

Introducción a SDN

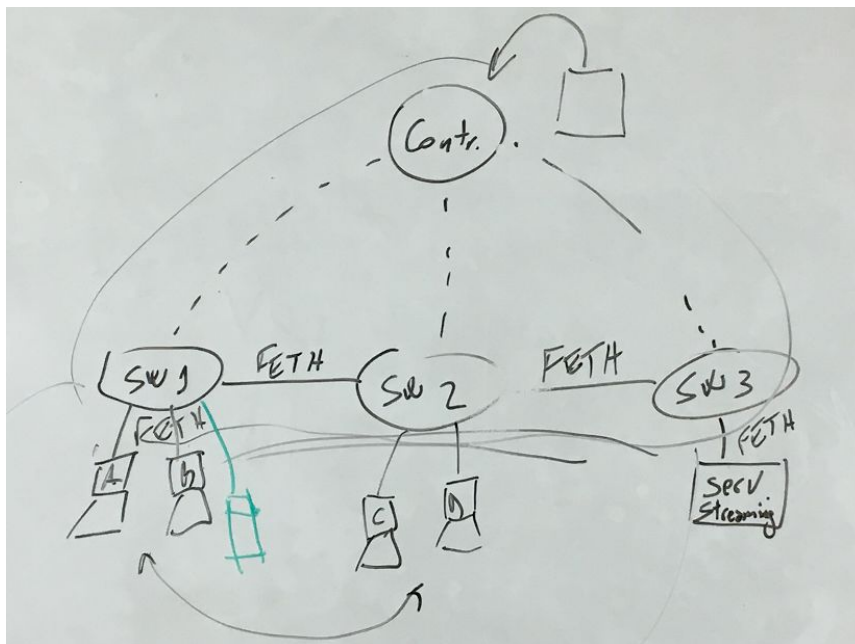
Se decide trabajar con una VM llamada Mininet, basada en UBUNTU. (<http://mininet.org/>)

Objetivo General de la investigación:

Hacer pruebas fehacientes del uso de SDN.

Trabajos a Realizar:

1. Que el grupo conozca el software y los rudimentos del uso de SDN
2. Armar maqueta con una red virtual para poder realizar diferentes pruebas. (mininet)
 - a. switches en cascada (FETH)
 - i. 1 controlador
 - ii. 1 servidor
 - iii. x PC
3. Correr Ejercicios, realizar pruebas y obtener resultados.
 - a. Prueba de conectividad entre todos los nodos
 - i. Ping
 - ii. Tráfico
 - iii. Falla
 - iv. Redundancia
 - b. Implementación de SDN
 - i. Uso sin configurar (no funca :()
 - ii. Configurar Switches como HUBS (paquetes 4 everywhere)
 - iii. Configurar Switches como Switches (cada paquete por su camino)
 - iv. Configurar Switches como Routers (diferentes redes)
 - v. Medir Performance de cada configuración (jperf)



Se deberá elaborar un informe de acuerdo a las siguientes pautas:

A nivel general

- Pre-requisitos / Herramientas utilizadas (Incluídas las versiones)

Por cada prueba:

- Escenario de prueba / Gráfico de la maqueta
- Descripción de la prueba (Breve descripción escrita de la prueba a realizar)
- Código de los scripts de mininet ejecutados + detalle de los componentes de software ejecutados en el controlador
- Resultados esperados (Breve descripción escrita de lo que se esperaba obtener)
- Ejecución de la prueba (Detalle utilizando capturas de pantalla)
- Resultados obtenidos (Análisis y conclusiones escritas de la prueba)

Versiones del documento

Version	Fecha	Descripción del cambio
0.1	9 de septiembre de 2017	Versión Inicial
0.2	10 de septiembre de 2017	Agregamos Switch como Router

Índice

2. Ambiente de prueba	4
3.b.I Controlador sin configurar	6
3.b.II Controlador como hub	7
3.b.III Controlador como Switch	9
3.b.IV Switch como Router	11
Conclusiones	20

2. Ambiente de prueba

Se decide a trabajar con el controlador Ryu.

