
INGI 2142 - Group 8

Assignment 3 :

Border Gateway Protocol (BGP)

Laurent Lamouline - 3597-05-00
Thibault Leruitte - 2972-05-00
Vincent Nuttin - 5772-05-00

April 28, 2010

Contents

1 Bootstrap AS1000	1
2 Going online	2
3 Lets make them pay	3
4 Engineer your traffic	4
5 Protect yourself from denial of service attack	5
6 BGP convergence	6

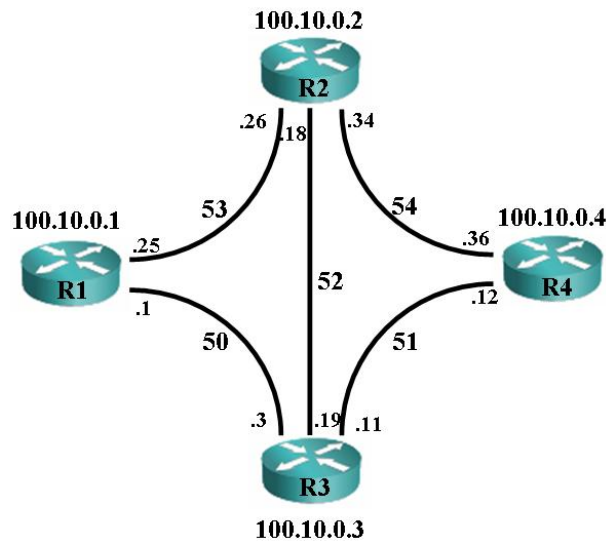
The aim of this third assignment was to configure an entire network by making multiple protocols (BGP, OSPF) coexist and to configure some VLANs between four routers interconnected by a switch. The final aim was to configure this network in order that it becomes a small realistic one. That is, a network with some clients, peers and providers that interact together by applying the corresponding traffic rules.

1 Bootstrap AS1000

This interconnection scheme is useful for an operator because in this way, it can know how traffic is transiting

Since we activate OSPF in AS1000, the four routers inside this area will propagate the path they learn everywhere if we do not specify that this traffic must stay in AS1000. To do so, we set passive OSPF on each external interface. This prevents the router from forwarding OSPF information outside the area it belongs to.

You can see an illustration of our VLANs inside AS1000:



Inside AS1000, there are only the routers R1, R2, R3, and R4, so we choose to make a full mesh between all those routers. We choose to do so, because we know that to establish iBGP sessions, each router must be connected to at least one router. But redundancy is necessary in case of a crash, so we add one more link for each router. With this choice, we were not far from full mesh configuration, so to make the system more robust and because there are not too many routers, we choose the full mesh solution.

To establish the iBGP sessions we used the following commands,

Listing 1: iBGP sessions establishment

```
1 router bgp "numero_AS"
2 neighbor "ip" remote-as "current_AS_id" //for each router in AS1000
```

These commands assign an AS id to the router first, and then we specify the router with which we establish the iBGP session by specifying its IP address and the same AS id as the source router.

2 Going online

To make all the routers in AS1000 advertise AS1000's prefix, we first have to enable a BGP routing process on these routers. Then, specify via the network command that there is a local network to this AS and enters it into the BGP table. Finally, we specify manually a route towards AS1000's prefix (pointing to null) in the router's table to make it advertised. The fact that it points to *null* is not a problem because the routing uses the longest prefix match so the packets will be correctly routed and not lost.

