Reliable PBR with IP SLA

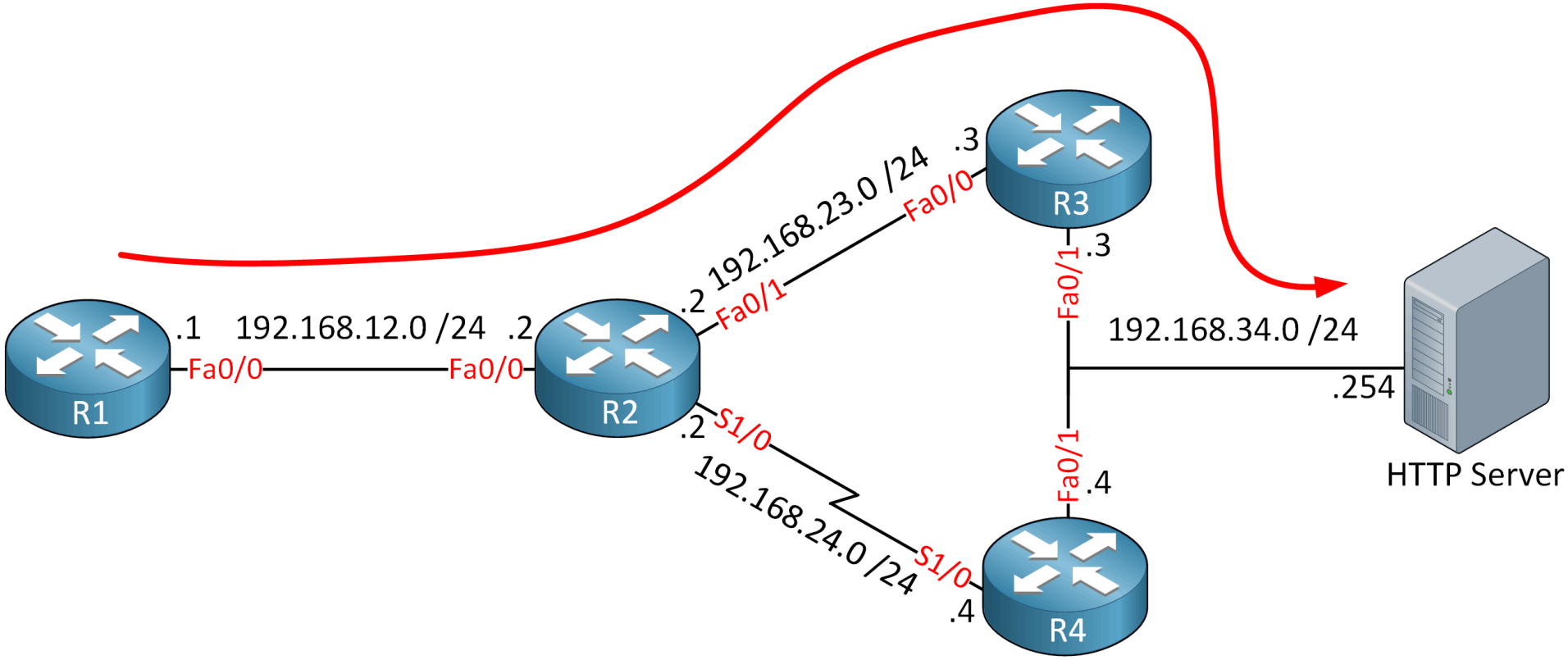
Lesson Contents

In previous lessons I explained how you can use PBR (Policy Based Routing) to overrule the routing table for certain types of traffic. I also explained in another lesson how IP SLA can be used to measure your network performance.

This lesson will combine those two topics, we'll use PBR to overrule the routing table but only when our IP SLA operation is up and running. Let's check out the configuration!

Configuration

Here's the topology we will use:



We have 4 routers and a webserver that we want to reach from R1. Because of the slow serial link between R2 and R4, all traffic is routed through R3:

```
R1#traceroute 192.168.34.254

Type escape sequence to abort.
Tracing the route to 192.168.34.254

  1 192.168.12.2 44 msec 44 msec 12 msec
```

```
 2 192.168.23.3 40 msec 44 msec 24 msec
 3 192.168.34.254 32 msec 60 msec 52 msec
```

For whatever reason we prefer to use R4 when we want to reach the webserver at 192.168.34.254. The serial link however isn't very reliable so instead of simply using PBR to forward traffic to R4, we'll combine it with IP SLA. On R2 we will ping the other side of the serial link (192.168.24.4) and when we get a reply, we'll use R4 as the next hop to reach 192.168.34.254. Here's how it's done:

```
R2(config)#ip sla 1
R2(config-ip-sla)#icmp-echo 192.168.24.4
R2(config-ip-sla-echo)#frequency 10

R2(config)#ip sla schedule 1 start-time now life forever
```

