

# Emulando un datacenter

**Introducción a los Sistemas Distribuidos (75.43)**

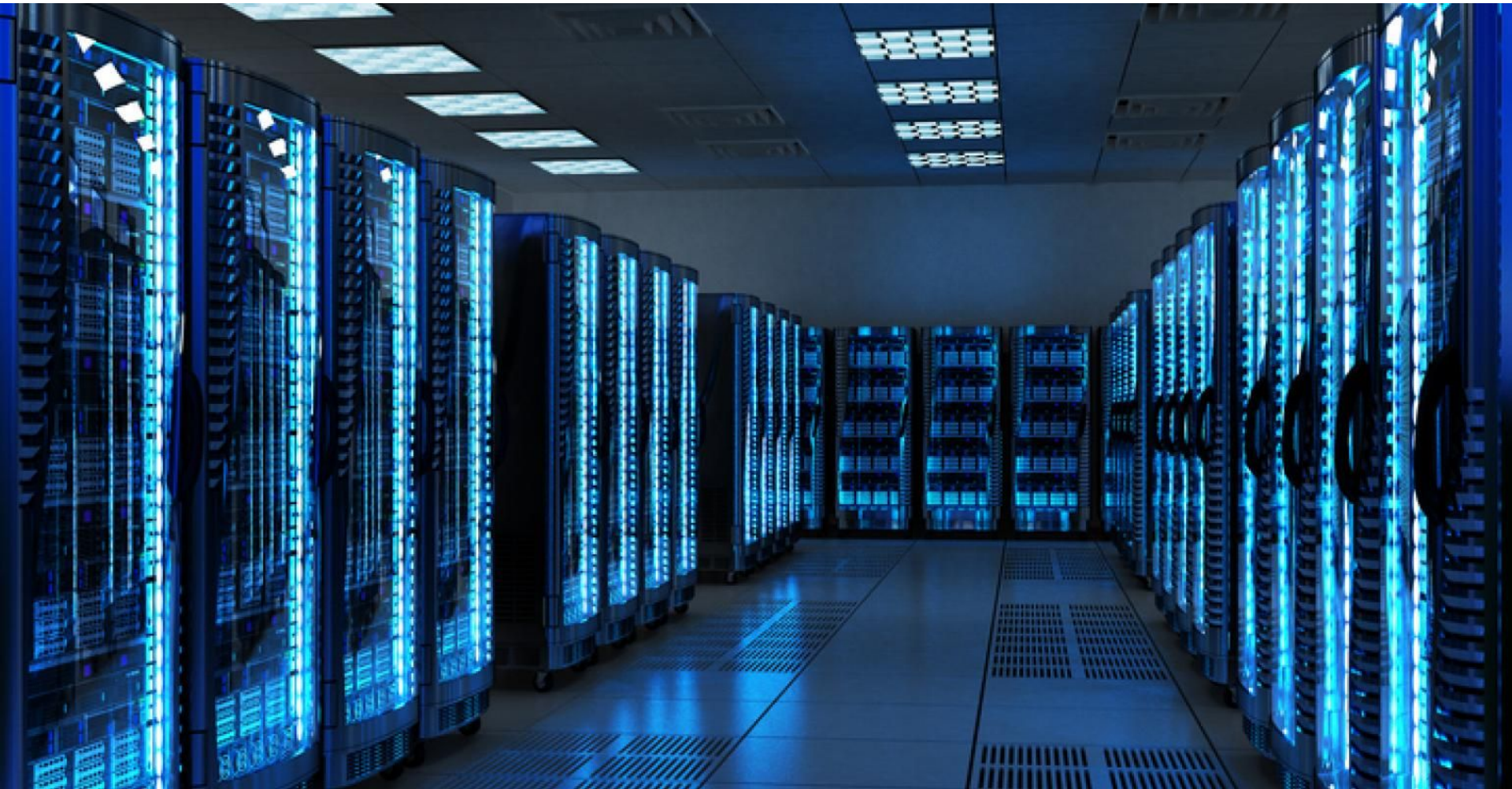
**Universidad de Buenos Aires, Facultad de Ingeniería**

