes processados pelo fluxo (estatistica)

\$ dpctl dump-ports tcp:15.200.124.107:6633

Informações de Portas físicas

Contadores Rx,Tx

Erros nos contadores

\$ dpctl mod-port tcp:15.200.124.107:6633 17 down

manipular portas (Up, down, flood, noflood)

# Lab 2 - Controle do Switch

```
$ dpctl mod-port tcp:15.200.124.107:6633 17 down  
manipular portas (Up, down, flood, noflood)  
$ dpctl mod-port tcp:15.200.124.107:6633 2 down  
$ dpctl mod-port tcp:15.200.124.107:6633 2 up
```

```
$ dpctl add-flow tcp:15.200.124.107:6633  
in_port=10,actions=output:14  
$ dpctl add-flow tcp:15.200.124.107:6633  
in_port=14,actions=output:10  
$ dpctl add-flow tcp:15.200.124.107:6633  
ip,nw_dst=10.10.10.1,priority=1,actions=output:2  
$ dpctl add-flow tcp:15.200.124.107:6633  
ip,nw_dst=10.10.10.2,priority=2,actions=output:17
```

# Lab 2 - Controle do Switch

```
$ sudo mn --custom lab1.py --topo mytopo
```

# Lab 2 - Controle do Switch

## Set regras para S3 e S4

```
s3 dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2  
s3 dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1  
s4 dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2  
s4 dpctl add-flow tcp:127.0.0.1:6634 in_port=2,actions=output:1
```

# Lab 2 - Controle do Switch

**Log de comunicação**

```
mininet> h1 tcpdump -w /tmp/mylog &
```

**Teste de conectividade**

```
mininet> h1 ping -c 5 h2
```

# Lab 2 - Controle do Switch

```
mininet> s3 dpctl dump-flows tcp:127.0.0.1:6634  
mininet> s4 dpctl dump-flows tcp:127.0.0.1:6634
```

```
mininet> s3 dpctl dump-flows tcp:127.0.0.1:6634  
stats reply (xid=0xe77fe648): flags=none type=1(flow)  
  cookie=0, duration_sec=42s, duration_nsec=102000000s, table_id=0, priority=327  
68, n_packets=7, n_bytes=574, idle_timeout=60, hard_timeout=0, in_port=1, actions=o  
utput:2  
  cookie=0, duration_sec=37s, duration_nsec=330000000s, table_id=0, priority=327  
68, n_packets=8, n_bytes=675, idle_timeout=60, hard_timeout=0, in_port=2, actions=o  
utput:1  
mininet> s4 dpctl dump-flows tcp:127.0.0.1:6634  
stats reply (xid=0x59dc1869): flags=none type=1(flow)  
  cookie=0, duration_sec=56s, duration_nsec=218000000s, table_id=0, priority=327  
68, n_packets=7, n_bytes=574, idle_timeout=60, hard_timeout=0, in_port=1, actions=o  
utput:2  
  cookie=0, duration_sec=51s, duration_nsec=446000000s, table_id=0, priority=327  
68, n_packets=8, n_bytes=675, idle_timeout=60, hard_timeout=0, in_port=2, actions=o  
utput:1
```

# Lab 2 - Controle do Switch

**mininet> s3 dpctl show tcp:127.0.0.1:6634**

**Num. De Tabelas e buffer**

**Identificado Único OpenFlow**

```
mininet> s3 dpctl show tcp:127.0.0.1:6634
features_reply (xid=0x3755d761): ver:0x1, dpid:3
n_tables:255, n_buffers:256
features: capabilities:0xc7, actions:0xffff
  1(s3-eth1): addr:8e:0d:b3:a3:4b:83, config: 0, state:0
    current: 10GB-FD COPPER
  2(s3-eth2): addr:0a:00:0a:14:6f:67, config: 0, state:0
    current: 10GB-FD COPPER
  LOCAL(s3): addr:12:1e:2c:b9:59:4f, config: 0x1, state:0x1
get config reply (xid=0x4cbfa971): miss send len=0
```

