

Practica P4

Alumno : Nicolas Bousquet

Practica 1

Ejercicio 1. Dentro de la carpeta ejercicio1, ejecutar el comando `$ make run` para crear la red de mininet y las tablas en los switches con las reglas predefinidas.

a) Dibujar la topología de la red, incluyendo switches y hosts junto con sus direcciones IP y MAC. ¿Por qué existe conectividad total a nivel ICMP entre los equipos (aunque pertenezcan a subredes distintas)? (Nota: usar los archivos disponibles en la carpeta pod-topo como ayuda).

Existe una conectividad total a nivel ICMP entre los dispositivos aunque pertenezcan a subredes diferentes porque, como se puede ver en el archivo "s1-runtime.json", el switch s1 tiene acceso a las direcciones ip de origen y destino (capa 3). Entonces se comporta como un router y sabe en cuál de sus puertos se encuentra cada subred. Todo lo que tiene que hacer es redirigir el paquete al puerto correspondiente a la subred de destino.

b) Comprobar la información del log disponible (/logs/s1.log), ¿qué información útil podemos obtener? ¿Qué acción se le está aplicando al paquete? ¿Se modifica alguna cabecera? ¿Qué puerto de salida del switch se usa? (Nota: para ver cómo se actualiza en tiempo real mientras se lanza un ping, se puede usar el comando `$ tail -f s1.log`)

Realizo un "h1 ping h2".

Se puede ver lo que ocurre en el switch paso a paso. En particular, podemos ver cada paso del procesamiento del paquete por el switch.

La acción que se aplica al paquete es "ipv4_forward".

```
Action MyIngress.ipv4_forward
```

En particular, cambiamos la dirección mac de origen y de destino. La dirección mac de origen se convierte en la dirección mac del switch y la dirección mac de destino se convierte en la dirección mac del host de destino (h2 en este caso). También disminuimos el ttl y actualizamos el puerto de salida.

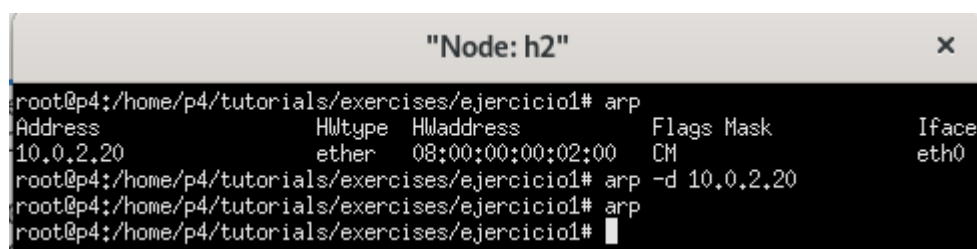
```
standard_metadata.egress_spec = port;  
hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;  
hdr.ethernet.dstAddr = dstAddr;  
hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
```

```
Processing packet received on port 1
```

El paquete llega por el puerto 1 y sale por el puerto 2.

```
Egress port is 2
```

c) Eliminar la entrada ARP en cualquiera de los hosts (por ejemplo, se puede usar el siguiente comando para eliminar la entrada ARP del host 2: \$ arp -d 10.0.2.20). Comprobar si existe conexión desde/hacia ese mismo host y explicar la razón.



```
"Node: h2"
root@p4:/home/p4/tutorials/exercises/ejercicio1# arp
Address          HWtype  HWaddress          Flags Mask          Iface
10.0.2.20         ether   08:00:00:00:02:00   CM                  eth0
root@p4:/home/p4/tutorials/exercises/ejercicio1# arp -d 10.0.2.20
root@p4:/home/p4/tutorials/exercises/ejercicio1# arp
root@p4:/home/p4/tutorials/exercises/ejercicio1#
```

```
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
^C
--- 10.0.2.2 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 14474ms

mininet> h2 ping h1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
From 10.0.2.2 icmp_seq=1 Destination Host Unreachable
From 10.0.2.2 icmp_seq=2 Destination Host Unreachable
From 10.0.2.2 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.1.1 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4127ms
pipe 4
```

h2 sólo tiene una entrada ARP en su tabla, la del switch. Obviamente, si lo eliminamos, h2 no puede transmitir paquetes a nadie. Por otro lado, las recibe todas pero no puede responder.

Ejercicio 2. Crear o modificar los archivos necesarios en la carpeta ejercicio2 para que la red tenga la siguiente configuración, existiendo conectividad ICMP total entre los hosts. (Nota: se puede comprobar usando el comando \$ pingall)

He modificado los archivos y todo funciona. Tuve que añadir manualmente las direcciones ARP (arp -i en pod-topo.json) porque el programa P4 no soporta el protocolo ARP.

```

() topology.json x {} s1-runtime.json
home > p4 > tutorials > exercises > ejercicio2 > pod-topo > {} topology.json > ...
1 {}
2 "hosts": {
3   "h1": {"ip": "10.0.1.1/24", "mac": "08:00:00:00:01:11",
4         "commands": ["route add default gw 10.0.1.10 dev eth0",
5                      "arp -i eth0 -s 10.0.1.10 08:00:00:00:01:10",
6                      "arp -i eth0 -s 10.0.1.2 08:00:00:00:01:12"]},
7   "h2": {"ip": "10.0.1.2/24", "mac": "08:00:00:00:01:12",
8         "commands": ["route add default gw 10.0.1.20 dev eth0",
9                      "arp -i eth0 -s 10.0.1.20 08:00:00:00:01:20",
10                     "arp -i eth0 -s 10.0.1.1 08:00:00:00:01:11"]},
11   "h3": {"ip": "10.0.2.1/24", "mac": "08:00:00:00:02:11",
12         "commands": ["route add default gw 10.0.2.10 dev eth0",
13                      "arp -i eth0 -s 10.0.2.10 08:00:00:00:02:10",
14                      "arp -i eth0 -s 10.0.2.2 08:00:00:00:02:12"]},
15   "h4": {"ip": "10.0.2.2/24", "mac": "08:00:00:00:02:12",
16         "commands": ["route add default gw 10.0.2.20 dev eth0",
17                      "arp -i eth0 -s 10.0.2.20 08:00:00:00:02:20",
18                      "arp -i eth0 -s 10.0.2.1 08:00:00:00:02:11"]},
19 },
20 "switches": {
21   "s1": { "runtime_json" : "pod-topo/s1-runtime.json"},
22   "s2": { "runtime_json" : "pod-topo/s2-runtime.json"}
23 },
24 "links": [
25   ["h1", "s1-p1"], ["h2", "s1-p2"], ["h3", "s2-p1"], ["h4", "s2-p2"], ["s1-p3", "s2-p3"]
26 ]
27
28