Julio, 2020



# Datacenter

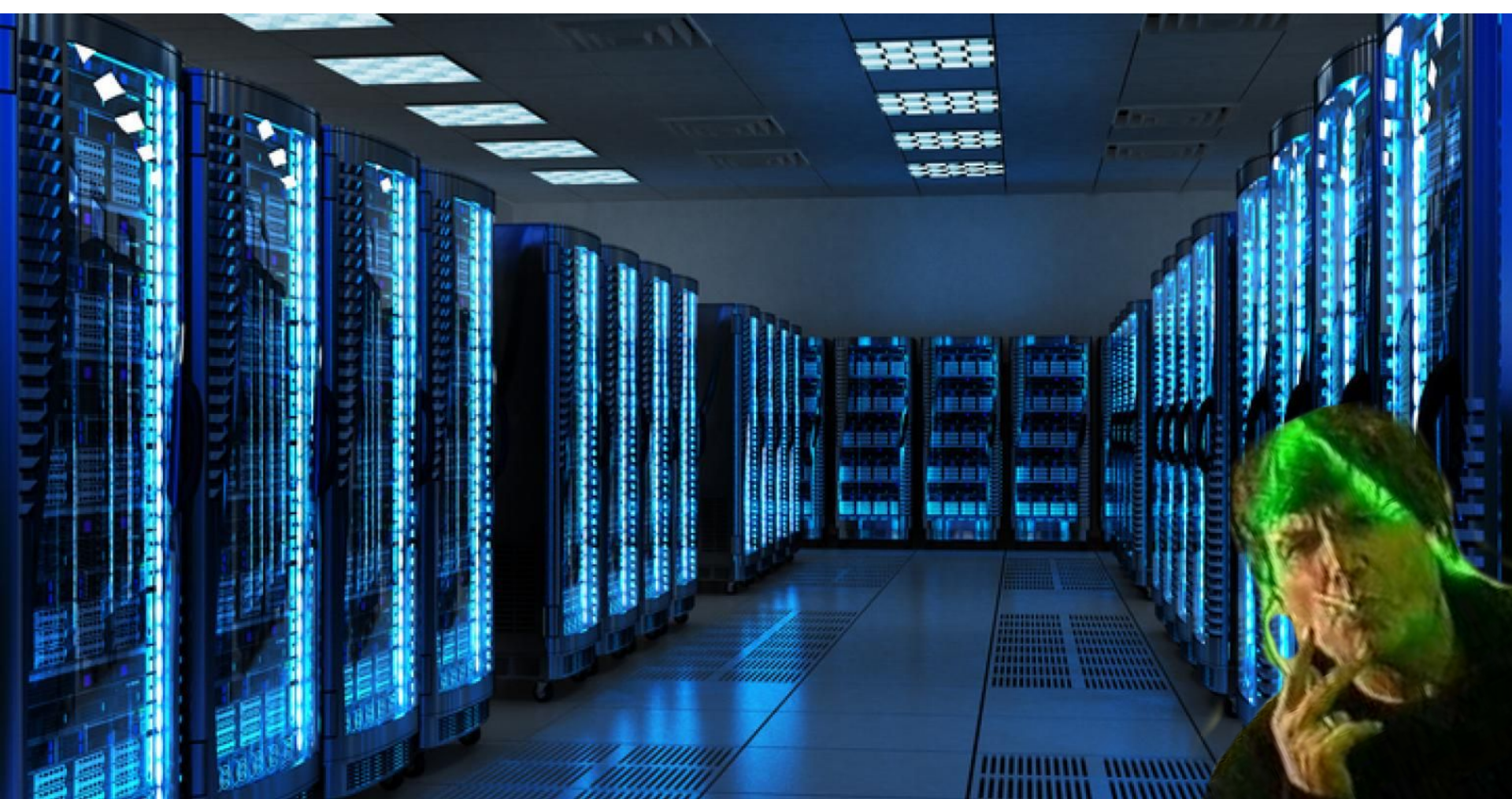
- Cluster de servidores





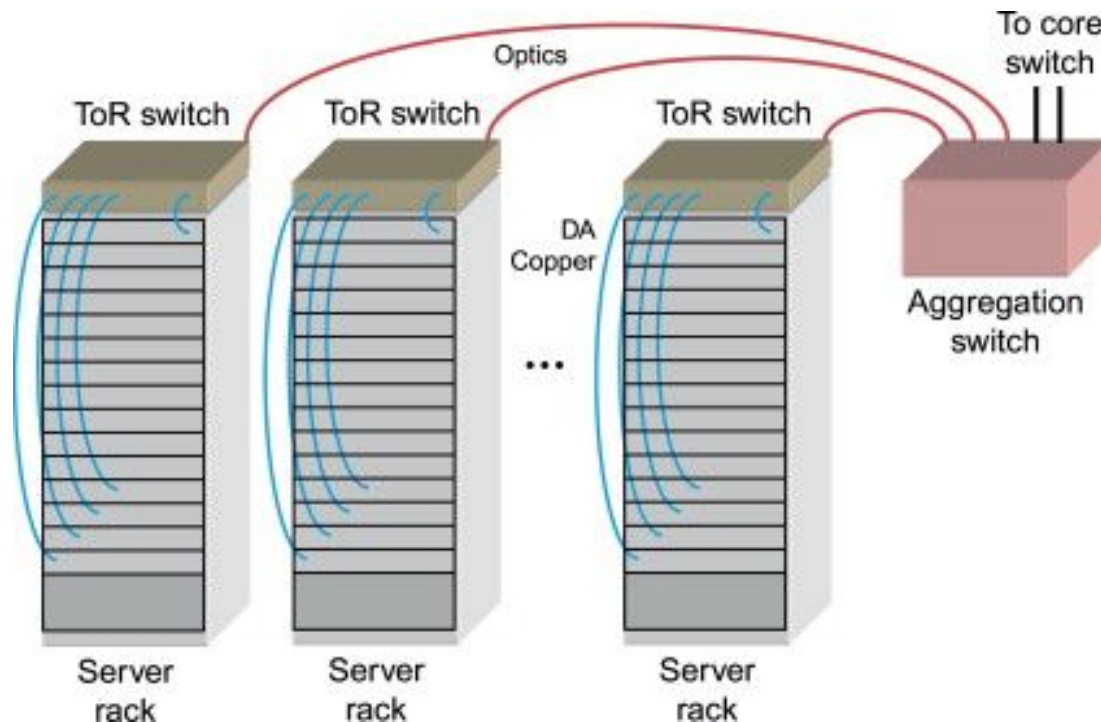
# Datacenter

- Cluster de servidores



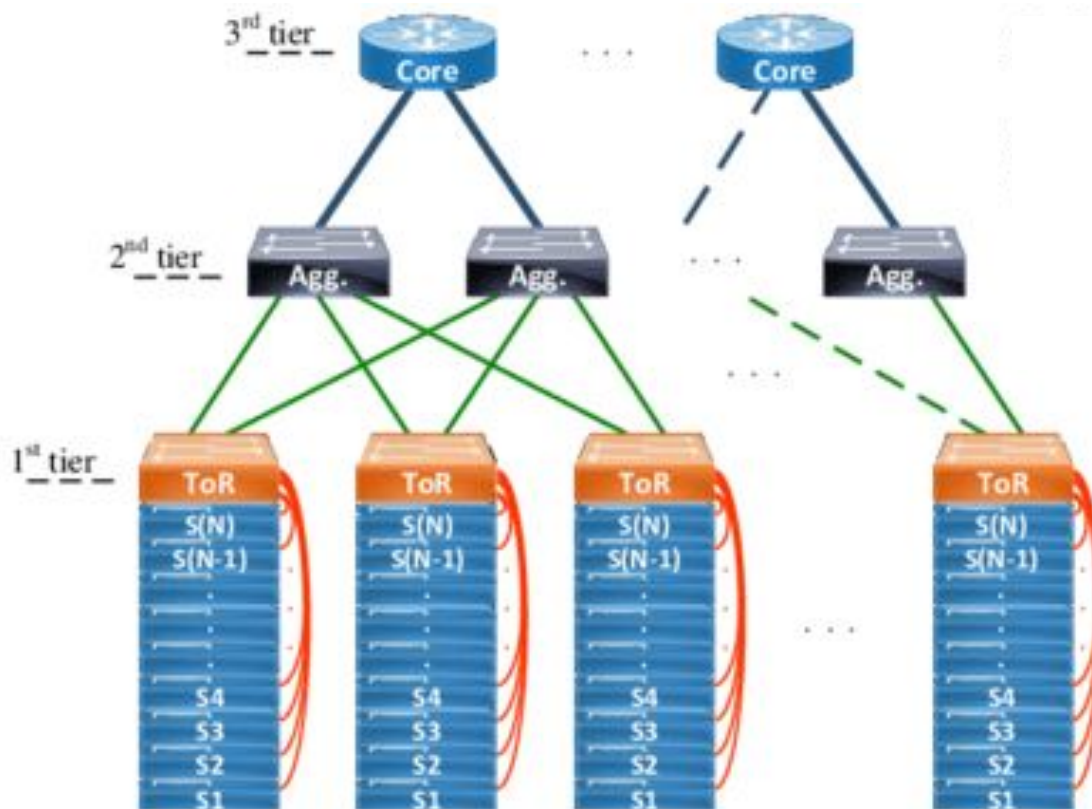
# Datacenter - arquitectura

- Cada servidor se conecta a un switch
- Que se conectan al TOR switch
- Que se conectan a un switch agregador



# Datacenter - arquitectura fattree

- Cada servidor se conecta a un switch
- Que se conectan al TOR switch
- Que se conectan a un switch agregador