**Portas**

# Lab 2 - Controle do Switch

**mininet> exit**

**wireshark /tmp/mylog**

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::38eb:10ff:fe78:5a9 ff02::fb		MDNS	101	Standard query PTR _san
2	0.056214	fe80::9cde:a0ff:fe52:459 ff02::fb		MDNS	101	Standard query PTR _san
3	31.992135	fe80::38eb:10ff:fe78:5a9 ff02::fb		MDNS	101	Standard query PTR _san
4	32.052820	fe80::9cde:a0ff:fe52:459 ff02::fb		MDNS	101	Standard query PTR _san
5	80.496689	be:a8:05:42:48:5d	Broadcast	ARP	42	Who has 10.0.0.2? Tell
6	80.500386	d2:68:ea:f9:93:fd	be:a8:05:42:48:5d	ARP	42	10.0.0.2 is at d2:68:ea
7	80.500548	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id
8	80.504051	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id
9	81.497144	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id
10	81.497927	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id
11	82.498822	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id
12	82.499066	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id
13	83.497789	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id
14	83.497874	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id
15	84.497068	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id
16	84.497185	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id
17	85.519119	d2:68:ea:f9:93:fd	be:a8:05:42:48:5d	ARP	42	Who has 10.0.0.1? Tell
18	85.519158	be:a8:05:42:48:5d	d2:68:ea:f9:93:fd	ARP	42	10.0.0.1 is at be:a8:05

# Lab 2 - Controle do Switch (Again)

```
$ sudo mn --custom lab1.py --topo mytopo
```

# Lab 2 - Controle do Switch (Again)

## Set regras para S3

```
s3 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=1,in_port=1,actions=output:2
```

```
s3 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=1,in_port=2,actions=output:1
```

```
s3 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=10,ip,nw_dst=10.0.0.1,actions=output:1
```

```
s3 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=10,ip,nw_dst=10.0.0.2,actions=output:2
```

```
s3 dpctl dump-flows tcp:127.0.0.1:6634
```

# Lab 2 - Controle do Switch (Again)

## Set regras para S4

```
s4 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=1,in_port=1,actions=output:2
```

```
s4 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=1,in_port=2,actions=output:1
```

```
s4 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=10,ip,nw_dst=10.0.0.1,actions=output:1
```

```
s4 dpctl add-flow tcp:127.0.0.1:6634 idle_timeout=0,hard_timeout=0,  
priority=10,ip,nw_dst=10.0.0.2,actions=output:2
```

```
s4 dpctl dump-flows tcp:127.0.0.1:6634
```

# Lab 2 - Controle do Switch (Again)

## Teste de conectividade

**mininet>h1 ping -c 5 h2**

**mininet>s3 dpctl dump-flows tcp:127.0.0.1:6634**

arp

Ping (echo)

Ping (reply)

```
mininet> s3 dpctl dump-flows tcp:127.0.0.1:6634
stats reply (xid=0x4c09a44a): flags=none type=1(flow)
  cookie=0, duration_sec=92s, duration_nsec=685000000s, table_id=0, priority=1,
  n_packets=1, n_bytes=84, idle_timeout=0, hard_timeout=0, in_port=1, actions=output:
  2
  cookie=0, duration_sec=82s, duration_nsec=805000000s, table_id=0, priority=1,
  n_packets=3, n_bytes=185, idle_timeout=0, hard_timeout=0, in_port=2, actions=output
  :1
  cookie=0, duration_sec=74s, duration_nsec=820000000s, table_id=0, priority=10,
  n_packets=5, n_bytes=490, idle_timeout=0, hard_timeout=0, ip,nw_dst=10.0.0.1,acti
  ons=output:1
  cookie=0, duration_sec=65s, duration_nsec=202000000s, table_id=0, priority=10,
  n_packets=5, n_bytes=490, idle_timeout=0, hard_timeout=0, ip,nw_dst=10.0.0.2,acti
  ons=output:2
```