Listing 2: Example on R1

```
1 router bgp 1000
2 network 100.10.0.0 mask 255.255.0.0
3 ip route 100.10.0.0 255.255.0.0 nul 0
```

Now that all AS1000's routers are configured, we have to make them communicate with the rest of the network. To do this, we used eBGP sessions between R1, R2, R3, R4 and all their neighbors. The following command (exactly the same as for iBGP sessions but with other AS id):

Listing 3: eBGP sessions establishment

```
1 router bgp "numero_AS"
2 neighbor "ip" remote-as "current_AS_id" //For each router in AS1000 and
   their neighbor
```

Listing 4: Example on R3

```
1 router bgp 1000
2 neighbor 100.10.10.2 remote-as 1
3 neighbor 100.10.10.6 remote-as 2
```

We can see an example of routing table obtained with BGP bellow:

Listing 5: R1's BGP table

```
1 BGP table version is 13, local router ID is 100.10.0.1
2 Status codes: s suppressed, d damped, h history, * valid, > best, i -
   internal,
3           r RIB-failure, S Stale
4 Origin codes: i - IGP, e - EGP, ? - incomplete
5
6   Network          Next Hop          Metric LocPrf Weight Path
7   *>i1.0.0.0/16    100.10.10.2       0      200      0 1 i
8   *                100.10.10.14      75      100      0 100 10 1 i
9   *                100.10.10.10      100     100      0 10 1 i
10  *>i1.0.3.4/32     100.10.66.66      0      100      0 1 i
11  * 1.0.128.0/17   100.10.10.14      75      100      0 100 10 1 i
12  *>                100.10.10.10      100     100      0 10 1 i
13  *>i2.0.0.0/16    100.10.10.6       0      200      0 2 i
14  * 10.0.0.0/16    100.10.10.14      75      100      0 100 10 i
15  *>                100.10.10.10      0      100      0 10 i
16  *>i20.0.0.0/16   100.10.10.30      0      100      0 20 i
17  *                100.10.10.14      75      100      0 100 200 20 i
18  * i100.0.0.0/16  100.10.10.18      0      75      0 100 i
19  *>                100.10.10.14      0      75      0 100 i
20  *>i100.10.0.0/16 100.10.0.3         0      100      0 i
21  * i              100.10.0.4         0      100      0 i
22  * i200.0.0.0/16 100.10.10.18      0      75      0 100 200 i
23  *>                100.10.10.14      75      100      0 100 200 i
```

3 Lets make them pay

This part is dedicated to the building of economical relationships in the network. To do that, we used BGP communities to constraint route propagation. The idea is to set a tag to the routes in order to know from where the route has been learned and forward it consequently. In this way, we can manage the way routes are redistributed between routers. The commands we used to build this configuration are:

Listing 6: Configuring BGP communities

```
1 // Configuration of the format for the tags
2 ip bgp-community new-format
3
4 // assign an input route map
5 neighbor 100.10.10.2 route-map Client in
6
7 // route-map client
8 route-map Client permit 10
9 set community 1000:100
10
11 // assign an output route map
12 neighbor 100.10.10.18 route-map DataAS out
13
14 // route map data as
15 route-map DataAS permit 10
16 match ip address 101
17
18 route-map DataAS permit 20
19 match community 100
20
21 // to complete the route map
22 ip community-list 100 permit 1000:100
23 access-list 101 permit ip host 100.10.0.0 host 255.255.0.0
```

We pay attention to correctly assign the options according the type of the AS. From a provider to a clients, we give all the route known and to the peers and the providers, we announce all the internal routes and the ones of its client(s).

In order to check if the policies are correctly applied, we can use the *traceroute* command from PEER1 to PEER2. The result obtained is given below:

Listing 7: Configuring BGP communities

```
1 PEER1#traceroute 20.0.0.1 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 20.0.0.1
5
6  1 100.0.10.1 [AS 100] 8 msec 4 msec 4 msec
7  2 100.0.10.6 [AS 100] 68 msec 32 msec 44 msec
8  3 200.0.10.2 [AS 200] 64 msec * 116 msec
```

Here, we can see that packets go from PEER1 to PROV1 in the first time. It's exactly what it is expected because packets can not go trough AS1000 because PEER1 does not know the routes from this AS and neither from AS1 so the only way packets can go is via AS100. Then, the packets go from AS100 to AS200 via the peer link because this time the destination is one of the AS200 client. Finally, the packets go to the destination which is AS20. It seems that the policy is respected and other *traceroute* show us that it works correctly.

