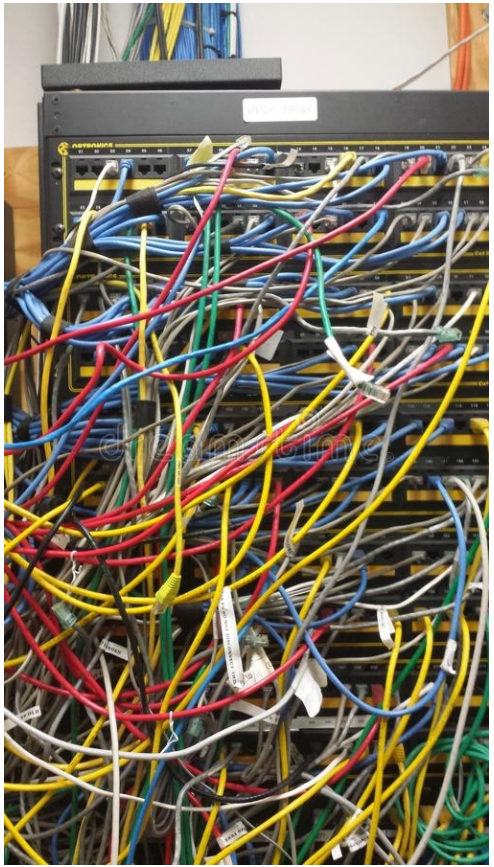
# Cómo desarrollar un switch de capa 2 sobre un controlador SDN

(usando Ryu framework)

Joaquín Gonzalez
PyConAr 2020

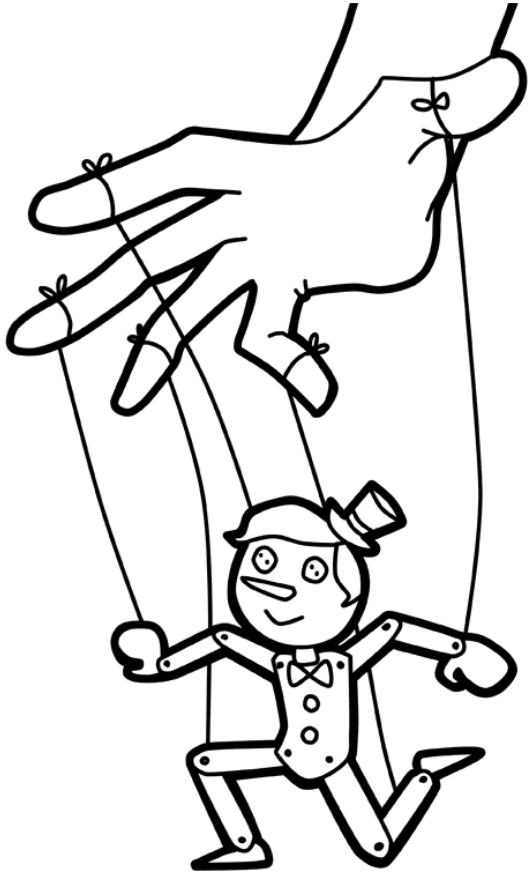
# Qué es un switch?