# Lab 2 - Controle do Switch (Again)

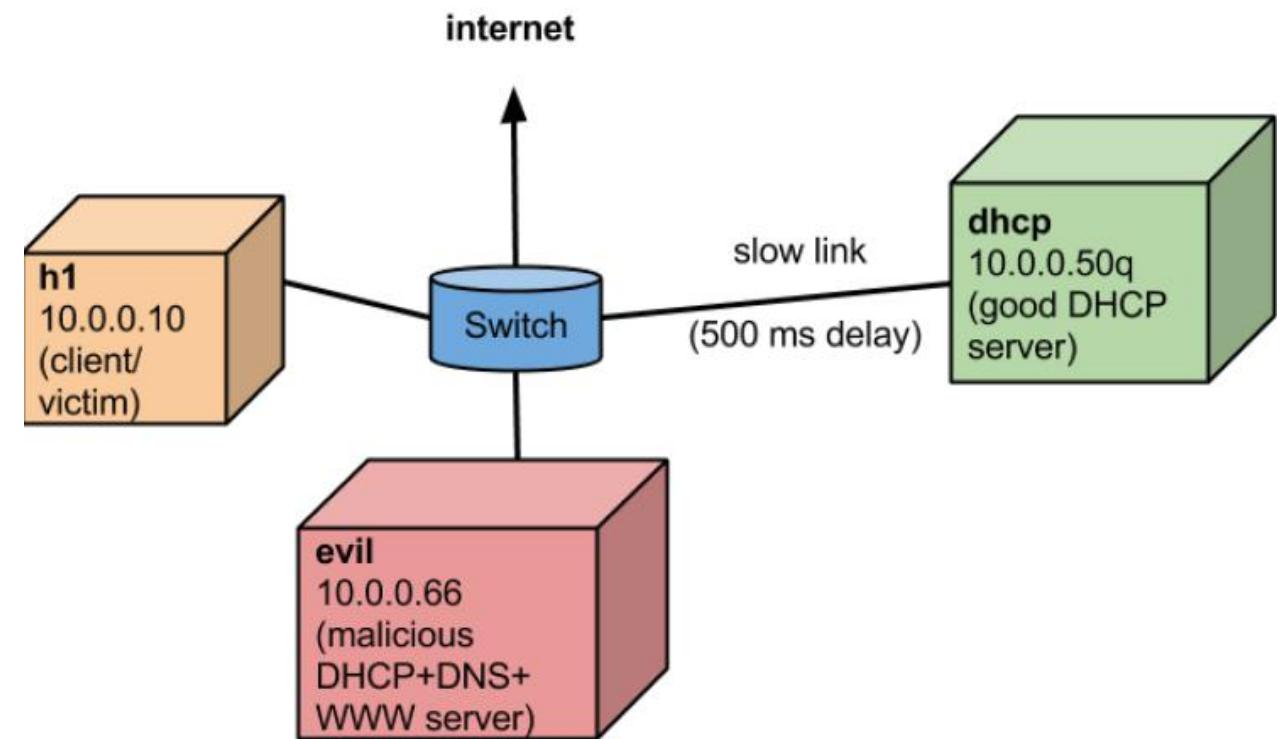
**mininet>s4 dpctl dump-flows tcp:127.0.0.1:6634**

```
mininet> s4 dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x94ca73e6): flags=none type=1(flow)
  cookie=0, duration sec=145s, duration nsec=3000000s, table id=0, priority=1, n_packets=2, n_bytes=84, idle_timeout=0,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration sec=135s, duration nsec=123000000s, table id=0, priority=1, n_packets=3, n_bytes=185, idle_timeout=0,hard_timeout=0,in_port=2,actions=output:1
  cookie=0, duration sec=127s, duration nsec=138000000s, table id=0, priority=10, n_packets=5, n_bytes=490, idle_timeout=0,hard_timeout=0,ip,nw_dst=10.0.0.1,actions=output:1
  cookie=0, duration sec=117s, duration nsec=520000000s, table id=0, priority=10, n_packets=5, n_bytes=490, idle_timeout=0,hard_timeout=0,ip,nw_dst=10.0.0.2,actions=output:2
```

# Lab 3 - Dhcp masquerade attack

Nesta demonstração, construímos uma rede simples para demonstrar um ataque de DHCP. Este é um exemplo do uso Mininet para executar uma experiência que você (geralmente) não gostaria de tentar em uma rede regular que as pessoas estão tentando usar!

# Lab 3 - Dhcp masquerade attack



h1 (10.0.0.10) é a "vítima" ligada a rede e vai fazer solicitações DHCP.

dhcp (10.0.0.53) é o "bom" servidor DHCP que fornece informações corretas, mas que está ligado ao comutador por uma ligação lenta (500 ms de atraso neste exemplo).