Se descarga Mininet con Ryu incluido desde la pagina oficial de Ryu:

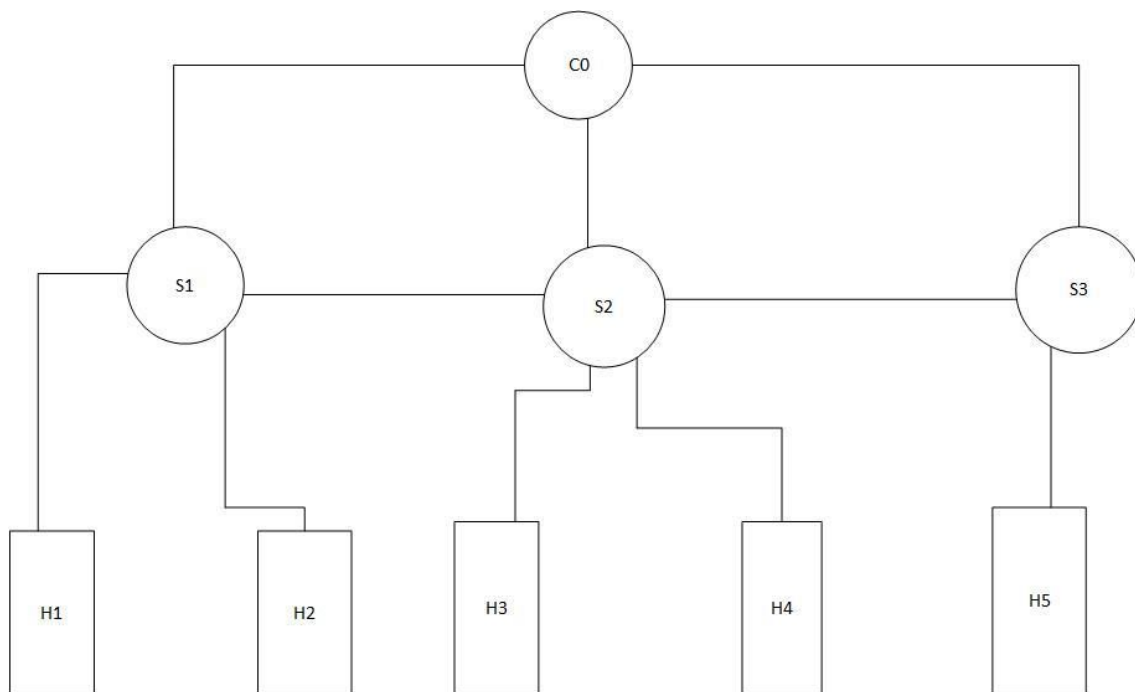
<https://osrg.github.io/ryu/>

Versión de Mininet: 2.1.0 y 2.3.0

Versión Ryu 3.6 y 4.17

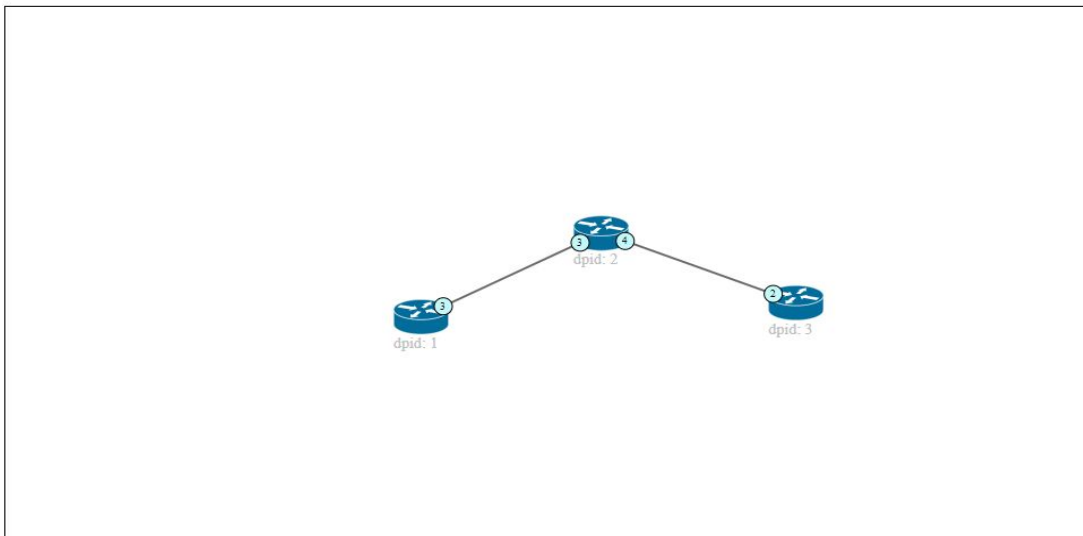
Versión Virtual Box 5.1.26

Topología Pedida: 1 Controlador, 3 switch, 5 host.