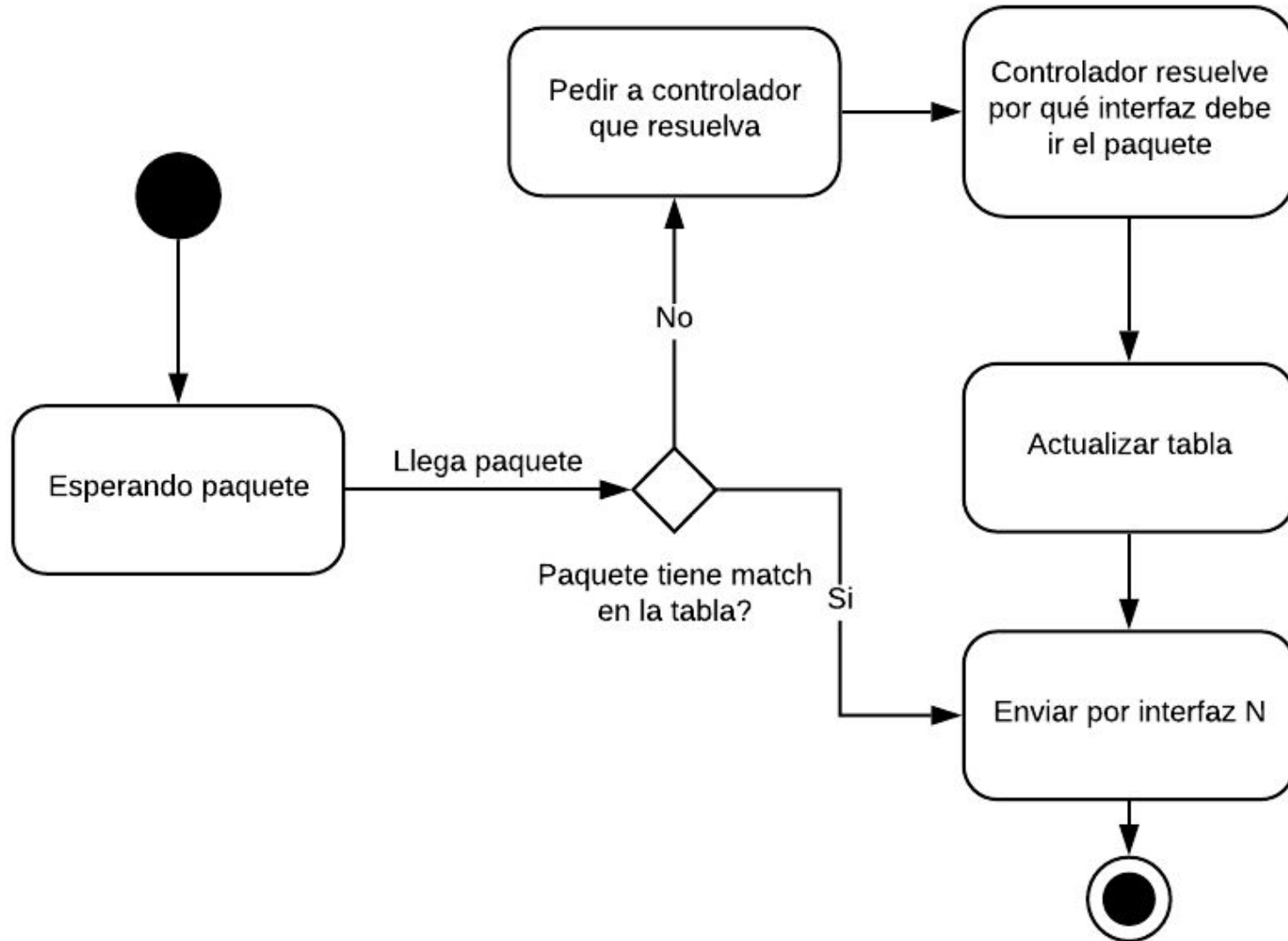
# SDNs

- Los routers y switches clásicos dependen del fabricante
- -> inflexibilidad. Es difícil adaptarse a cambios
- ¿Por qué no pensar en un switch o router programable?
- -> Software Defined Networking (SDN)

# OpenFlow

- Protocolo para interactuar con las tablas de CAM / routeo de los sw / routers
- Veamos un ejemplo de como funciona

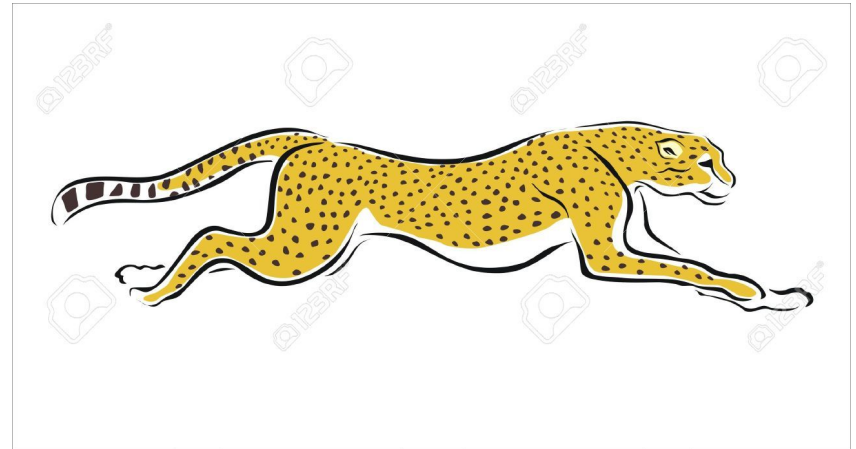
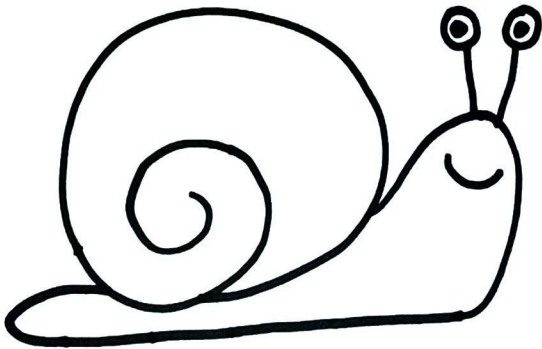
# Estados de un SW OpenFlow