We can see the path taken between PROV1 and CUST2:

Listing 8: Traceroute from PROV1 to CUST2

```
1 PROV1#traceroute 2.0.0.1 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 2.0.0.1
5
6  1 100.10.10.17 [AS 1000] 16 msec 8 msec 20 msec
7  2 100.10.1.19 [AS 1000] 68 msec 120 msec 68 msec
8  3 100.10.10.6 [AS 1000] 140 msec * 60 msec
```

In this case, we can see that the path followed goes from the source (PROV1) to R2 and then from R2 to R3 and finally from R3 to CUST2. These paths are known because each client gives all its internal routes and the ones of its own clients to its provider.

Listing 9: Traceroute from R1 to CUST2

```
1 R1#traceroute 2.0.0.1 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 2.0.0.1
5
6  1 100.10.1.3 12 msec 72 msec 36 msec
7  2 100.10.10.6 144 msec * 28 msec
```

In this case, the path followed is a subset of the previous one so the idea is the same.

Listing 10: Traceroute from R1 to PROV2

```
1 R1#traceroute 200.0.0.1 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 200.0.0.1
5
6  1 100.10.1.26 4 msec 4 msec 12 msec
7  2 100.10.10.22 64 msec * 28 msec
```

Via iBGP, R1 knows that it can reach PROV2 by going through R2, so it sends its packets to R2. Then R2 knows PROV2 and sends the packets to it.

Listing 11: Traceroute from R4 to CUST2

```
1 R4#traceroute 2.0.0.1 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 2.0.0.1
5
6  1 100.10.10.34 8 msec * 16 msec
```

CUST2 is a direct client of R4 so the packets are directly sent to CUST2.

4 Engineer your traffic

To modelize the fact that AS100 connectivity is cheaper than AS200, we use the *localpref* attribute. By default, we choose to set a localpref of 50 for the providers, 100 for the peers and 200 for the clients. So, in order to make AS1000 prefer AS100 instead of AS200, we set a localpref of 75 on the links going from R1 to PROV1 and from R2 to PROV1.

If we want that CUST2 uses its link with R3 as a primary link and the one with R4 as a backup link for the outgoing traffic, we can use the *localpref* attribute anew. We choose to set a localpref of 25 on the CUST2-R4 link and 50 for the CUST2-R3 link. For the incoming traffic,

if we do not want to involve AS1000 in the configuration, we can use the *MED* property or the *AS prepending*. The first property consists in setting a MED of 1000 on the CUST2-R4 link so that another link will be preferred by the BGP decision process. The second idea is to specify prepending within the route-map with the *set as-path prepend* command for the incoming traffic. In this way, we can repeat several times the AS id so that the BGP decision process prefers a route with less ASes. The commands we can use are:

Listing 12: AS prepending

```
1 router bgp 2
2 route-map prepend permit 10
3 set as-path prepend 2 2 2 2 2
```

We have implemented the *MED* solution.

To deal with the congestion situation between R3 and CUST1, we choose to split the prefix from a /16 to two /17. We only advertise one /17 on the CUST1-PEER1 link so that half of the traffic matches. The remaining half part will follow the /16 advertised on both links. For the commands, we only had to advertise the /17 in addition of the /16 already configured on BGP and without forgetting to add this route in our local routing table to make it advertised.

5 Protect yourself from denial of service attack

To deal with DoS attack we will use the BGP communities anew. The main idea is to set a new black hole on each AS1000's router so that when AS1 warns that one of its IP (with a /32) is under attack, the traffic destined to it is "thrown" in the black hole of AS1000 routers.

Listing 13: Commands to configure all AS1000 routers

```
1 ip route 100.10.66.66 255.255.0.0 nul 0
```

The router R3 will look for 1:66 community tag from the links advertised by AS1. If such a tag exists, the route is modified such as the next-hop is the black hole. Then, this modified route is propagated to the other routers of the AS1000 via iBGP.

