Experiment 2a - Setup eBGP

DE-CIX Academy

Version 2.1

1 Introduction

In this experiment we connect our AS to the outside world. In BGP we talk about eBGP if two different ASes interconnect with each other. This experiment is very simple as you just connect to one other AS and receive a few prefixes.

2 Network Setup

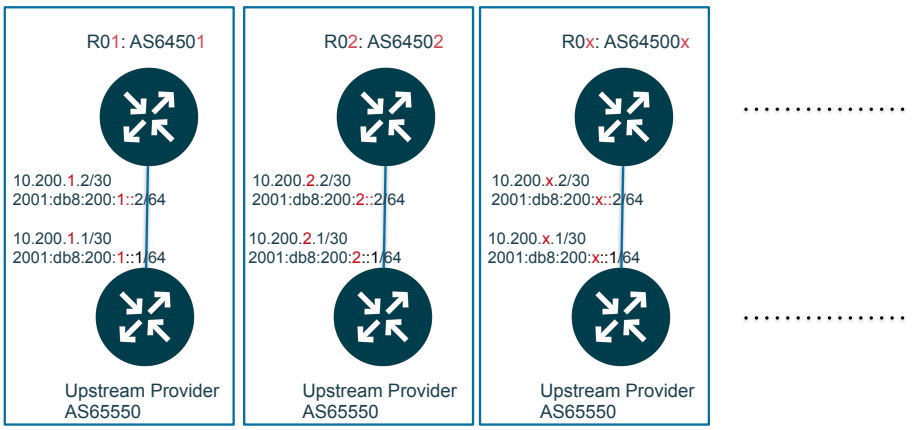


Figure 1: Network Setup

Figure 1 shows the network topology for this experiment. Participants are using AS6450x and are connected to AS65550 as a single upstream.

3 Slides

Configure eBGP to upstream AS65550

- We define empty route-maps
- Like in iBGP we put common configs into a peer-group
- some defaults are already pre-defined

```
route-map upstream-in permit 100
route-map upstream-out deny 100
!
router bgp 645xx
 neighbor upstream-v4 peer-group
 address-family ipv4 unicast
  neighbor upstream-v4 soft-reconfig inbound
  neighbor upstream-v4 route-map upstream-in in
  neighbor upstream-v4 route-map upstream-out out
  neighbor upstream-v4 activate
 exit-address-family
 neighbor 10.200.xx.1 remote-as 65550
 neighbor 10.200.xx.1 peer-group upstream-v4
```

Show commands:

- show bgp ipv4 summary
- show bgp ipv4 neighbor 10.200.xx.1
- show bgp ipv4
- show bgp ipv4 neighbor 10.200.xx.1 routes
- show bgp ipv4 neighbor 10.200.xx.1 received-routes
  why is one prefix you see not accepted?
- show bgp ipv4 neighbor 10.200.xx.1 advertised-routes

Configure eBGP for IPv6

- We use the same route-maps!
- But we need a different peer-group.

```
router bgp 645xx
 neighbor upstream-v6 peer-group
 address-family ipv6 unicast
  neighbor upstream-v6 route-map upstream-in in
  neighbor upstream-v6 route-map upstream-out out
  neighbor upstream-v6 soft-reconfig inbound
  neighbor upstream-v6 activate
 exit-address-family
 neighbor 2001:db8:200:xx::1 remote-as 65550
 neighbor 2001:db8:200:xx::1 peer-group upstream-v6
```

Show commands:

- show bgp ipv6 summary
- show bgp ipv6 neighbor 2001:db8:200:X::1
- show bgp ipv6
- show bgp ipv6 neighbor 2001:db8:200:X::1 routes
- show bgp ipv6 neighbor 2001:db8:200:X::1 received-routes
  why is one prefix you see not accepted?
- show bgp ipv6 neighbor 2001:db8:200:X::1 advertised-routes

4 Setup eBGP

4.1 Tasks:

- Define two peer groups for eBGP upstream neighbors: one for IPv4 and one for IPv6
- Set up eBGP sessions to AS65550
- Check if the sessions come up
- Look if you are receiving anything
- define (empty) route-maps for in and out
- Check again for received prefixes
- Try all commands listed below and look for anything unusual.

4.2 To set up eBGP you need to:

- configure two peer-groups for (in this case) upstream eBGP neighbors, one for IPv4 and one for IPv6
- configure (empty) filters
- configure neighbors which are member of these peer-groups

Do IPv4 only first and once it is up and running and you receive a prefix, add IPv6.