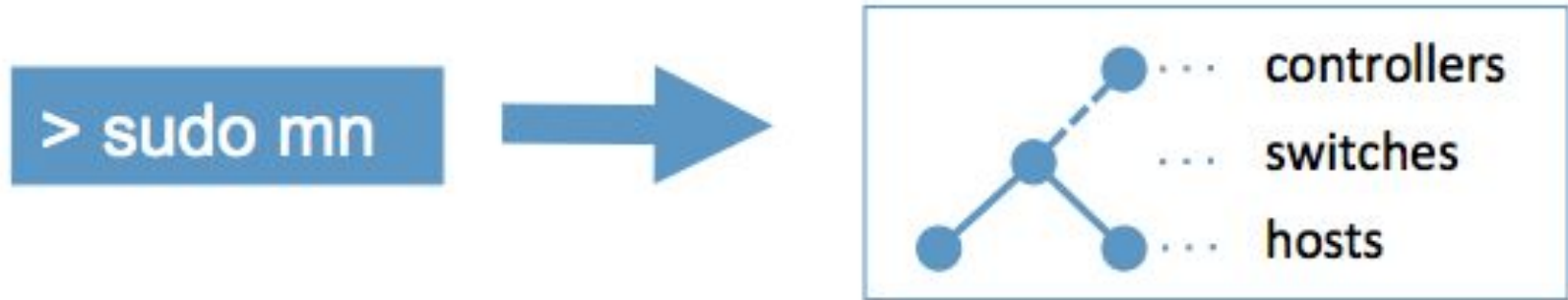


# OpenFlow

- Importante:
  - Actualizar las tablas luego de que el controlador resuelva
  - Las memorias de las tablas son muy rápidas
  - El controlador, no tanto



# Mininet



Crea una red virtual, ejecutando kernel, switch y código de aplicación, en una sola máquina.  
Se utiliza para probar, desarrollar, y experimentar con OpenFlow y SDN.<sup>(1)</sup>

(1) <http://mininet.org/>

# Mininet - Comandos

- **Lanzar mininet (CLI):**  
`$ sudo mn`
- **Listar nodos:**  
`mininet> nodes`
- **Listar enlaces:**  
`mininet> links`
- **Listar información de los nodos:**  
`mininet> dump`
- **Ejecutar comando en un device:**  
`mininet> dev1 command`  
`mininet> h1 ifconfig -a`
- **Probar conectividad entre hosts:**  
`mininet> h1 ping h2`
- **Abrir una xterm:**  
`mininet> h1 xterm`
- **Ping a todos:**  
`mininet> pingall`

# Mininet - Ejemplo

Lanzar una instancia de mininet, sin parámetros. Analizar la topología y probar la conectividad entre dispositivos.

*\$ sudo mn*

Por defecto, se crea una topología con un **switch** y dos **hosts**.

```
➔ ~ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

# Mininet - Ejemplo

```
mininet> links
```

```
h1-eth0<->s1-eth1 (OK OK)
```

```
h2-eth0<->s1-eth2 (OK OK)
```

```
mininet> dump
```

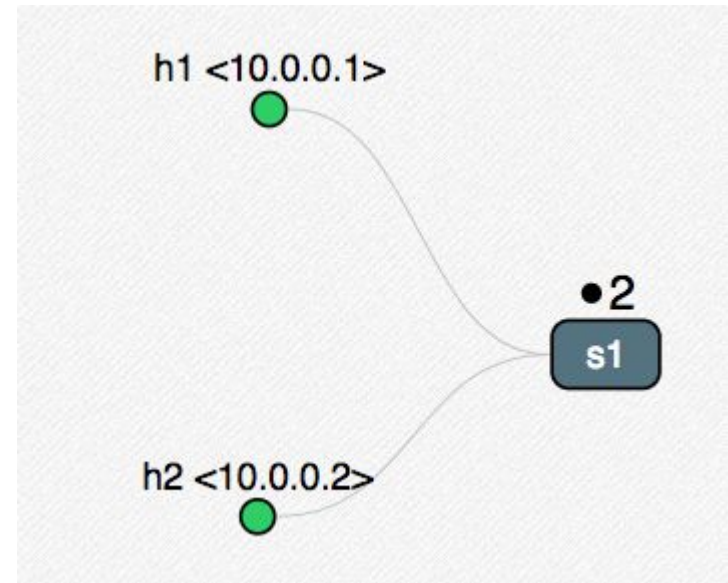
```
<Host h1: h1-eth0:10.0.0.1  
pid=1907>
```

```
<Host h2: h2-eth0:10.0.0.2  
pid=1909>
```

```
<OVSSwitch s1:
```

```
lo:127.0.0.1,s1-eth1:None,s1-eth2  
:None pid=1914>
```

```
<Controller c0: 127.0.0.1:6653  
pid=1900>
```



¿Qué es esto?



# Mininet - Ejemplo

Probar la conectividad entre los dispositivos

```
mininet> pingall
```

```
*** Ping: testing ping  
reachability
```

```
h1 -> h2
```

```
h2 -> h1
```

```
*** Results: 0% dropped (2/2  
received)
```

Otra forma

```
➔ ~ sudo mn --test pingall
```

El comando pingall se utiliza para probar la conectividad entre todos los dispositivos, se ejecuta el comando ping en todos los dispositivos de la topología

```
> dev1 ping -c1 dev2
```

# Mininet - Ejemplo

The image shows a Wireshark network traffic capture window titled '\*s1-eth1'. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar with various icons. The packet list pane shows four ICMP packets:

No.	Time	Source	Destination	Protocol	Length	Info
12	2.286346933	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0cc4, seq=1/256, ttl=64 (reply in 13)
13	2.286575779	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0cc4, seq=1/256, ttl=64 (request in 12)
14	2.288168716	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x0cc5, seq=1/256, ttl=64 (reply in 15)
15	2.288175647	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0cc5, seq=1/256, ttl=64 (request in 14)

The packet details pane for packet 12 shows the following structure:

- Frame 12: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
- Ethernet II, Src: 5e:92:4c:be:54:c2 (5e:92:4c:be:54:c2), Dst: 92:81:00:6d:c1:6d (92:81:00:6d:c1:6d)
- Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
- Internet Control Message Protocol

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000  92 81 00 6d c1 6d 5e 92  4c be 54 c2 08 00 45 00  ...m.m^..L.T...E.
0010  00 54 df 86 40 00 40 01  47 20 0a 00 00 01 0a 00  .T..@.@. G .....
0020  00 02 08 00 16 e2 0c c4  00 01 82 1f 04 5b 00 00  .....[...
0030  00 00 81 0b 0e 00 00 00  00 00 10 11 12 13 14 15  .....
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25  ..... !"#$$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35  &'()*+,-./012345
```

The status bar at the bottom indicates: Internet Control Message Protocol: Protocol, Packets: 61 · Displayed: 4 (6.6%), Profile: Default.