Listing 14: Community specification for the DoS attack by CUST1

```
1 router bgp 1
2   network 1.0.3.4 mask 255.255.255.255
3   !
4   ip access-list standard Victims
5     permit 1.0.3.4
6   !
7   route-map normal-out permit 30
8     match ip address Victims
9     set community 1:66
```

Listing 15: Community specification for the DoS attack by R3

```
1 route-map Client permit 5
2   match community 67
3   set ip next-hop 100.10.66.66
4   !
5 route-map Client permit 10
6   set local-preference 200
7   set community 1000:100
```

We can see that the DOS prevention is working by doing a traceroute from PROV2 to the attacked IP. The ping will be blocked at the first router traversed in AS1000.

Listing 16: Traceroute of the attacked IP

```

1 PROV2#traceroute 1.0.3.4 numeric
2
3 Type escape sequence to abort.
4 Tracing the route to 1.0.3.4
5
6   1 100.10.10.21 [AS 1000] 36 msec 4 msec 60 msec
7   2 100.10.10.21 [AS 1000] !H * !H

```

6 BGP convergence

Before shutdown the interface, here is a traceroute to reach CUST2 in normal conditions.

Listing 17: R2 traceroute to CUST2

```

1 R2#debug ip bgp updates
2 BGP updates debugging is on for address family: IPv4 Unicast
3 R2#traceroute 2.0.0.1 numeric
4
5 Type escape sequence to abort.
6 Tracing the route to 2.0.0.1
7
8   1 100.10.1.19 4 msec 52 msec 24 msec
9   2 100.10.10.6 32 msec * 16 msec

```

First, we shut down the interface facing R3.

Listing 18: Shutdown interface

```

1 R2#
2 R2#configure
3 R2(config)#interface f0/0.52
4 R2(config-subif)#shutdown
5
6 *Apr  2 10:29:57.123: %OSPF-5-ADJCHG: Process 8, Nbr 100.10.0.3 on
      FastEthernet0/0.52 from FULL to DOWN, Neighbor Down: Interface down or
      detachedxit
7 R2(config)#exit

```

After shutting down the interface, we can see a lot of BGP messages arriving at R2. We can see that in a first time, we do not know any route to reach CUST1 and CUST2 because we used the VLAN 52 that we have just shutted down. A new route to CUST1 is installed via PROV1 (AS 100) because iBGP only advertises the best route. From the route 2's point of view, the only way to reach AS1 is to use a eBGP link (an UPDATE message concerning CUST1 and a WITHDRAW message concerning CUST2 are sent to R2's neighbor).

Listing 19: BGP messages at R2

```

1 *Apr  2 10:30:02.195: BGP(0): no valid path for 2.0.0.0/16
2 *Apr  2 10:30:02.511: BGP(0): Revise route installing 1 of 1 routes for
      1.0.0.0/16 -> 100.10.10.18(main) to main IP table
3 *Apr  2 10:30:02.519: BGP(0): nettable_walker 2.0.0.0/16 no best path
4 *Apr  2 10:30:02.519: BGP(0): Revise route installing 1 of 1 routes for
      100.10.0.0/16 -> 100.10.0.4(main) to main IP table
5 *Apr  2 10:30:02.527: BGP(0): 100.10.0.1 NEXT_HOP part 1 net 1.0.0.0/16,
      next 100.10.10.18
6 *Apr  2 10:30:02.531: BGP(0): 100.10.0.1 send UPDATE (format) 1.0.0.0/16,
      next 100.10.10.18, metric 0, path 100 10 1
7 *Apr  2 10:30:02.535: BGP(0): 100.10.10.18 send unreachable 1.0.0.0/16
8 *Apr  2 10:30:02.535: BGP(0): 100.10.10.18 send UPDATE 1.0.0.0/16 --
      unreachable
9 *Apr  2 10:30:02.539: BGP(0): 100.10.10.18 send UPDATE 2.0.0.0/16 --
      unreachable