# El problema

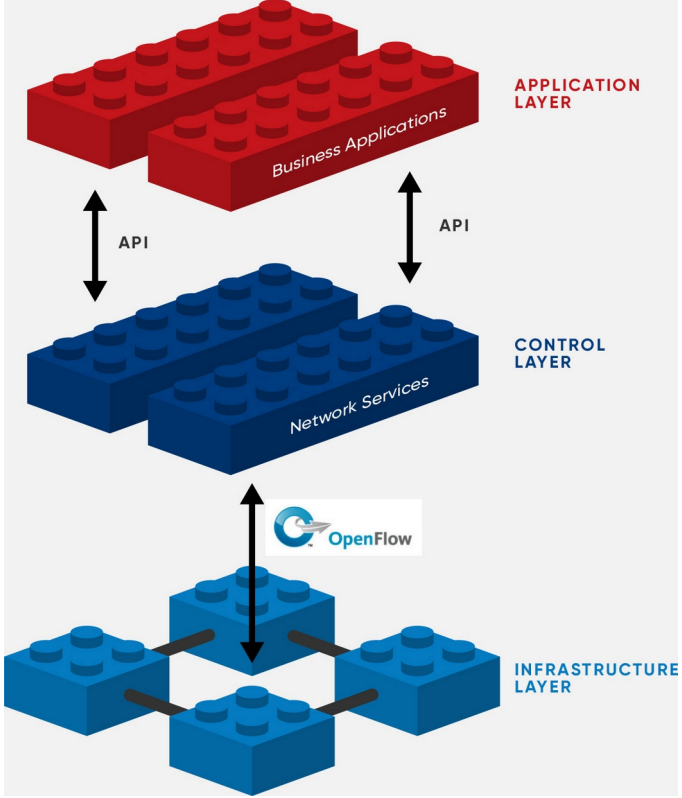
- Conectividad
  - Multiplicidad de vendors
  - Interfaces (CLI/API) no estándar
- Operación
  - Configuraciones manuales
  - Esquemas de LLD-HLD-MOP-Ventanas
- Automatización?

# Por qué no podemos operar elementos de red como infraestructura IT?



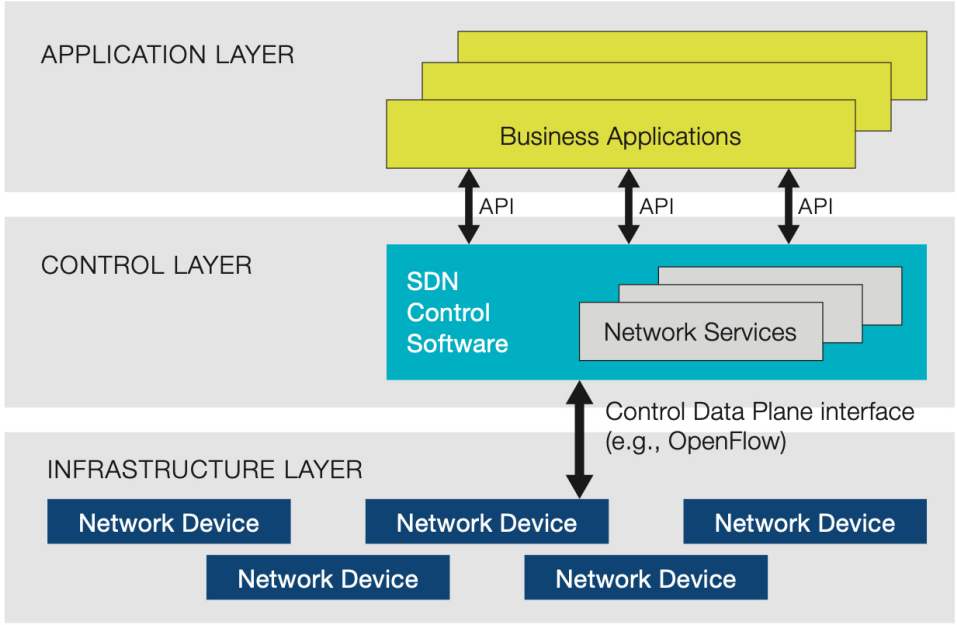
# SDN

- Desacoplemos el plano de control del plano de datos
- Estandaricemos las interfaces contra los elementos de red (southbound interfaces)
- Estandaricemos interfaces contra aplicaciones (northbound interfaces)