4.3 Information you need:

- Your AS number is 645xx
- Your upstream providers AS number is 65550
- Have a look at the interface connecting to your upstream. If your IP address is 10.200.1.2/30, then your neighbors IP is 10.200.1.1. For IPv6 your address is something like 2001:db8:200:2::2 and your upstream is 2001:db8:200:2::1.

4.4 To configure eBGP for IPv4 in config mode:

If you only configure IPv4 you do not to take care of *address family contexts* - see section about IPv6 about details.

neighbor <name> peer-group starts a peer group named *<name>*.

neighbor <name> route-map <rmname> in/out filters all prefixes in or out through route-map *<rmname>*. If the route-map is not defined (or if there is a typo in the name) it blocks everything in or out.

neighbor <name> soft-reconfiguration inbound allows you to see (via *show bgp ipv4/ipv6 neighbor <address> received-routes*) what the router receives before any incoming filters are processed.

neighbor <ip-address> remote-as <asnumber> sets up an neighbor with IP *<ip-address>* and AS *<asnumber>*. This command becomes active as soon as it is entered, so if you want to enter more config items for this peer, you either be quick or you need to shut it down until config is complete.

neighbor <ip-address> shutdown shuts down the BGP connection to this neighbor. Useful when you want to take some time configuring.

neighbor <ip-address> peer-group <name> makes a neighbor with IP <*ip-address*> a member of the peer group <*name*> (uses the peer-group as a template).

The remote AS is not configured in the peer-group but in the peer-group member, as you might want to re-use the same peer-group for more then one upstream provider.

4.5 To configure a route-map in config mode:

route-map <name> permit/deny <number> starts a route-map statement for a named route-map. Each route-map is referenced by <*name*> and statements are processed in order, starting with the lowest <*number*>. Inside each statement there can be zero to multiple *match* and *set* clauses. If the statement is evaluated true (all match conditions are true) permit or deny is executed. If not (matches evaluate false) the next statement in order is processed. If there are no statements left, there is an implicit deny at the end.

match <condition> An empty/missing *match* is always true. If there are multiples *match* statements, they all must be true for the entry to be true.

set <parameter> <value> If all matches are true (or if there are no match statements), all *set* statements are executed, the route-map is terminated and either permit or deny is executed (depending on the value in the statement header).

If the termination happens in a *permit* entry, the prefix currently processed is let through. If it is a *deny* statement, the prefix is discarded.

Route maps have many more possibilities, check your routers documentation or the online documentation of FRRouting at `http://docs.frrouting.org/en/latest/routemap.html`

4.6 Commands to check your eBGP session:

show bgp ipv4/ipv6 summary give you information about your peers

show bgp neighbor <ip-address> gives you detailed information about BGP neighbor with IP <*ip-address*>. If you omit the IP, it lists all neighbors very detailed (long output).

show bgp ipv4/ipv6 lists all IPv4 prefixes in BGP. Be aware that once you are on a real router connected to the real internet the output can be very long.

show bgp ipv4/ipv6 <prefix> gives you detailed BGP information about IP <*prefix*>

show bgp ipv4/ipv6 neighbor <ip-address> routes shows all valid IPv4 routes received from a specific neighbor (after the filters).

show bgp ipv4/ipv6 neighbor <ip-address> received-routes shows all IPv4 routes received from a specific neighbor (before filters are applied). Useful for checking if your filters work.

show bgp ipv4/ipv6 neighbor <ip-address> advertised-routes shows all IPv4 prefixes advertised to a specific neighbor.

4.7 Configure eBGP for IPv6:

As IPv6 is younger then the config syntax, they introduced the concept of “address-families” contexts. The commands within the address-family context relate to what is *transported* (IPv4- or IPv6-prefixes) *not* to the protocol of the BGP session.

To make things even more confusing, most of the IPv4 related commands are configured by default and not shown in the configuration.

Commands for configuration:

bgp ebgp-requires-policy prevents eBGP sessions without any policy (route-maps) configured from flooding your neighbors by installing a default-deny policy for in and out.

neighbor <name> peer-group starts a peer group. Configure outside of any context to define a new peer group.

neighbor <ipv6-address> remote-as <asnumber> defines an IPv6 neighbor. Also outside of any context.

address-family ipv6 unicast enters context configuration of IPv6 unicast.

address-family ipv4 unicast enters context configuration of IPv4 unicast.

exit-address-family leaves the current address-family context

neighbor <name> activate activates the peer group <*name*> in the current address-family context. Only necessary for IPv6, as peer-groups are activated for IPv4 by default.

neighbor <name> route-map <rmname> in/out defines route-maps for the peer-group *name* in the current context. Must be inside IPv6 address-family context for IPv6!

neighbor <name> soft-reconfiguration inbound switches on soft-reconfig for the current address-family. Must be defined within the address-family context.

neighbor <ipv6-address> peer-group <name> makes a neighbor with IPv6-address <*ipv6-address*> a member of the peer-group <*name*>. Must be configured within IPv6 address-family context.

no neighbor <ipv6-address> activate deactivates the session in the given context. For IPv6 transport, this should be configured in IPv4 context to avoid mismatching capabilities.