Interfaz gráfica del Ryu, la cual solo muestra los switch:

Ryu Topology Viewer



Script con la topología pedida:

```
from mininet.topo import Topo

class CincoHost( Topo ):

    def __init__( self ):
        Topo.__init__( self )

        h1 = self.addHost( "h1" )
        h2 = self.addHost( "h2" )
        h3 = self.addHost( "h3" )
        h4 = self.addHost( "h4" )
        h5 = self.addHost( "h5" )
        s1 = self.addSwitch( "s1" )
        s2 = self.addSwitch( "s2" )
        s3 = self.addSwitch( "s3" )

        self.addLink( h1, s1 )
        self.addLink( h2, s1 )
        self.addLink( h3, s2 )
        self.addLink( h4, s2 )
        self.addLink( h5, s3 )
        self.addLink( s1, s2 )
        self.addLink( s2, s3 )

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

3.b.I Controlador sin configurar

Para correr la topología:

```
ryu@ryu-vm:~$ sudo mn --custom ~/mininet/custom/cincohost.py --topo mytopo --controller remote --mac
```

--custom indica el parámetro en donde se encuentra el script que se va a correr.

--topo indica qué topología dentro del script debe ejecutarse. En la imagen se ve al final en donde definimos el nombre "mytopo" para ejecutarla.

--mac crea las direcciones en una manera fácil de leer. Arrancando desde la 00:00:00:00:00:00 en lugar de aleatoriamente.

Lo cual nos crea la topología pedida:

```
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
mininet>
```

Se ve como se creo los cinco hosts, los tres switch y se forman los links entre los distintos hosts con sus respectivos switch.

Sin configurar el controlador

Nodos disponibles:

```
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 s1 s2 s3
```

Pingall:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X
h2 -> X X X X
h3 -> X X X X
h4 -> X X X X
h5 -> X X X X
*** Results: 100% dropped (0/20 received)
```

Se ve cómo ningún paquete alcanza destino.

3.b.II Controlador como hub

En ryu/app se tiene el siguiente script hub.py:

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_0

class L2Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofp = dp.ofproto
        ofp_parser = dp.ofproto_parser

        actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
        out = ofp_parser.OFPPacketOut(
            datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port,
            actions=actions)
        dp.send_msg(out)
```

Se ejecuta para tener al controlador funcionando:

```
ryu@ryu-vm:~/ryu/ryu/app$ ryu-manager hub.py
loading app hub.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app hub.py of L2Switch
```

Y se ejecuta la topología en el mininet.

```
ryu@ryu-vm:~$ sudo mn --custom ~/mininet/custom/cincohost.py --topo mytopo --con
troller remote --mac
```

Se puede hacer un ping entre todos los hosts y vemos que se comunican sin problema:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Se prueba con un iperf en la versión 3.6 del controlador Ryu:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['2.19 Mbits/sec', '2.24 Mbits/sec']
```

Se prueba nuevamente iperf en la versión 4.17 de Ryu:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['7.25 Mbits/sec', '8.77 Mbits/sec']
```

El resultado obtenido en la última versión es más del doble que la primera.

Se abre en el Xterm los cinco hosts y se prueba mandar un ping del host 1 al 3 y ver quienes lo pueden ver/recibir.

```
root@ryu-vm:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=82.6 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=55.0 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=43.6 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=72.1 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 43.600/63.368/82.657/15.076 ms
root@ryu-vm:~# █
```


The image displays four network packet capture windows, each showing traffic between a specific node (h2, h3, h4, h5) and a destination IP 10.0.0.1. The traffic includes ICMP echo replies and ARP requests/replies. The hex and ASCII data are visible for each packet, showing details like source and destination MAC addresses, IP addresses, and protocol types.

3.b.III Controlador como Switch

Se ejecuta en `ryu/ryu/app simple_switch.py` para que el controlador funcione como switch. Este mismo archivo viene en distintas versiones.

Cabe aclarar que OpenFlow viene hasta la versión 1.5, siendo 1.3 la más utilizada comúnmente. Para el caso este se va a utilizar la versión base, 1.0.

```
ryu@ryu-vm:~/ryu/ryu/app$ ryu-manager simple_switch.py
loading app simple_switch.py
loading app ryu.controller.ofp_handler
instantiating app simple_switch.py of SimpleSwitch
instantiating app ryu.controller.ofp handler of OFPHandler
```

Se prueba el ping entre todos los hosts.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Se mide el ancho de banda con `iperf` en la versión 3.6 del controlador Ryu:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['3.35 Mbits/sec', '3.45 Mbits/sec']
```

Ahora se prueba iperf en la versión 4.17 del controlador:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['12.4 Gbits/sec', '12.4 Gbits/sec']
```

Se puede ver una diferencia considerable. Mucho más que en el caso del hub en ambas versiones del controlador. El iperf de la versión anterior es muy similar al del hub en su misma versión y bastante inferior en la última versión del controlador. Si bien se esperaba una diferencia, la diferencia es muy notoria.

Luego se prueba que realmente funcione como switch, se abren los cinco hosts en xterm y se envía desde el host 1 ping al host 3 y se ve que solamente ese host lo recibe.