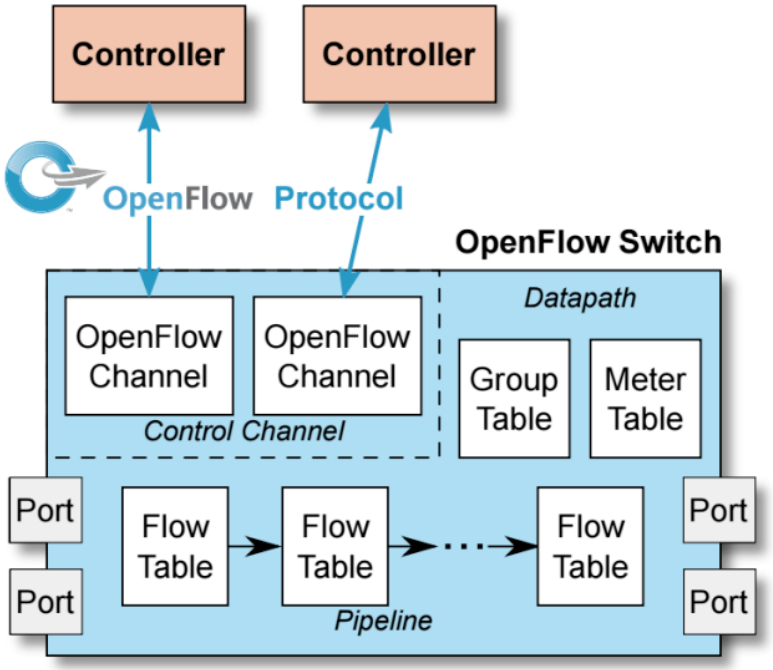
# SDN

- Las apliacaciones pueden pedir recursos a la red
- Podemos desarrollar nuevas funcionalidades **fuera de los elementos de red**
- Automatización :)



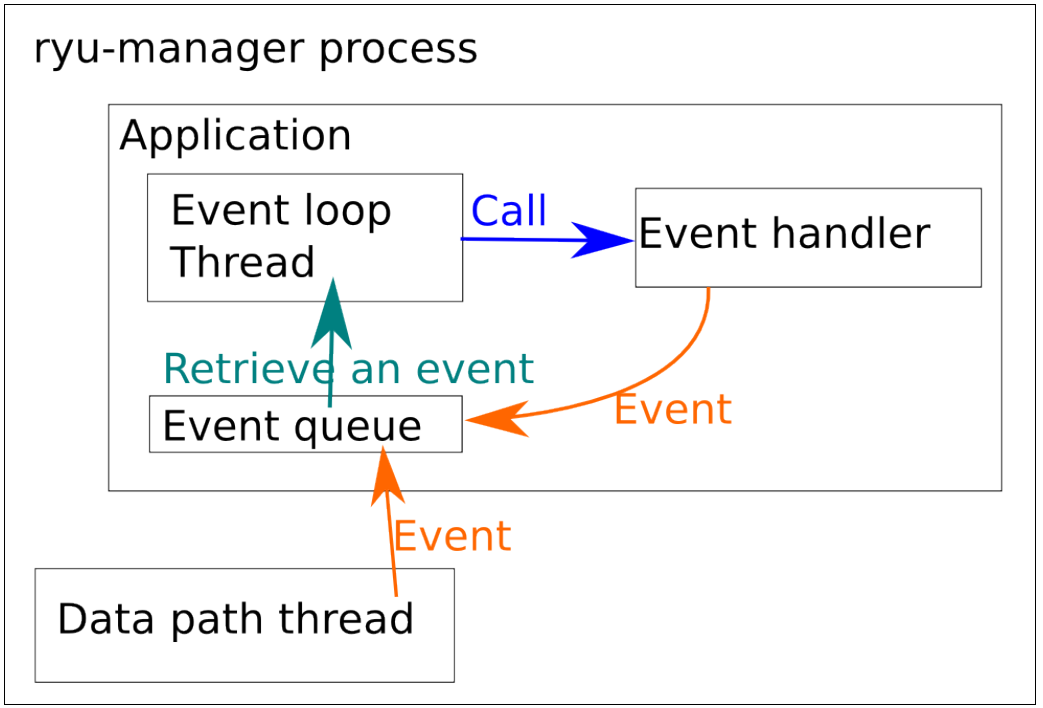
# SI - OpenFlow

- Protocolo southbound interface
- Basado en flujos para la toma de desiciones de conmutación