Evil (10.0.0.66) é um host malicioso que é conectado diretamente ao switch, que fornece respostas DHCP maliciosas e também abriga um servidor DNS malicioso e um servidor DHCP mal-intencionado.

# Lab 3 - Dhcp masquerade attack

Quando o h1 faz a solicitação DHCP, ele é encaminhado para ambos e dhcp o mal responde primeiro é aceito por h1. o Servidor DHCP Evil fornece o endereço como o endereço do servidor DNS, de modo a vítima envia suas Solicitações de DNS para o servidor DNS forjado, que fornece o seu próprio endereço IP em vez do endereço IP correto para a pesquisa de DNS. A vítima em seguida liga-se o endereço do servidor web mal, que serve o seu próprio conteúdo malicioso, pedindo a vítima para um e-mail endereço e senha.

# Lab 3 - Dhcp masquerade attack

```
sudo apt-get install Firefox dnsmasq
```

```
git clone https://bitbucket.org/lantz/cs144-dhcp  
cd cs144-dhcp  
sudo -E ./dhcp.py
```

Em h1 rodar wireshark e filtrar por bootp tráfego em h1-eth0

# Lab 3 - Dhcp masquerade attack

```
# Acesse outro endereço e veja a captura  
# pode-se utilizar o dig para validar a busca  
  
# na janela do Mininet pressione return para acionar o  
dhcp malicioso  
# faça a navegação novamente  
# veja os resultados no wireshark.  
# pressione Enter para encerrar e saia do mininet
```

# Laboratórios - SDN

## **Lab 1: Criar uma rede e executar um teste de performance**

```
chmod +x mymininet1.py
sudo python mymininet1.py
```

# Laboratórios - SDN

**Lab 2: Criar uma rede e usar o controlador POX para o comportamento do switch**

```
chmod +x mymininet2.py
sudo ./mymininet2.py
```

# Laboratórios - SDN

## **Lab 3: usar o comando ovs-vsctl para controlar o open vswitch**

### **mymininet4.py**

um switch e dois hosts. Quando há tráfego que vem na porta 1, encaminhá-lo à porta 2, e vice-versa.

**mymininet4\_1.py:** 2 switches e 2 hosts. Host 0 é conectado ao Switch 0 enquanto o Host 1 é conectado ao Switch 1. E o switch 0 e switch 1 estão conectados

**mymininet4\_3.py:** Parecido com mymininet4\_1.py. A principal diferença é que ao criar links, vamos definir os parâmetros de largura de banda, atraso e taxa de perda

# Laboratórios - SDN

## Lab 4: “ovs-vsctl” avançado

**mymminet6.py:** Regra padrão de flooding para o switch.  
Setar regras de alta prioridade para realizar o forward de pacotes