```
root@ryu-vm:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=1.33 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=2.38 ms
64 bytes from 10.0.0.3: icmp_req=3 ttl=64 time=1.30 ms
64 bytes from 10.0.0.3: icmp_req=4 ttl=64 time=1.19 ms
64 bytes from 10.0.0.3: icmp_req=5 ttl=64 time=0.694 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4040ms
rtt min/avg/max/mdev = 0.694/1.382/2.382/0.552 ms
root@ryu-vm:~# █
```

```

Node: h2
root@ryu-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
[]

Node: h3
0x0010: 0054 0000 4000 4001 26a6 0a00 0001 0a00 .T.,@.@.&.....
0x0020: 0003 0800 5c8a 11e4 0005 96e4 b559 0000 ...Y.....
0x0030: 0000 7b7b 0500 0000 0000 1011 1213 1415 ..{.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 ..!"#$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 ..67
21:54:46.359789 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 4580, seq 5, length 64
0x0000: 0000 0000 0001 0000 0000 0003 0800 4500 .....E.
0x0010: 0054 af5a 0000 4001 b74b 0a00 0003 0a00 .T.Z.,@.K.....
0x0020: 0001 0000 648a 11e4 0005 96e4 b559 0000 ...d.....Y..
0x0030: 0000 7b7b 0500 0000 0000 1011 1213 1415 ..{.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 ..!"#$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 ..67
21:54:47.332757 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
0x0000: 0000 0000 0001 0000 0000 0003 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0003 0a00 0003 .....
0x0020: 0000 0000 0000 0a00 0001 .....
21:54:47.356285 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0003 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0003 0a00 0003 .....

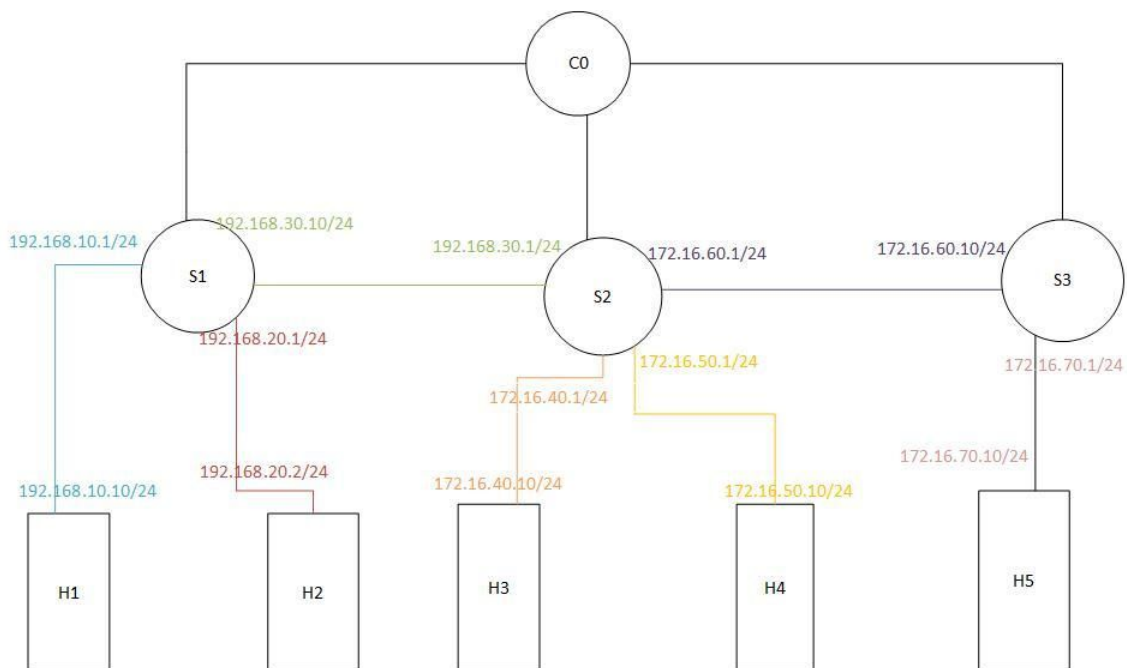
Node: h4
root@ryu-vm:~# tcpdump -XX -n -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
[]