```

```

() topology.json {} s1-runtime.json x
home > p4 > tutorials > exercises > ejercicio2 > pod-topo > {} s1-runtime.json > [ ] table_entries > {} 2 > {} match > [ ] hdr.ipv4
1 {
2   "target": "bmv2",
3   "p4info": "build/ejercicio2.p4.p4info.txt",
4   "bmv2_json": "build/ejercicio2.json",
5   "table_entries": [
6     {
7       "table": "MyIngress.ipv4_lpm",
8       "match": {
9         "hdr.ipv4.dstAddr": ["10.0.1.1", 32]
10      },
11      "action_name": "MyIngress.ipv4_forward",
12      "action_params": {
13        "dstAddr": "08:00:00:00:01:11",
14        "port": 1
15      }
16    },
17    {
18      "table": "MyIngress.ipv4_lpm",
19      "match": {
20        "hdr.ipv4.dstAddr": ["10.0.1.2", 32]
21      },
22      "action_name": "MyIngress.ipv4_forward",
23      "action_params": {
24        "dstAddr": "08:00:00:00:01:12",
25        "port": 2
26      }
27    }
28  ]
29 }

```

```

{
  "table": "MyIngress.ipv4_lpm",
  "match": {
    "hdr.ipv4.dstAddr": ["10.0.2.1", 32]
  },
  "action_name": "MyIngress.ipv4_forward",
  "action_params": {
    "dstAddr": "08:00:00:00:02:11",
    "port": 3
  }
},
{
  "table": "MyIngress.ipv4_lpm",
  "match": {
    "hdr.ipv4.dstAddr": ["10.0.2.2", 32]
  },
  "action_name": "MyIngress.ipv4_forward",
  "action_params": {
    "dstAddr": "08:00:00:00:02:12",
    "port": 3
  }
}
]
}

```

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)

```

Practica 2

Ejercicio 1. Usando como base el ejercicio1 de la Práctica 1.

a) Modificar los archivos necesarios para bloquear todo el tráfico, excepto el que tenga origen h1 y destino h4. Para ello, crear una nueva tabla llamada 'firewall' que pueda hacer match de forma exacta con las direcciones IP de origen y destino. La

acción por defecto será descartar el paquete. (Nota: Todos los paquetes deben ser procesados por ambas tablas).

Todo funciona correctamente.

```
table firewall {
    key = {
        hdr.ipv4.srcAddr: exact;
        hdr.ipv4.dstAddr: exact;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}

apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
        firewall.apply();
    }
}
```

Sólo he añadido la conexión de h4 a h1 para que h4 pueda responder al ping de h1.

```
{
  {
    "table": "MyIngress.firewall",
    "match": {
      "hdr.ipv4.srcAddr": "10.0.1.1",
      "hdr.ipv4.dstAddr": "10.0.4.4"
    },
    "action_name": "MyIngress.ipv4_forward",
    "action_params": {
      "dstAddr": "08:00:00:00:04:44",
      "port": 4
    }
  },
  {
    "table": "MyIngress.firewall",
    "match": {
      "hdr.ipv4.srcAddr": "10.0.4.4",
      "hdr.ipv4.dstAddr": "10.0.1.1"
    },
    "action_name": "MyIngress.ipv4_forward",
    "action_params": {
      "dstAddr": "08:00:00:00:01:11",
      "port": 1
    }
  }
}
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X h4
h2 -> X X X
h3 -> X X X
h4 -> h1 X X
*** Results: 83% dropped (2/12 received)
```

b) Modificar el código para que se reste 10 al campo TTL en vez de 1 y demostrarlo capturando en Wireshark.

Aquí está el código que he modificado. No sé cómo podría probarlo. Aparte de crear un bucle de red, pero con la topología del ejercicio 1 (1 switch), eso es imposible.

```

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    if (hdr.ipv4.ttl > 10) {
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }
}

```

c) ¿Qué diferencias existen en el procesamiento de los paquetes entre la opción 1 y la opción 2? Comprobarlo usando el código y justificar la respuesta.

La opción 1 funciona correctamente, el firewall bloquea todas las conexiones excepto la de h1 y h4.

```

apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
        firewall.apply();
    }
}

```

```

mininet> pingall
*** Ping: testing ping reachability
h1 -> X X h4
h2 -> X X X
h3 -> X X X
h4 -> h1 X X
*** Results: 83% dropped (2/12 received)

```

Sin embargo, si se invierte el orden de aplicación de las dos tablas (opción 2), el firewall deja de funcionar y todas las conexiones son posibles.

```

apply {
    if (hdr.ipv4.isValid()) {
        firewall.apply();
        ipv4_lpm.apply();
    }
}

```



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Obviamente, cuando dos mesas compiten, sólo se mantiene la última.