# Ryu framework

- Framework para desarrollar aplicaciones de red
- API bien definida
- Soporte OpenFlow



# Switch L2

Con flujos podemos simular comportamientos de distintas funcionalidades de red

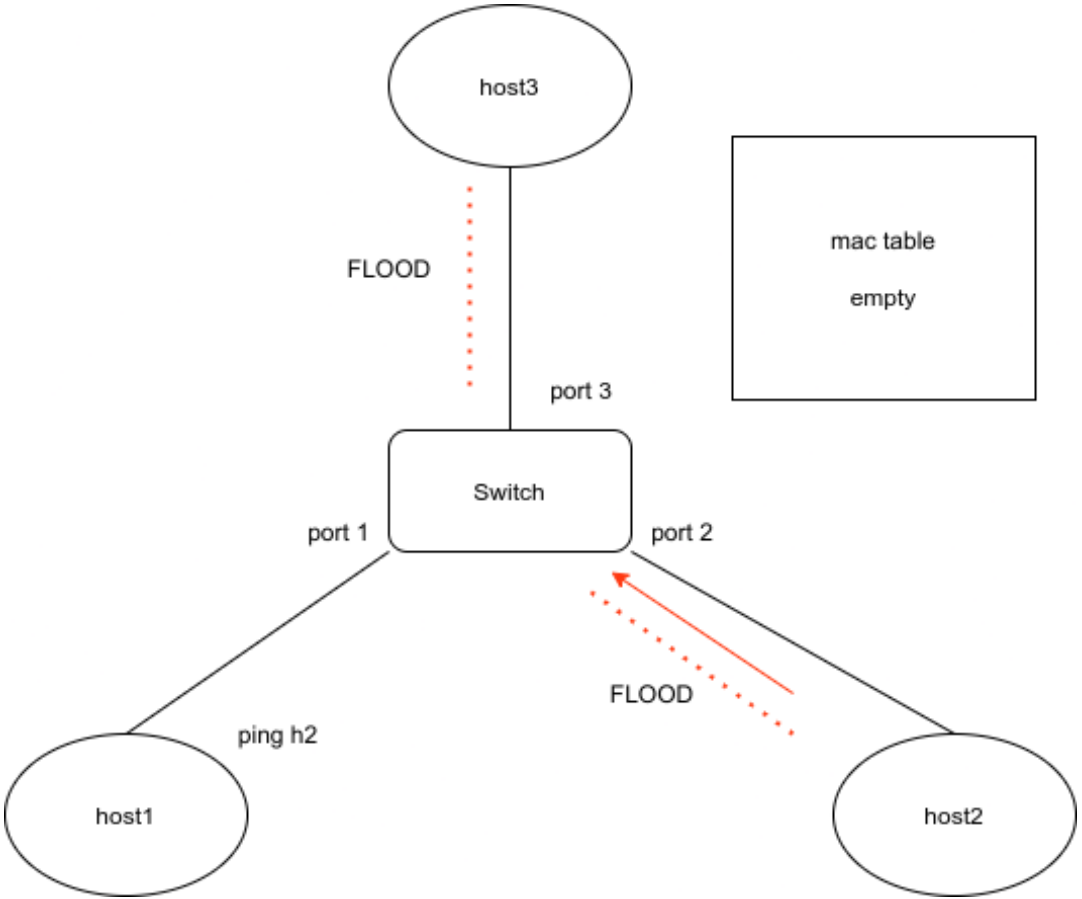
Leemos headers L2 => Switch

Leemos headers L3 => Router

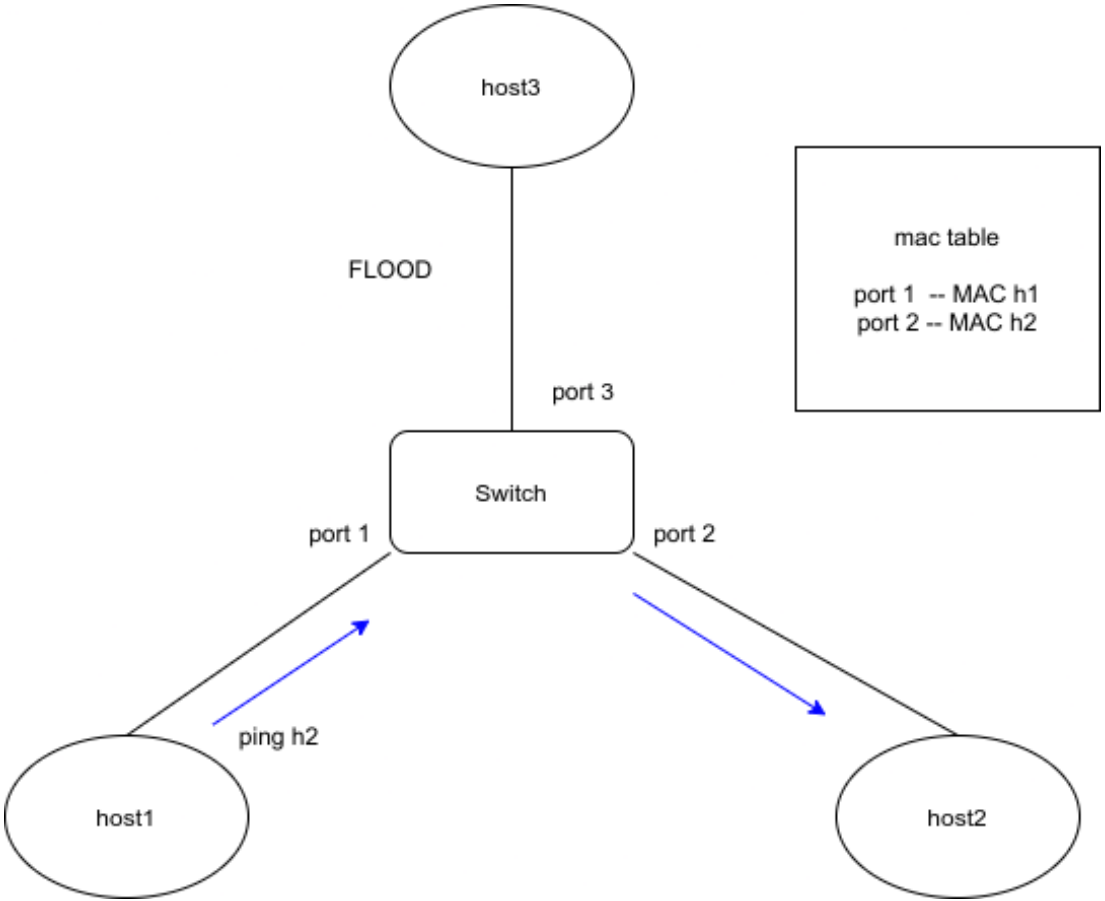
Leemos headers L4 => Firewall

```
class PyConArSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(PyConArSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

# Switch L2


host3
FLOOD
mac table
empty
port 3
Switch
port 1
port 2
FLOOD
ping h2
host1
host2

# Switch L2


host3
FLOOD
port 3
mac table
port 1 -- MAC h1
port 2 -- MAC h2
Switch
port 1
port 2
ping h2
host1
host2

# Switch L2

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):

    # usamos api de ryu para leer información del evento
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    dst = eth.dst
    src = eth.src
```

# Switch L2

Actualizamos tabla MAC del switch y, de no conocer el destino, hacemos FLOOD a todos los puertos del switch