Mininet  
Introducción a los Sistemas Distribuidos (75.43)

# Mininet - Topologías más complejas

```
➔ ~ sudo mn --topo linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2,
s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Topología “linear”, recibe por parámetro la cantidad de switches, y crea una topología donde cada switch tiene un host conectado, y todos los switches están conectados en forma lineal.

# Mininet - Topologías más complejas

➔ ~ sudo mn --topo linear,4

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

**h1 h2 h3 h4**

\*\*\* Adding switches:

**s1 s2 s3 s4**

\*\*\* Adding links:

**(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)**

\*\*\* Configuring hosts

**h1 h2 h3 h4**

\*\*\* Starting controller

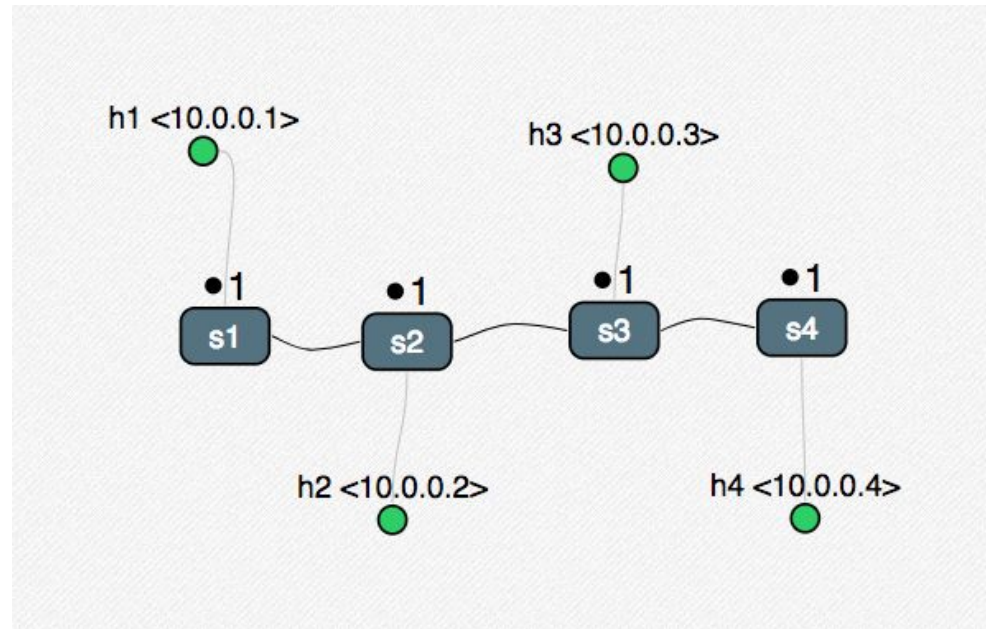
**c0**

\*\*\* Starting 4 switches

**s1 s2 s3 s4 ...**

\*\*\* Starting CLI:

**mininet>**



# Mininet - Topologías más complejas

➔ ~ sudo mn --topo tree,4

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

**h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12**

**h13 h14 h15 h16**

\*\*\* Adding switches:

**s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14  
s15**

\*\*\* Adding links:

**(s1, s2) (s1, s9) (s2, s3) (s2, s6) (s3, s4) (s3,  
s5) (s4, h1) (s4, h2) (s5, h3) (s5, h4) (s6, s7)  
(s6, s8) (s7, h5) (s7, h6) (s8, h7) (s8, h8) (s9,  
s10) (s9, s13) (s10, s11) (s10, s12) (s11, h9)  
(s11, h10) (s12, h11) (s12, h12) (s13, s14)  
(s13, s15) (s14, h13) (s14, h14) (s15, h15)  
(s15, h16)**

\*\*\* Configuring hosts

**h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12**

**h13 h14 h15 h16**

\*\*\* Starting controller

**c0**

\*\*\* Starting 15 switches

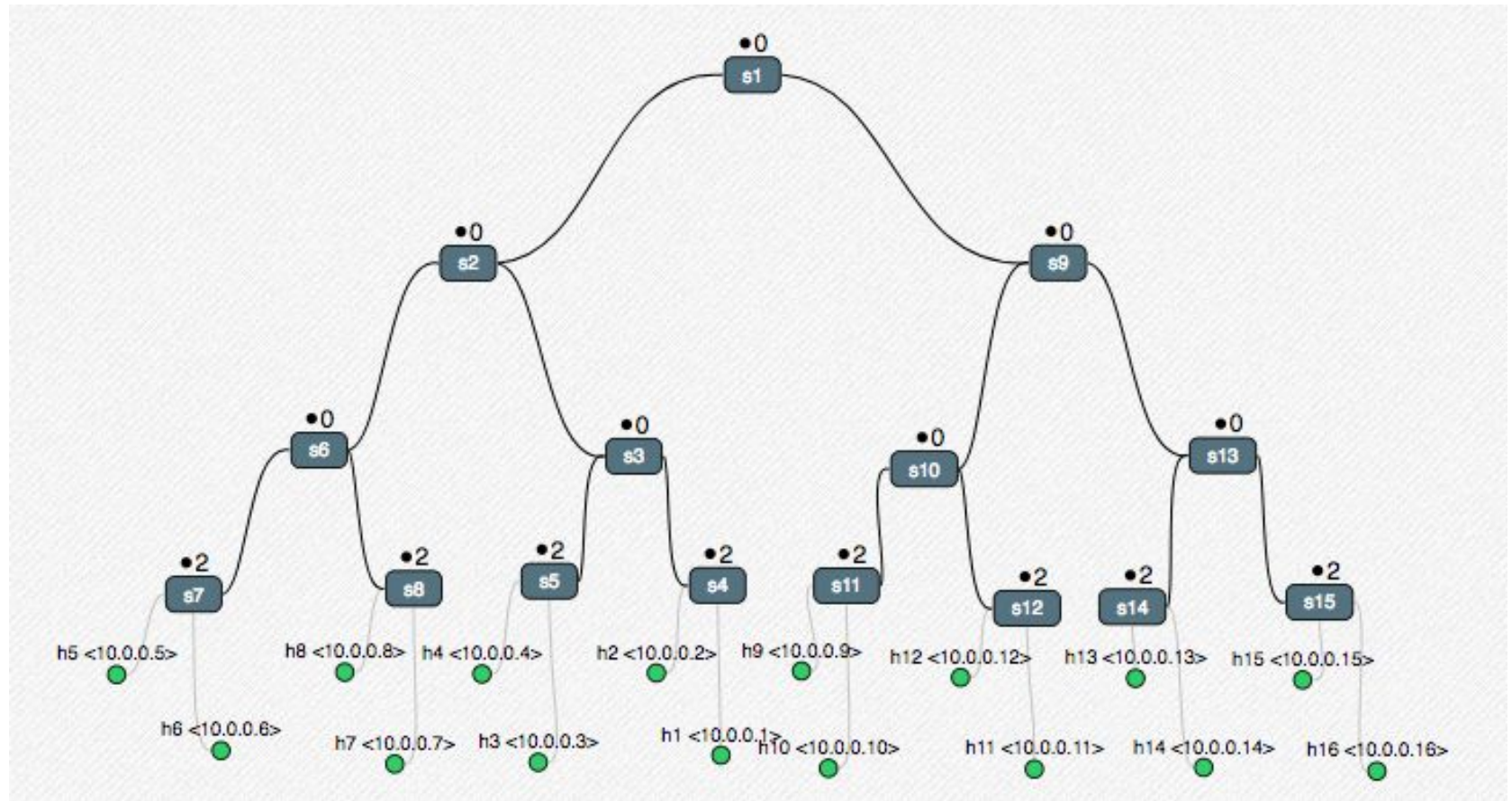
**s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13  
s14 s15 ...**

\*\*\* Starting CLI:

**mininet>**



# Mininet - Topologías más complejas



# Mininet - Topologías personalizadas

Mininet provee las siguientes topologías:

- linear
- minimal
- reversed
- single
- torus
- tree

Sin embargo, también se pueden crear topologías personalizadas usando la API Python que nos provee la herramienta.

# Mininet - Topologías personalizadas