```

```

10 *Apr 2 10:30:02.543: BGP(0): 100.10.10.18 skip UPDATE 100.10.0.0/16 (
    chgflags: 0x800), next 100.10.0.4, path
11 *Apr 2 10:30:02.603: BGP(0): 100.10.10.18 rcv UPDATE w/ attr: nexthop
    100.10.10.18, origin i, originator 0.0.0.0, path 100 1000 2, community ,
    extended community
12 *Apr 2 10:30:02.611: BGP(0): 100.10.10.18 rcv UPDATE about 2.0.0.0/16 --
    DENIED due to: AS-PATH contains our own AS;
13 *Apr 2 10:30:02.619: BGP(0): updgrp 1 - 100.10.10.18 updates replicated for
    neighbors: 100.10.10.22
14 *Apr 2 10:30:02.623: BGP(0): updgrp 2 - 100.10.0.1 updates replicated for
    neighbors: 100.10.0.3 100.10.0.4

```

As we can see on the next listing, at this time, we are not able to ping CUST2 anymore.

Listing 20: R2 ping to CUST2

```

1 R2#ping 2.0.0.1
2
3 Type escape sequence to abort.
4 Sending 5, 100-byte ICMP Echos to 2.0.0.1, timeout is 2 seconds:
5 ....
6 Success rate is 0 percent (0/5)

```

R2 receives iBGP updates about CUST1 and CUST2 and will use these new paths to reach them.

Listing 21: BGP messages at R2 (following)

```

1 *Apr 2 10:30:31.215: BGP(0): Revise route installing 1 of 1 routes for
    1.0.0.0/16 -> 100.10.10.2(main) to main IP table
2 *Apr 2 10:30:31.223: BGP(0): Revise route installing 1 of 1 routes for
    2.0.0.0/16 -> 100.10.10.6(main) to main IP table
3 *Apr 2 10:30:31.223: BGP(0): 100.10.0.1 send unreachable 1.0.0.0/16
4 *Apr 2 10:30:31.227: BGP(0): 100.10.0.1 send UPDATE 1.0.0.0/16 --
    unreachable
5 *Apr 2 10:30:31.327: BGP(0): updgrp 2 - 100.10.0.1 updates replicated for
    neighbors: 100.10.0.3 100.10.0.4.
6 *Apr 2 10:30:33.331: BGP(0): 100.10.10.18 send UPDATE (format) 1.0.0.0/16,
    next 100.10.10.17, metric 0, path 1
7 *Apr 2 10:30:33.331: BGP(0): 100.10.10.18 send UPDATE (format) 2.0.0.0/16,
    next 100.10.10.17, metric 0, path 2
8 *Apr 2 10:30:33.431: BGP(0): updgrp 1 - 100.10.10.18 updates re222
9 plicated for neighbors: 100.10.10.22
10 \end{lstlisting}
11
12 Now, thanks to these new routes, R2 is able to ping CUST2 anew.
13 \begin{lstlisting}[label=q6l1,caption=R2 ping to CUST2]
14 R2#ping 2.0.0.1
15
16 Type escape sequence to abort.
17 Sending 5, 100-byte ICMP Echos to 2.0.0.1, timeout is 2 seconds:
18 !!!!!
19 Success rate is 100 percent (5/5), round-trip min/avg/max = 60/96/144 ms
20 R2#traceroute 2.0.0.1 numeric
21
22 Type escape sequence to abort.
23 Tracing the route to 2.0.0.1
24
25  1 100.10.1.36 12 msec
26    100.10.1.25 8 msec
27    100.10.1.36 4 msec
28  2 100.10.1.3 44 msec
29    100.10.1.11 48 msec
30    100.10.1.3 60 msec
31  3 100.10.10.6 68 msec * 80 msec

```

We can see that CUST2 was unreachable during approximately 30 seconds.