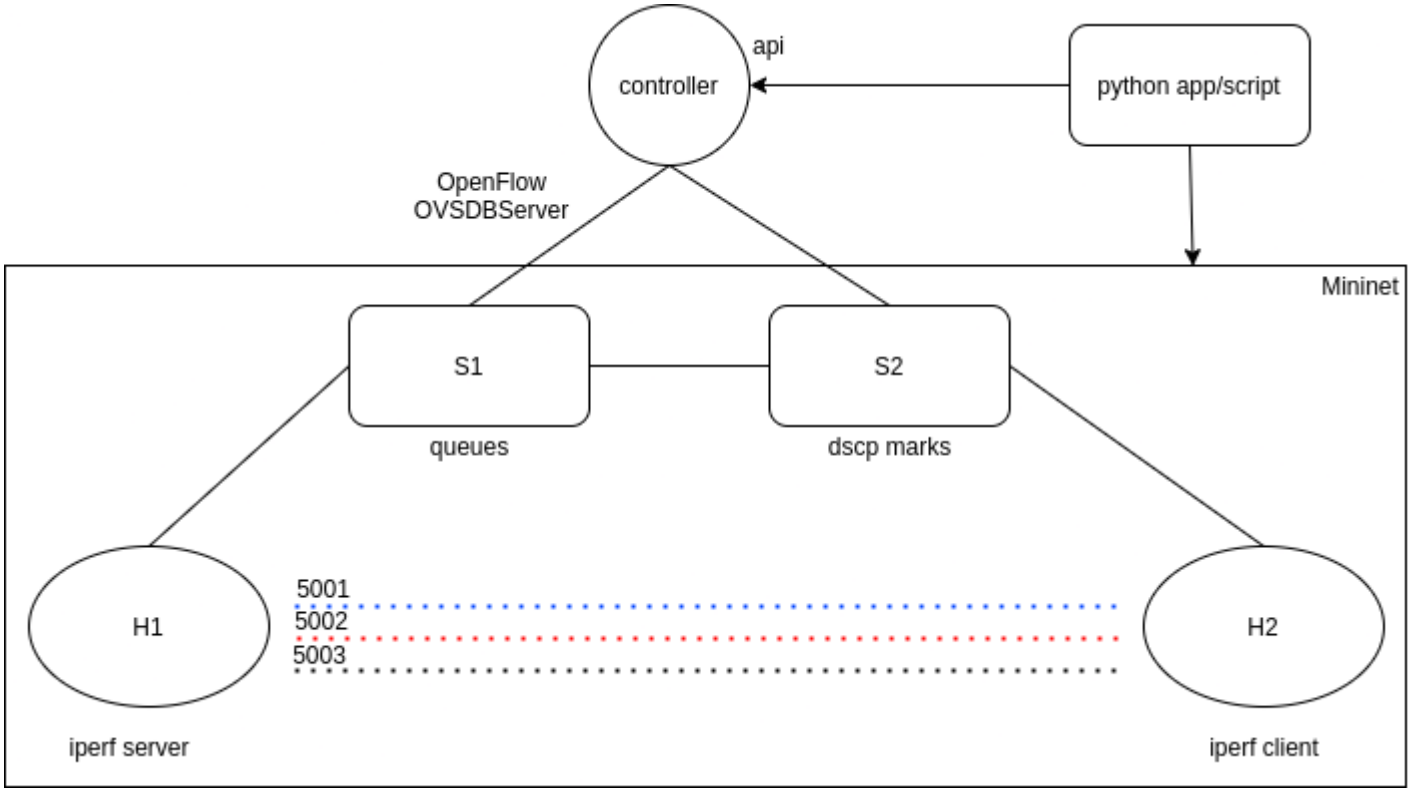
```
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

actions = [parser.OFPActionOutput(out_port)]
```

# Switch L2

Si el puerto OUTPUT no es FLOOD, entonces agregamos un flujo guardando las acciones a tomar para próximos paquetes con mismo destino y forwardeamos el paquete

```
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)

    self.add_flow(datapath, 1, match, actions)

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)

    actions = [parser.OFPActionOutput(out_port)]

    datapath.send_msg(out)
```

# Calidad de Servicio



# Referencias


- https://ryu-sdn.org
- http://mininet.org
- https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
- https://www.opennetworking.org/sdn-definition/
- https://www.openvswitch.org/
- https://www.openconfig.net/

# Gracias :)

Dudas? Ideas? Proyectos?

https://github.com/joagonzalez/pyconar-2020

joagonzalez@gmail.com