```
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() ) }
```

custom/topo-2sw-2host.py

Mininet  
Introducción a los Sistemas Distribuidos (75.43)

# Mininet - Topologías custom

➔ ~ sudo mn --custom /tmp/topology/example.py --topo example --mac --switch ovsk

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

sw1 sw2

\*\*\* Adding links:

(h1, sw1) (sw1, sw2) (sw2, h2)

\*\*\* Configuring hosts

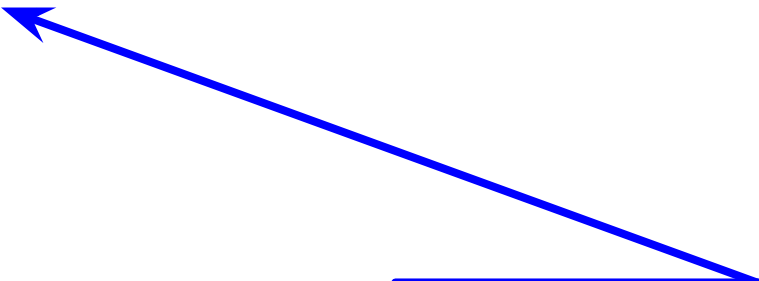
h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 2 switches

sw1 sw2 ...



--custom file.py:  
Ubicación del archivo  
donde está definida la  
topología

# Mininet - Topologías custom

➔ ~ sudo mn --custom /tmp/topology/example.py --**topo example** --mac --switch  
ovsk

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

sw1 sw2

\*\*\* Adding links:

(h1, sw1) (sw1, sw2) (sw2, h2)

\*\*\* Configuring hosts


h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 2 switches

sw1 sw2 ...



--topo nombre: Nombre de  
la topología  
topos = { example: ( lambda:  
ExampleTopology() ) }



# Mininet - Topologías custom

➔ ~ sudo mn --custom /tmp/topology/example.py --topo example --mac

--switch ovsk

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

sw1 sw2

\*\*\* Adding links:

(h1, sw1) (sw1, sw2) (sw2, h2)

\*\*\* Configuring hosts

h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 2 switches

sw1 sw2 ...

--mac:

Se configuran direcciones MAC sencillas  
para un usuario

Sin --mac:

HWaddr 9a:e9:bf:6c:86:95

Con --mac:

HWaddr 00:00:00:00:00:01

# Mininet - Topologías custom

➔ ~ sudo mn --custom /tmp/topology/example.py --topo example --mac --switch ovsk

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

sw1 sw2

\*\*\* Adding links:

(h1, sw1) (sw1, sw2) (sw2, h2)

\*\*\* Configuring hosts

h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 2 switches

sw1 sw2 ...



--switch ovsk:  
Se utiliza un tipo de switch  
“Open vSwitch”<sup>(2)</sup>

(2) <http://www.openvswitch.org/>

Mininet  
Introducción a los Sistemas Distribuidos  
(75.43)\

# Mininet - iperf

## ➔ ~ man iperf

Es una herramienta para realizar mediciones de rendimiento de la red. Puede probar el rendimiento TCP o UDP. Para realizar una prueba iperf, el usuario debe establecer tanto un servidor (para descartar tráfico) como un cliente (para generar tráfico).

# Mininet - iperf

## Modo servidor:

➔ ~ sudo iperf -s -p 80

---

**Server** listening on TCP port 80  
TCP window size: 85.3 KByte  
(default)

---

## Modo cliente:

➔ ~ sudo iperf -c 10.0.0.1 -p 80

---

**Client** connecting to localhost, TCP port 80  
TCP window size: 2.50 MByte (default)

---

[ 3] local 127.0.0.1 port 48512 connected  
with 127.0.0.1 port 80

[ ID]	Interval	Transfer	Bandwidth
[ 3]	0.0-10.0 sec	56.9 GBytes	48.9 Gbits/sec

# Mininet - iperf

## Desde la consola de Mininet

```
mininet> iperf h1 h2
```

```
*** Iperf: testing TCP bandwidth between h1  
and h2
```

```
*** Results: ['24.9 Gbits/sec', '25.0 Gbits/sec']
```



# Mininet - Controlador remoto

```
from mininet.topo import Topo
```

```
class MyTopo( Topo ):
```

```
    def __init__( self ):
```

```
        # Initialize topology
```

```
        Topo.__init__( self )
```

```
        # Add hosts and switches
```

```
        leftHost = self.addHost( 'h1' )
```

```
        rightHost = self.addHost( 'h2' )
```

```
        leftSwitch = self.addSwitch( 's1' )
```

```
        rightSwitch = self.addSwitch( 's2' )
```

```
        mid1 = self.addSwitch('s3')
```

```
        mid2 = self.addSwitch('s4')
```

```
        # Add links
```

```
        self.addLink( leftHost, leftSwitch )
```

```
        self.addLink( rightHost, rightSwitch )
```

```
        self.addLink( leftSwitch, mid1 )
```

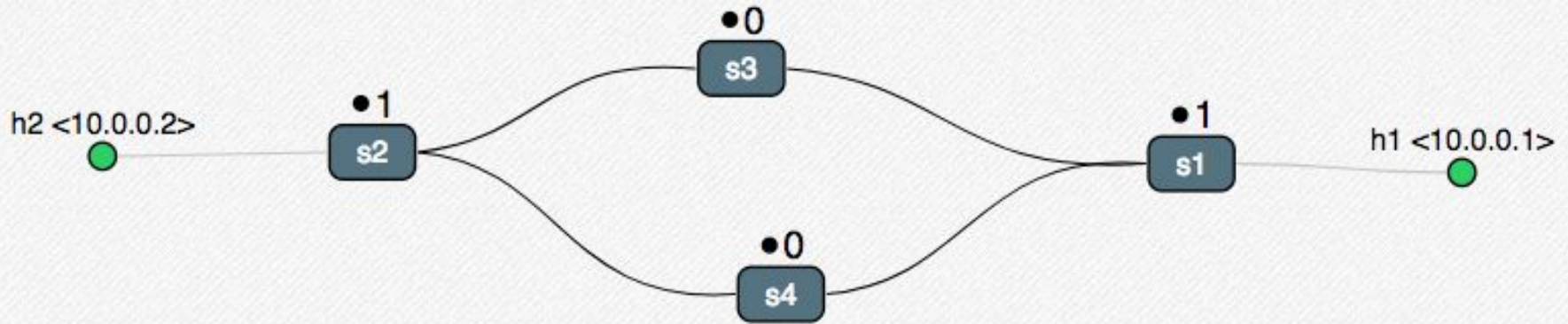
```
        self.addLink( leftSwitch, mid2 )
```

