Experiment 4 - Working with BGP communities

DE-CIX Academy

Version 1.1 - Students Copy

1 Introduction

With BGP Communities you can add information to your prefixes. As communities itself are "just numbers", you need to define yourself what these numbers mean to you. In this experiment we will work with the different flavors of BGP communities (standard and extended).

2 Network Setup

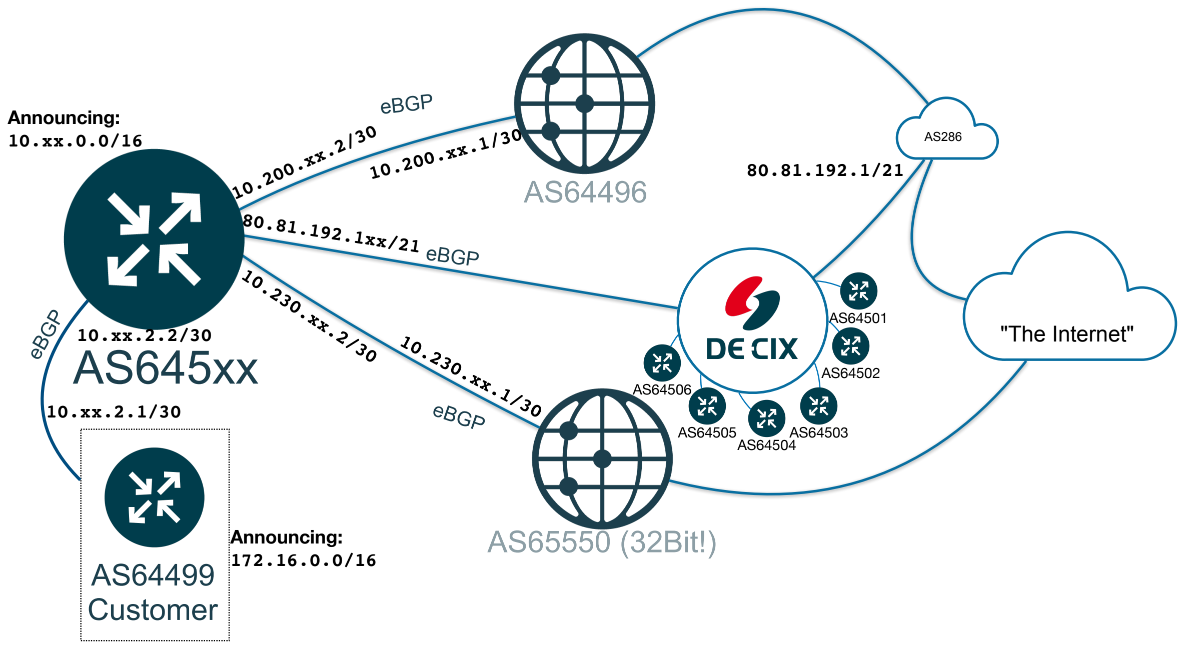

Figure 1: Network Setup

Figure 1 shows the network topology for this experiment. Two upstreams, one customer and everybody peers with each other.

3 Adding Informational Communities

To add communities to incoming prefixes you need to:

- Define a list of communities and their meaning
- Apply the communities at the entry of prefixes to your network

3.1 Task

Add informational communities to all prefixes in your routing table. The following information must be encoded:

- Received via peering, upstream or customer
- Location where prefix was received
- Router where prefix was received

Keep in mind that in a later task you must also:

- Define communities where you allow customers to influence routing
- Define communities to command your router to announce / not announce prefixes.

So what ever you define, leave space for expansion.

4 Community lists

To define lists of communities you can use community lists. There are lists for standard, extended and large communities. And for all three a standard or expanded type:

Standard type community lists do their matching on (complete) community numbers (like 64500:122).

Expanded community lists do their matching on regular expressions. That means that the communities to be checked are not seen as numbers but as strings. The usual regular expression matching can be used:

ˆ	matches start of string
$	matches end of string
[]	range of characters like [1-3]
.	any single character
_	underscore matches any whitespace, beginning or end of line, comma
*	matches zero or more occurrences of the pattern before
+	matches one or more occurrences of the pattern before
?	matches zero or one occurrences of the pattern before

Both list types can have *permit* and *deny* lines, lists are parsed sequentially and first match terminates. Result is the type of the line (permit or deny).

4.0.1 Examples

Match communities 64500:10000 or 64500:11000 (one of them must be there): As standard list:

```
bgp community-list standard Example permit 64500:10000
bgp community-list standard Example permit 64500:11000
bgp community-list standard Example deny
```

The last line is not really necessary as there is an implicit deny in Quagga, but that might not be the case in every router software.

Same as expanded list:

```
bgp community-list expanded Example permit _64500:1[01]000
```

So you see that using expanded lists your lists can be shorter (but sometimes also less readable). Also expanded community lists take up more CPU cycles to process. Still, most of the time they are more useful then standard lists.