4.8 Commands to check BGP sessions:

show bgp summary shows all BGP sessions of all address families.

show bgp neighbors shows detailed information about all BGP neighbors independent of protocol

show bgp neighbors <address> show information about one neighbor. <*address*> can be an IPv4- or an IPv6-address.

show bgp neighbors <address> received-routes shows received prefixes from a neighbor *before* any filtering.

show bgp neighbors <address> routes shows prefixes learned from a neighbor (after filtering)

show bgp ipv4 shows all IPv4 prefixes in the BGP table

show bgp ipv6 shows all IPv6 prefixes in the BGP table

5 Solution

5.1 Solution for FRRouting

```
hostname r0x
router-id 172.16.1.x
!
router bgp 645xx
 no bgp default ipv4-unicast
 bgp ebgp-requires-policy
 neighbor upstream-v4 peer-group
 neighbor upstream-v6 peer-group
 neighbor 10.200.x.1 remote-as 65550
 neighbor 10.200.x.1 peer-group upstream-v4
 neighbor 2001:db8:200:x::1 remote-as 65550
 neighbor 2001:db8:200:x::1 peer-group upstream-v6
 !
 address-family ipv4 unicast
  neighbor upstream-v4 soft-reconfiguration inbound
  neighbor upstream-v4 route-map upstream-in in
  neighbor upstream-v4 route-map upstream-out out
  neighbor upstream-v4 activate
 exit-address-family
 !
 address-family ipv6 unicast
  neighbor upstream-v6 activate
  neighbor upstream-v6 soft-reconfiguration inbound
  neighbor upstream-v6 route-map upstream-in in
  neighbor upstream-v6 route-map upstream-out out
 exit-address-family
!
route-map upstream-out deny 100
route-map upstream-in permit 100
```

5.2 Solution for Cisco IOS

```
router bgp 645xx
  neighbor upstream-v4 peer-group
  neighbor upstream-v6 peer-group
  neighbor 10.200.x.1 remote-as 65550
  neighbor 10.200.x.1 peer-group upstream-v4
  neighbor 2001:db8:200:x::1 remote-as 65550
  neighbor 2001:db8:200:x::1 peer-group upstream-v6
  !
  address-family ipv4
    neighbor upstream-v4 route-map upstream-in in
    neighbor upstream-v4 route-map upstream-out out
    neighbor 10.200.x.1 activate
    no neighbor 2001:db8:200:x::1 activate
  exit-address-family
  !
  address-family ipv6
    neighbor upstream-v6 route-map upstream-in in
    neighbor upstream-v6 route-map upstream-out out
    neighbor 2001:db8:200:x::1 activate
  exit-address-family
!
route-map upstream-out deny 100
route-map upstream-in permit 100
```

5.3 Solution for Mikrotik

Mikrotik routers do not support something like peer-groups. So each peer needs all parameters[a].

```
/system identity
set name=r0x

/routing bgp instance
set default as=645xx router-id=172.16.1.x

/routing bgp peer
add in-filter=upstream-in name=upstreamA-v4 out-filter=upstream-out \
    remote-address=10.200.x.1 remote-as=65550
add address-families=ipv6 in-filter=upstream-in name=upstreamA-v6 out-filter=\
    upstream-out remote-address=2001:db8:200:x::1 remote-as=65550

/routing filter
add action=discard chain=upstream-out
add action=accept chain=upstream-in
```

[a] This configuration has only been tested for syntax.

5.4 Solution for Juniper

Example solution for Juniper routers[a]:

```
system {
  host-name R0x;
}
routing-options {
    autonomous-system 645xx;
}
protocols {
  bgp {
    local-as 645xx;
    group upstream-v4 {
      import upstream-in;
      family inet {
        unicast;
      }
      export upstream-out;
      neighbor 10.200.x.1 {
        peer-as 65550;
      }
    }
    group upstream-v6 {
      import upstream-in;
      family inet6 {
        unicast;
      }
      export upstream-out;
      neighbor 2001:db8:200:x::1 {
        peer-as 65550;
      }
    }
  }
}
policy-options {
  policy-statement upstream-in {
    then accept;
  }
  policy-statement upstream-out {
    then reject;
  }
}
```

[a] While syntactically correct, this configuration has not been tested