First we configure IP SLA. I'll use a simple ICMP echo and we will run this operation forever. We can't "attach" IP SLA directly to the route-map that we will use for policy based routing so we'll configure object tracking:

```
R2(config)#track 1 ip sla 1
```

There we go, object number 1 is now connected to IP SLA operation 1. RTR (Response Time Reporter) is the old name for IP SLA. Let's continue:

```
R2(config)#ip access-list extended HTTP_SERVER
R2(config-ext-nacl)#permit ip any host 192.168.34.254
```

The access-list above will be used in the route-map for PBR. It matches the IP address of the webserver. Now we can create the route-map:

```
R2(config)#route-map PBR permit 10
R2(config-route-map)#match ip address HTTP_SERVER
R2(config-route-map)#set ip next-hop verify-availability 192.168.24.4 1 track 1
```

Our route-map matches on the access-list and sets the next hop to 192.168.24.4. The important part here is to use the **verify-availability** parameter. This is used to check our tracked object.

Last but not least, let's activate the route-map:

```
R2(config)#interface FastEthernet 0/0
R2(config-if)#ip policy route-map PBR
```

Everything is now in place. Let's verify our work.

Verification

Let's use a couple of show commands to see if everything is configured correctly. Let's start with IP SLA:

```
R2#show ip sla statistics

Round Trip Time (RTT) for Index 1
  Latest RTT: 19 milliseconds
Latest operation start time: *00:11:36.843 UTC Fri Mar 1 2002
Latest operation return code: OK
Number of successes: 21
```

```
Number of failures: 0
Operation time to live: Forever
```

Above you can see that IP SLA is working. Let's check object tracking:

```
R2#show track 1
Track 1
  Response Time Reporter 1 state
  State is Up
    1 change, last change 00:02:45
  Latest operation return code: OK
  Latest RTT (millisecs) 43
  Tracked by:
    ROUTE-MAP 0
```

Object tracking is looking good, finally let's check the route-map:

```
R2#show route-map PBR
route-map PBR, permit, sequence 10
  Match clauses:
    ip address (access-lists): HTTP_SERVER
  Set clauses:
    ip next-hop verify-availability 192.168.24.4 1 track 1  [up]
  Policy routing matches: 0 packets, 0 bytes
```

Everything is looking good. Let's generate some traffic on R1 to see if it actually works. Before I do this, let's enable a debug on R2:

```
R2#debug ip policy
Policy routing debugging is on
```

Let's try a trace on R1:

```
R1#traceroute 192.168.34.254

Type escape sequence to abort.
Tracing the route to 192.168.34.254

  1 192.168.12.2 72 msec 52 msec 8 msec
  2 192.168.24.4 52 msec 60 msec 44 msec
  3 192.168.34.254 64 msec 60 msec 52 msec
```

Great it's now using R4 as the next hop. You will see this on R2:

```
R2#
IP: s=192.168.12.1 (FastEthernet0/0), d=192.168.34.254, len 100, FIB policy match
IP: s=192.168.12.1 (FastEthernet0/0), d=192.168.34.254, g=192.168.24.4, len 100, FIB policy routed
```

Our debug tells us that the packets have been policy routed. Let's find out what happens when R4 is no longer reachable. To test this, I'll create an inbound access-list on R4 that denies ICMP traffic:

```
R4(config)#ip access-list extended NO_ICMP
R4(config-ext-nacl)#deny icmp any any
R4(config-ext-nacl)#permit ip any any
```

```
R4(config)#interface Serial 1/0
R4(config-if)#ip access-group NO_ICMP in
```

When the next IP SLA probe fails, you'll see this on R2:

```
R2#
%TRACKING-5-STATE: 1 rtr 1 state Up->Down
```

Object tracking has failed so PBR should no longer work:

```
R2#show route-map PBR
route-map PBR, permit, sequence 10
  Match clauses:
    ip address (access-lists): HTTP_SERVER
  Set clauses:
    ip next-hop verify-availability 192.168.24.4 1 track 1  [down]
  Policy routing matches: 7 packets, 474 bytes
```

Let's try another trace on R1 to see if this is true or not:

```
R1#traceroute 192.168.34.254

Type escape sequence to abort.
Tracing the route to 192.168.34.254
```

```
  1 192.168.12.2 44 msec 64 msec 16 msec
  2 192.168.23.3 48 msec 44 msec 36 msec
  3 192.168.34.254 52 msec 64 msec 52 msec
```

As you can see it's using R3 again as the next hop. You can also see this in the debug on R2:

```
R2#
IP: s=192.168.12.1 (FastEthernet0/0), d=192.168.34.254, len 28, FIB policy match
IP: s=192.168.12.1 (FastEthernet0/0), d=192.168.34.254, len 28, FIB policy rejected - normal forwarding
```

That's all there is to it. I hope this exampe has been useful to understand how you can combine PBR and IP SLA.