Node: h5
root@ryu-vm:~# tcpdump -XX -n -i h5-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
[]

```

3.b.IV Switch como Router

Lo que se quiere lograr es lo siguiente:



Paso 1: Primero se carga mininet.

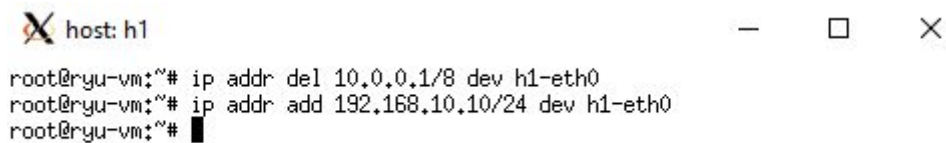
```
ryu@ryu-vm:~$ sudo mn --custom ~/mininet/custom/cincohost.py --topo mytopo --mac  
--controller remote --x
```

Paso 2: En otra ventana de Putty se indica a los switch que actúen con la versión de Open Flow 1.3. En donde x es el número de switch.

```
ryu@ryu-vm:~$ sudo ovs-vsctl set Bridge s1 protocols=OpenFlow13  
ryu@ryu-vm:~$ sudo ovs-vsctl set Bridge s2 protocols=OpenFlow13  
ryu@ryu-vm:~$ sudo ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

Paso 3: En las ventanas del xterm de cada host se borra la IP y se configura una nueva.

Ejemplo en el caso del host 1:



```
host: h1  
root@ryu-vm:~# ip addr del 10.0.0.1/8 dev h1-eth0  
root@ryu-vm:~# ip addr add 192.168.10.10/24 dev h1-eth0  
root@ryu-vm:~#
```

Paso 4: En otra ventana del Putty se corre el controlador de Ryu (rest_router.py).

```
ryu@ryu-vm:~/ryu/ryu/app$ ryu-manager rest_router.py
```

Al ejecutarse:

```
[RT][INFO] switch_id=0000000000000003: Set default route (drop) flow [cookie=0x0  
]  
[RT][INFO] switch_id=0000000000000003: Start cyclic routing table update.  
[RT][INFO] switch_id=0000000000000003: Join as router.  
[RT][INFO] switch_id=0000000000000002: Set SW config for TTL error packet in.  
[RT][INFO] switch_id=0000000000000002: Set ARP handling (packet in) flow [cookie  
=0x0]  
[RT][INFO] switch_id=0000000000000002: Set L2 switching (normal) flow [cookie=0x  
0]  
[RT][INFO] switch_id=0000000000000002: Set default route (drop) flow [cookie=0x0  
]  
[RT][INFO] switch_id=0000000000000002: Start cyclic routing table update.  
[RT][INFO] switch_id=0000000000000002: Join as router.  
[RT][INFO] switch_id=0000000000000001: Set SW config for TTL error packet in.  
[RT][INFO] switch_id=0000000000000001: Set ARP handling (packet in) flow [cookie  
=0x0]  
[RT][INFO] switch_id=0000000000000001: Set L2 switching (normal) flow [cookie=0x  
0]  
[RT][INFO] switch_id=0000000000000001: Set default route (drop) flow [cookie=0x0  
]  
[RT][INFO] switch_id=0000000000000001: Start cyclic routing table update.  
[RT][INFO] switch_id=0000000000000001: Join as router.
```

Paso 5: Se tiene que configurar los puntos de salida de cada router.

Del Router 1: La primera línea es para el link con el H1, la segunda con el H2 y la tercera con el S2

```
curl -X POST -d '{"address": "192.168.10.1/24"}'  
http://localhost:8080/router/000000000000000001
```

```
curl -X POST -d '{"address": "192.168.20.1/24"}'  
http://localhost:8080/router/000000000000000001
```

```
curl -X POST -d '{"address": "192.168.30.10/24"}'  
http://localhost:8080/router/000000000000000001
```

Del router 2: La primera es para el link con el S1, la segunda con el H3, la tercera con el H4 y la cuarta con el S3.

```
curl -X POST -d '{"address": "192.168.30.1/24"}'  
http://localhost:8080/router/000000000000000002
```

```
curl -X POST -d '{"address": "172.16.40.1/24"}'  
http://localhost:8080/router/000000000000000002
```

```
curl -X POST -d '{"address": "172.16.50.1/24"}'  
http://localhost:8080/router/000000000000000002
```

```
curl -X POST -d '{"address": "172.16.60.1/24"}'  
http://localhost:8080/router/000000000000000002
```