```
        self.addLink( rightSwitch, mid1 )
```

```
        self.addLink( rightSwitch, mid2 )
```

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

# Mininet - Controlador remoto



Mininet  
Introducción a los Sistemas Distribuidos (75.43)

# Mininet - Controlador remoto

```
➔ ~ sudo mn --custom  
~/mininet/custom/loop.py --topo mytopo  
--arp --mac --switch ovsk --test pingall
```

```
*** Creating network
```

```
*** Adding controller
```

```
*** Adding hosts:
```

```
h1 h2
```

```
*** Adding switches:
```

```
s1 s2 s3 s4
```

```
*** Adding links:
```

```
(h1, s1) (h2, s2) (s1, s3) (s1, s4) (s2, s3)
```

```
(s2, s4)
```

```
*** Configuring hosts
```

```
h1 h2
```

```
*** Starting controller
```

```
c0
```

```
*** Starting 4 switches
```

```
s1 s2 s3 s4 ...
```

```
*** Waiting for switches to connect
```

```
s1 s2 s3 s4
```

```
*** Ping: testing ping reachability
```

```
h1 -> X
```

```
h2 -> X
```

```
*** Results: 100% dropped (0/2  
received)
```

```
*** Stopping 1 controllers
```

```
c0
```

```
*** Stopping 6 links
```

```
.....
```

```
*** Stopping 4 switches
```

```
s1 s2 s3 s4
```

```
*** Stopping 2 hosts
```

```
h1 h2
```


```
*** Done
```

```
completed in 26.096 seconds
```

# Mininet - POX

POX es una plataforma para desarrollar controladores SDN con fines académicos y de investigación.<sup>(3)</sup>

Mininet se conecta al controlador mediante la dirección y puerto indicados (por defecto 127.0.0.1:6653). Por lo que debemos correr el controlador escuchando en dicho socket.



## Volviendo al primer ejemplo

```
mininet> links
```

```
h1-eth0<->s1-eth1 (OK OK)
```

```
h2-eth0<->s1-eth2 (OK OK)
```

```
mininet> dump
```

```
<Host h1: h1-eth0:10.0.0.1 pid=1907>
```

```
<Host h2: h2-eth0:10.0.0.2 pid=1909>
```

```
<OVSSwitch s1:
```

```
  lo:127.0.0.1,s1-eth1:None,s1-eth2:None  
  pid=1914>
```

```
<Controller c0: 127.0.0.1:6653 pid=1900>
```

(3) [https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch#Controller\\_Choice\\_POX\\_Python](https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch#Controller_Choice_POX_Python)

# Mininet - Controlador remoto

➔ ~ pox/pox.py samples.spanning\_tree  
POX 0.1.0 (beta) / Copyright 2011-2013 James  
McCauley, et al.

```
[core ] POX 0.1.0 (beta) is up.  
[openflow.of_01 ] [1] connected  
[openflow.of_01 ] [3] connected  
[openflow.of_01 ] [4] connected  
[openflow.of_01 ] [2] connected  
[openflow.discovery ] link detected: 3 -> 1  
[openflow.spanning_tree] 5 ports changed
```

```
sudo mn --custom ~/mininet/custom/loop.py  
--topo mytopo --arp --mac --switch ovsk --test  
pingall --controller remote
```

...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

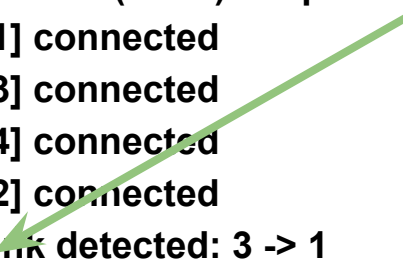
\*\*\* Results: 0% dropped (2/2 received)

...

# Mininet - Controlador remoto

➔ ~ pox/pox.py samples.spanning\_tree  
POX 0.1.0 (beta) / Copyright 2011-2013 James  
McCauley, et al.

```
[core                ] POX 0.1.0 (beta) is up.  
[openflow.of_01     ] [1] connected  
[openflow.of_01     ] [3] connected  
[openflow.of_01     ] [4] connected  
[openflow.of_01     ] [2] connected  
[openflow.discovery ] link detected: 3 -> 1  
[openflow.spanning_tree] 5 ports changed
```



## openflow.discovery

Se corre un proceso mediante el cual cada switch aprende toda la topología de red mediante el protocolo LLDP - I2\_learning switch

# Mininet - Controlador remoto

➔ ~ pox/pox.py samples.spanning\_tree  
POX 0.1.0 (beta) / Copyright 2011-2013 James  
McCauley, et al.

```
[core                ] POX 0.1.0 (beta) is up.  
[openflow.of_01     ] [1] connected  
[openflow.of_01     ] [3] connected  
[openflow.of_01     ] [4] connected  
[openflow.of_01     ] [2] connected  
[openflow.discovery ] link detected: 3 -> 1  
[openflow.spanning_tree] 5 ports changed
```

**openflow.spanning\_tree**  
Además del proceso de  
aprendizaje, se corre un proceso  
de spanning tree para evitar los  
bucles



# Mininet - Controlador remoto

→ ~ pox/pox.py samples.spanning\_tree  
POX 0.1.0 (beta) / Copyright 2011-2013 James  
McCauley, et al.