**mymminet6\_1.py:** TOS com diferentes valores seguem por caminhos diferentes.

# Laboratórios - SDN

## **Lab 5: Alterar dinamicamente os parâmetros de rede**

**myminet7.py:** 5 segundos após executar o script, mudamos o delay do link

# Laboratórios - SDN

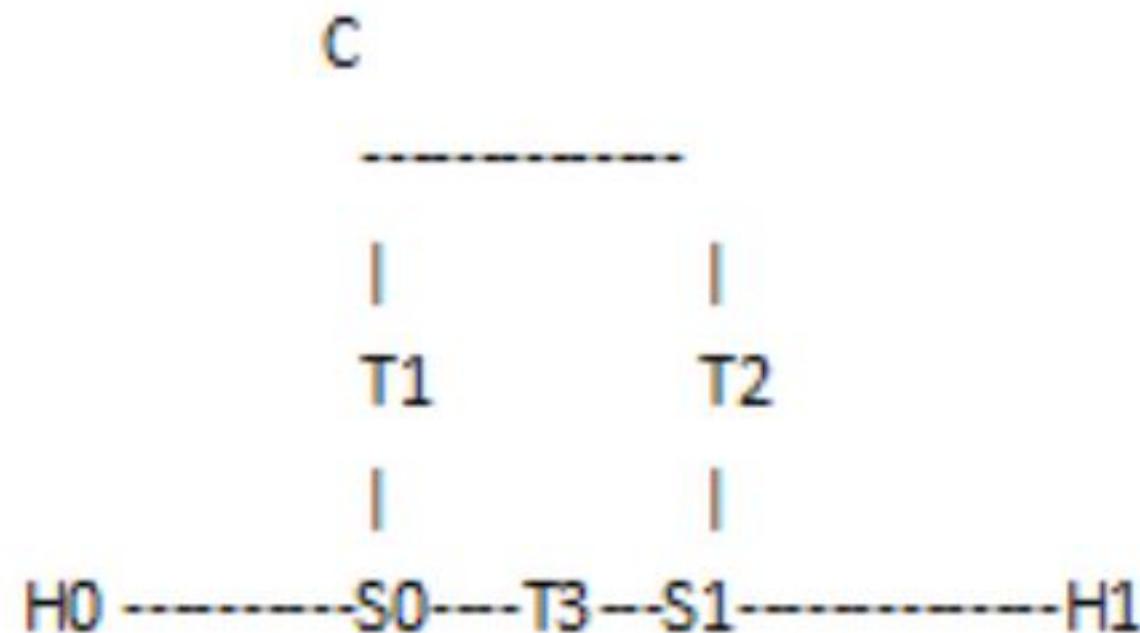
## **Lab 7: Medir Perda de Pacotes**

mininet10.py

# Laboratórios - SDN

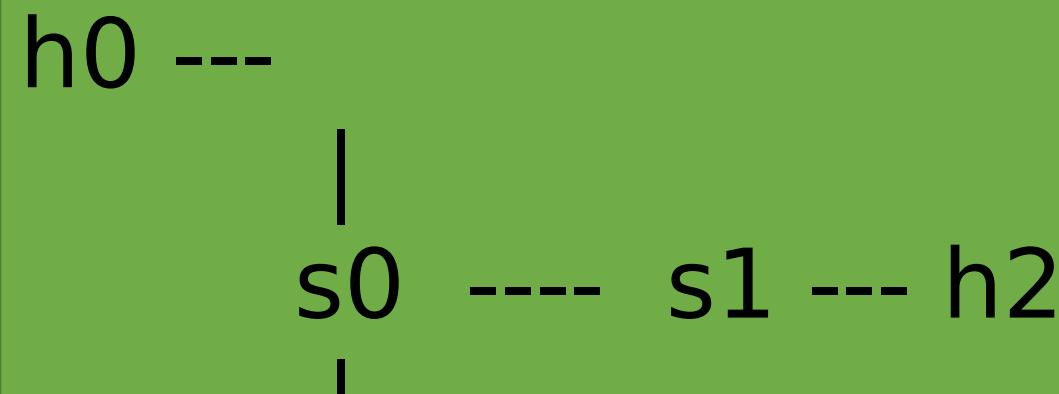
## Lab 8: Medir Latencia

mininet11.py



# Laboratórios - SDN

## Lab 9: Limitando a Banda



h1 ----

mininet12.py

# SDN



# SDN

- SDN - Tecnologia do Momento;
- Ataque de superfície – Confundir as redes tradicionais nos Planos de Dados e Controles. Separar Data Plane do Control Plane
- Vulnerabilidades
  - 1) XML eXternal Entity (XXE) vulnerability in OpenDaylight netconf (CVE20145035)
  - [https://wiki.opendaylight.org/view/Security\\_Advisories#.5BImportant.5D\\_CVE-2014-5035\\_netconf:\\_XML\\_eXternal\\_Entity\\_.28XXE.29\\_vulnerability](https://wiki.opendaylight.org/view/Security_Advisories#.5BImportant.5D_CVE-2014-5035_netconf:_XML_eXternal_Entity_.28XXE.29_vulnerability)
  - 2) Denial of service (DoS) when deserializing malformed packets in ONOS (CVE20151166)
  - <https://jira.onosproject.org/browse/ONOS-605>
  - 3) Topology spoofing via host tracking
  - [http://www.internetsociety.org/sites/default/files/10\\_4\\_2.pdf](http://www.internetsociety.org/sites/default/files/10_4_2.pdf)