Del router 3: La primera es con el S2 y la segunda con el H5.

```
curl -X POST -d '{"address": "172.16.60.10/24"}'  
http://localhost:8080/router/000000000000000003
```

```
curl -X POST -d '{"address": "172.16.70.1/24"}'  
http://localhost:8080/router/000000000000000003
```

Ejemplo del primer router:

```

ryu@ryu-vm:~$ curl -X POST -d '{"address": "192.168.10.1/24"}' http://localhost:8080/router/000000000000000001
[{"switch_id": "000000000000000001", "command_result": [{"result": "success", "details": "Add address [address_id=1]"}]]ryu@ryu-vm:~$
ryu@ryu-vm:~$
ryu@ryu-vm:~$ curl -X POST -d '{"address": "192.168.20.1/24"}' http://localhost:8080/router/000000000000000001
[{"switch_id": "000000000000000001", "command_result": [{"result": "success", "details": "Add address [address_id=2]"}]]ryu@ryu-vm:~$
ryu@ryu-vm:~$
ryu@ryu-vm:~$ curl -X POST -d '{"address": "192.168.30.10/24"}' http://localhost:8080/router/000000000000000001
[{"switch_id": "000000000000000001", "command_result": [{"result": "success", "details": "Add address [address_id=3]"}]]ryu@ryu-vm:~$

```

Paso 6: En cada host del xterm agregamos la ruta:

host: h1:

```
# ip route add default via 192.168.10.1
```

host: h2:

```
# ip route add default via 192.168.20.1
```

host: h3:

```
# ip route add default via 172.16.40.1
```

host: h4:

```
# ip route add default via 172.16.50.1
```

host: h5:

```
# ip route add default via 172.16.70.1
```

Ejemplo del Host 5:

```

X host: h5
root@ryu-vm:~# ip addr del 10.0.0.5/8 dev h5-eth0
root@ryu-vm:~# ip addr add 172.16.70.10/24 dev h5-eth0
root@ryu-vm:~# ip route add default via 172.16.70.1
root@ryu-vm:~# █

```

Paso 7: Se agrega la ruta por default de cada router:

Router 1 al router 2

```
# curl -X POST -d '{"gateway": "192.168.30.1"}' http://localhost:8080/router/000000000000000001
```

Router 2 al router 1

```
# curl -X POST -d '{"gateway": "192.168.30.10"}' http://localhost:8080/router/000000000000000002
```

Router 3 al router 2

```
# curl -X POST -d '{"gateway": "172.16.60.1"}'  
http://localhost:8080/router/000000000000000003
```

Ejemplo del router 2 al router 1:

```
ryu@ryu-vm:~$ curl -X POST -d '{"gateway": "192.168.30.10"}' http://localhost:8080/router/000000000000000002  
[{"switch_id": "000000000000000002", "command_result": [{"result": "success", "details": "Add route [route_id=1]"}]]ryu@ryu-vm:~$
```

Paso 7:

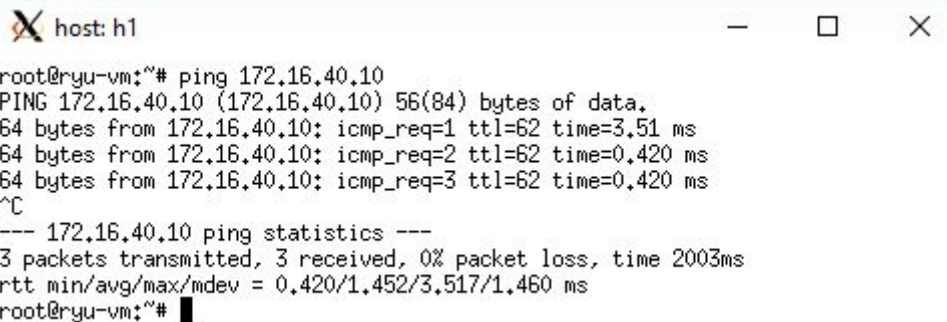
Por último, setear una ruta estática del router 2 al host 5.

```
# curl -X POST -d '{"destination": "172.16.70.0/24", "gateway": "172.16.60.10"}'  
http://localhost:8080/router/000000000000000002
```

```
ryu@ryu-vm:~$ curl -X POST -d '{"destination": "172.16.70.0/24", "gateway": "172.16.60.10"}' http://localhost:8080/router/000000000000000002  
[{"switch_id": "000000000000000002", "command_result": [{"result": "success", "details": "Add route [route_id=2]"}]]ryu@ryu-vm:~$
```