```
[core] POX 0.1.0 (beta) is up.  
[openflow.of_01] [1] connected  
[openflow.of_01] [3] connected  
[openflow.of_01] [4] connected  
[openflow.of_01] [2] connected  
[openflow.discovery] link detected: 3 -> 1  
[openflow.spanning_tree] 5 ports changed
```

**openflow.spanning\_tree**  
Además del proceso de  
aprendizaje, se corre un proceso  
de spanning tree para evitar los  
bucles

**IMPORTANTE**  
**EN EL TP NO DEBEN USAR SPANNING TREE**

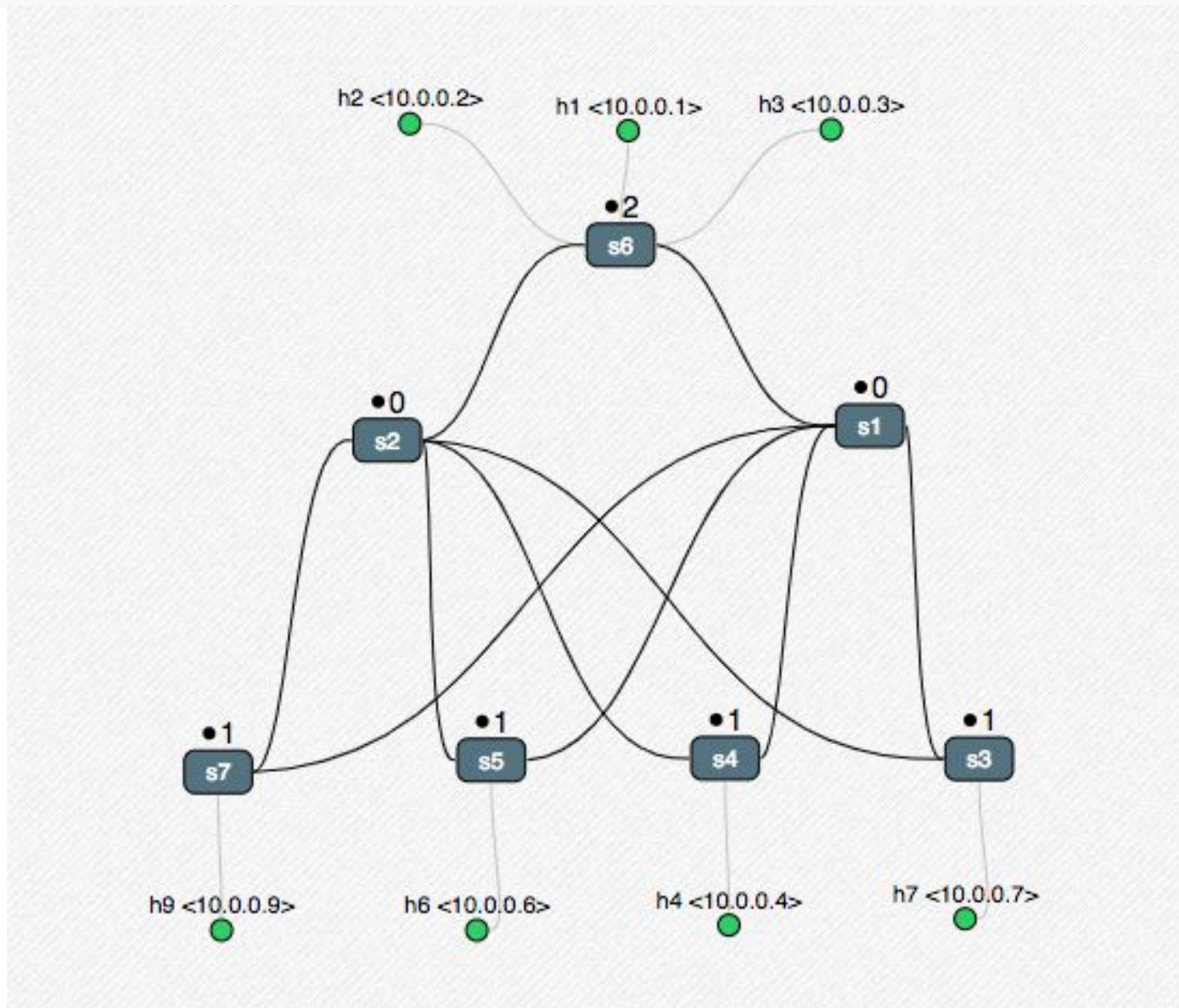
# TP 3 - Datacenter - Topología

Fat-tree<sup>(4)</sup> simplificada:

- 3 hosts conectados a la raíz - clientes
- Cantidad dinámica de niveles
  - $\#sw(n) = 2^{(n-1)}$
- 1 host conectado a cada hoja - proveedores

(4) <http://ccr.sigcomm.org/online/files/p63-alfares.pdf>

# TP 3 - Datacenter - Topología

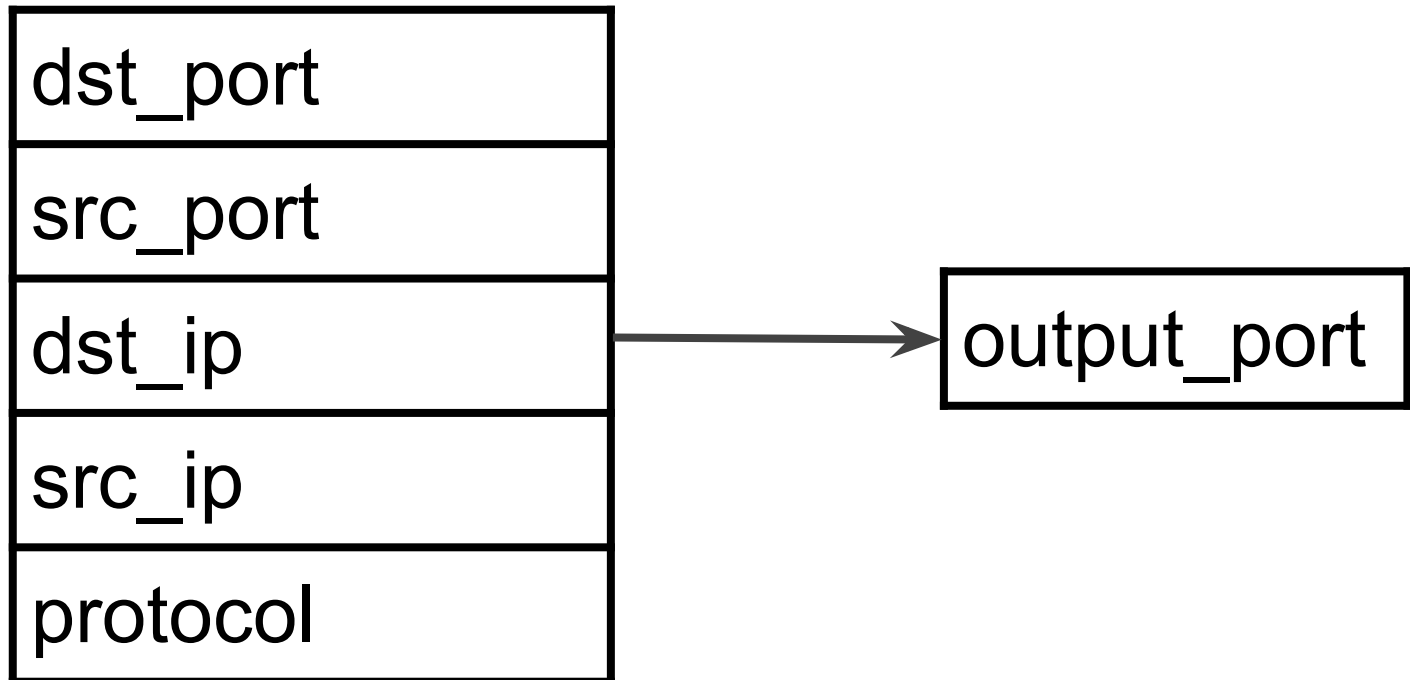


# TP 3 - Datacenter - Controlador

- Direccionar el tráfico por flujos utilizando ECMP.
  - Todos los paquetes pertenecientes a un flujo debén seguir la misma ruta.
  - El tráfico de diferentes flujos debe ser balanceado entre todos los enlaces posibles.

¿Cómo definimos un flujo?

# ¿Cómo definimos un flujo?



# TP 3 - Datacenter - Entrega

- **Mínimo 3 personas, idealmente 4.**
- Fecha Entrega: jueves 20/07/2020 - 19:00 hs
- Formato Entrega:
  - Informe y Código por campus - Una por grupo
- Pruebas:
  - iperf
  - ping
  - Wireshark