Habiendo hecho esto se puede probar la comunicación entre distintos hosts:

Del host 1 al host 3:



```
host: h1  
root@ryu-vm:~# ping 172.16.40.10  
PING 172.16.40.10 (172.16.40.10) 56(84) bytes of data:  
64 bytes from 172.16.40.10: icmp_req=1 ttl=62 time=3.51 ms  
64 bytes from 172.16.40.10: icmp_req=2 ttl=62 time=0.420 ms  
64 bytes from 172.16.40.10: icmp_req=3 ttl=62 time=0.420 ms  
^C  
--- 172.16.40.10 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2003ms  
rtt min/avg/max/mdev = 0.420/1.452/3.517/1.460 ms  
root@ryu-vm:~#
```

Del host 2 al host 5:

```
root@ryu-vm:~# ping 172.16.70.10
PING 172.16.70.10 (172.16.70.10) 56(84) bytes of data.
64 bytes from 172.16.70.10: icmp_req=1 ttl=61 time=3.03 ms
64 bytes from 172.16.70.10: icmp_req=2 ttl=61 time=1.34 ms
64 bytes from 172.16.70.10: icmp_req=3 ttl=61 time=7.58 ms
64 bytes from 172.16.70.10: icmp_req=4 ttl=61 time=3.46 ms
^C
--- 172.16.70.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 1.345/3.859/7.589/2.294 ms
root@ryu-vm:~# █
```

Del host 3 al host 1:

```
root@ryu-vm:~# ping 192.168.10.10
PING 192.168.10.10 (192.168.10.10) 56(84) bytes of data.
64 bytes from 192.168.10.10: icmp_req=1 ttl=62 time=17.8 ms
64 bytes from 192.168.10.10: icmp_req=2 ttl=62 time=1.02 ms
64 bytes from 192.168.10.10: icmp_req=3 ttl=62 time=1.02 ms
64 bytes from 192.168.10.10: icmp_req=4 ttl=62 time=1.60 ms
^C
--- 192.168.10.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3012ms
rtt min/avg/max/mdev = 1.023/5.368/17.816/7.190 ms
root@ryu-vm:~# █
```

Del Host 4 al host 2:

```
root@ryu-vm:~# ping 192.168.20.10
PING 192.168.20.10 (192.168.20.10) 56(84) bytes of data.
64 bytes from 192.168.20.10: icmp_req=1 ttl=62 time=21.7 ms
64 bytes from 192.168.20.10: icmp_req=2 ttl=62 time=2.52 ms
64 bytes from 192.168.20.10: icmp_req=3 ttl=62 time=0.421 ms
^C
--- 192.168.20.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.421/8.235/21.756/9.599 ms
root@ryu-vm:~# █
```

Del host 5 al Host 4:


```
host: h5
root@ryu-vm:~# ping 172.16.50.10
PING 172.16.50.10 (172.16.50.10) 56(84) bytes of data.
64 bytes from 172.16.50.10: icmp_req=1 ttl=62 time=3.27 ms
64 bytes from 172.16.50.10: icmp_req=2 ttl=62 time=0.418 ms
64 bytes from 172.16.50.10: icmp_req=3 ttl=62 time=3.97 ms
64 bytes from 172.16.50.10: icmp_req=4 ttl=62 time=0.634 ms
^C
--- 172.16.50.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.418/2.073/3.973/1.570 ms
root@ryu-vm:~#
```

Se puede ver la configuración de cada router:

```
# curl http://localhost:8080/router/0000000000000001
```

```
ryu@ryu-vm:~$ curl http://localhost:8080/router/0000000000000001
[{"internal_network": [{"route": [{"route_id": 1, "destination": "0.0.0.0/0", "gateway": "192.168.30.1"}], "address": [{"address_id": 2, "address": "192.168.20.1/24"}, {"address_id": 3, "address": "192.168.30.10/24"}, {"address_id": 1, "address": "192.168.10.1/24"}]}], "switch_id": "0000000000000001"}]ryu@ryu-vm:~$
```

```
# curl http://localhost:8080/router/0000000000000002
```

```
ryu@ryu-vm:~$ curl http://localhost:8080/router/0000000000000002
[{"internal_network": [{"route": [{"route_id": 2, "destination": "172.16.70.0/24", "gateway": "172.16.60.10"}, {"route_id": 1, "destination": "0.0.0.0/0", "gateway": "192.168.30.10"}], "address": [{"address_id": 4, "address": "172.16.60.1/24"}, {"address_id": 3, "address": "172.16.50.1/24"}, {"address_id": 1, "address": "192.168.30.1/24"}, {"address_id": 2, "address": "172.16.40.1/24"}]}], "switch_id": "0000000000000002"}]ryu@ryu-vm:~$
```

```
# curl http://localhost:8080/router/0000000000000003
```

```
ryu@ryu-vm:~$ curl http://localhost:8080/router/0000000000000003
[{"internal_network": [{"route": [{"route_id": 1, "destination": "0.0.0.0/0", "gateway": "172.16.60.1"}], "address": [{"address_id": 2, "address": "172.16.70.1/24"}, {"address_id": 1, "address": "172.16.60.10/24"}]}], "switch_id": "0000000000000003"}]ryu@ryu-vm:~$
```

Para realizar un iperf en el host 5 mediante iperf -s se le indica que va a estar escuchando.

```
"Node: h5"
root@ubuntu:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 24] local 172.16.70.10 port 5001 connected with 192.168.10.10 port 47608
[ ID] Interval      Transfer    Bandwidth
[ 24] 0,0-10,0 sec  16.0 GBytes 13.7 Gbits/sec
█
```

Luego en el host 1 mediante iperf -c y la dirección del host al que está escuchando se realiza la prueba.

```
"host: h1"
root@ubuntu:~# iperf -c 172.16.70.10
-----
Client connecting to 172.16.70.10, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 23] local 192.168.10.10 port 47608 connected with 172.16.70.10 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 23] 0,0-10,0 sec  16.0 GBytes 13.7 Gbits/sec
root@ubuntu:~# █
```

Pruebas de tcpdump: Desde el host 2 al host 5 se ponen en espera y desde el host 1 se envía un ping al host 4. Los resultados:

```
"host: h1"
root@ubuntu:~# ping 172.16.50.10
PING 172.16.50.10 (172.16.50.10) 56(84) bytes of data:
64 bytes from 172.16.50.10: icmp_seq=1 ttl=62 time=1.98 ms
64 bytes from 172.16.50.10: icmp_seq=2 ttl=62 time=0.072 ms
64 bytes from 172.16.50.10: icmp_seq=3 ttl=62 time=0.076 ms
^C
--- 172.16.50.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.072/0.710/1.984/0.900 ms
root@ubuntu:~# █
```

```
"host: h4"
0x0060: 3637 67
10:10:35.245680 IP 172.16.50.10 > 192.168.10.10: ICMP echo reply, id 2407, seq
, length 64
0x0000: 56ac 821d 23b8 0000 0000 0004 0800 4500 V...#.....E.
0x0010: 0054 32d0 0000 4001 9f0c ac10 320a c0a8 .T2...@.....2...
0x0020: 0a0a 0000 c51e 0967 0003 4b8b b659 0000 .....g..K..Y..
0x0030: 0000 6dbf 0300 0000 0000 1011 1213 1415 ..m.....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!"#$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
10:10:38.256224 ARP, Request who-has 172.16.50.1 tell 172.16.50.10, length 28
0x0000: 56ac 821d 23b8 0000 0000 0004 0806 0001 V...#.....
0x0010: 0800 0604 0001 0000 0000 0004 ac10 320a .....2.
0x0020: 0000 0000 0000 ac10 3201 .....2.
10:10:38.269014 ARP, Reply 172.16.50.1 is-at 56:ac:82:1d:23:b8, length 46
0x0000: 0000 0000 0004 56ac 821d 23b8 0806 0001 .....V...#.....
0x0010: 0800 0604 0002 56ac 821d 23b8 ac10 3201 .....V...#...2.
0x0020: 0000 0000 0004 ac10 320a 0000 0000 0000 .....2.....
0x0030: 0000 0000 0000 0000 0000 0000 .....
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

```
"host: h2"
root@ubuntu:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@ubuntu:~#
```

```
"host: h3"
root@ubuntu:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@ubuntu:~#
```

```
"Node: h5"
root@ubuntu:~# tcpdump -XX -n -i h5-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@ubuntu:~#
```

Conclusiones

Si tomamos en cuenta la misma versión del controlador para todos los test, por ejemplo la 4.17, vemos que la performance va en aumento siendo la de los switch como hub la menos performante, los switch como switch con unos valores más aceptables y el router el que tiene la mejor performance, aunque no muy por encima.

Por otro lado lo más sencillo de configurar fue como hub o como switch, requiriendo el router varios pasos extra.

Configuración	Resultado
Sin configurar	--
Hub	7.25 Mbps/sec
Switch	12.4 Gbits/sec
Router	13.7 Gbits/sec