Match my own communities (64500:.*) where

- the first digit is 4
- second digit is 2,3,6 or 7
- third to last is *not* 840

This can be used to filter outgoing prefixes, like announce to peers, but not announce in the US. As expanded list:

```
bgp community-list expanded PeeringOutNotUSA deny _64500:4[2367]840_
bgp community-list expanded PeeringOutNotUSA permit _64500:4[2367]..._
```

As standard list:

```
bgp community-list standard PeeringOutNotUSA deny 64500:42840
bgp community-list standard PeeringOutNotUSA deny 64500:43840
bgp community-list standard PeeringOutNotUSA deny 64500:46840
bgp community-list standard PeeringOutNotUSA deny 64500:47840
bgp community-list standard PeeringOutNotUSA permit 64500:42000
bgp community-list standard PeeringOutNotUSA permit 64500:43000
bgp community-list standard PeeringOutNotUSA permit 64500:46000
bgp community-list standard PeeringOutNotUSA permit 64500:47000
bgp community-list standard PeeringOutNotUSA permit 64500:42001
...
```

You see, this does not really work. So if you want to work with some wildcards, you need expanded lists (not all routers have them!).

5 Config statements you may need

5.1 Community lists

bgp community-list (<1-99>|standard <name>) (permit|deny) <AA:NN>... Adds lines to a standard community list for orignal communities.

bgp community-list (<300-500>|expanded <name>) (permit|deny) <regular expression> adds a line with a regular expression to an expanded community list.

bgp extcommunity-list (<1-99>|standard <name>) (permit|deny) (rt|soo) (<aaaa:nn>|<aa:nnnn>)... adds a *route target* or *site of origin* extended community rule to a standard list. One of *<aaaa>* or *<nnnn>* can be 32 bits long.

bgp extcommunity-list (<100-500>|expanded <name>) (permit|deny) <regular expression> defines a line in an expanded list for extended communities.

bgp large-community-list (<1-99>|standard <name>) (permit|deny) <aaaa:bbbb:cccc>... adds a line to a standard large-community list. All of *<aaaa>*, *<bbbb>* and *<cccc>* are 32 bits long.

bgp large-community-list (<100-500>|expanded <name>) (permit|deny) <regular expression> defines a line in an expanded community list for large communities.

Note that standard and expanded lists share the same name-space, so you the *<name>* you are using must be unique.

5.2 Within route-maps

5.2.1 Set statements

set communitity <AA:NN>... Clears all existing and sets a list of communities.

set communitiy <AA:NN>...additive Keeps existing communities in place and simply adds the listed communities. *AA* is a 16-bit AS number, *NN* some 16-bit number.

set extcommunity rt ASN:NN... Clears all existing extended communities and sets new ones. There is no “additive” command for extended communties. *ASN* is a 32-bit AS number, *NN* some 16-bit number.

set large-community AA:BB:CC...*[additive]* Similar to the original communities, you can set or add (using *additive* keyword) large communities. All of *AA,BB,CC* are 32-bit numbers.

5.2.2 Match statements

In route-maps matches against communities attached to prefix announcements can be made. For this, you need to define a community list first (can be any list, standard or expanded) and then match against this list:

match community *<listname>* parses the list named *<listname>* and checks if one of the communities attached to the prefix matches.

match community *<listname>* exact-match matches only if there is an exact match between the prefix and the community-list.

match extcommunity *<listname>* same matching for extended communities. There is not “exact-match” keyword here.

match large-community *<listname>* same for large community lists. Also no “exact-match” keyword here.

5.2.3 Deleting communities

For removing communities from prefix announcements also route-maps have to be used. The communities to be removed have to be defined in a community-list (standard or expanded):

set comm-list *<listname>* delete

set large-comm-list *<listname>* delete removes all communities matching the list “<listname>”.

There is no command to remove extended communities.

6 show commands

6.0.1 community lists

show bgp (community-list|extcommunity-list|large-community-list) [<1-500>|<name>] shows either all or a named or numbered community list.

6.0.2 Prefixes

show bgp (ipv4|ipv6) community <aa:nn>…[exact-match] list prefixes with communities *<aa:nn>* attached to them.

show bgp (ipv4|ipv6) large-community <aaaa:bbbb:cccc>…[exact-match] list prefixes with large communities *<aaaa:bbbb:cccc>* attached to them.

show bgp (ipv4|ipv6) community-list (<1-500>|<name>) [exact-match] shows bgp prefixes matching the standard or expanded community list *<name>*.

show bgp (ipv4|ipv6) large-community-list (<1-500>|<name>) shows bgp prefixes matching the standard or expanded large community list *<name>*.

7 Task: Control announcement of prefixes using communities

You receive prefixes from four different kinds of sources:

- From your customer, AS64499
- From your upstream providers
- From a number of peerings
- From yourself, your own netblock that you announce

Your task is to implement a pretty standard routing policy:

- Announce your own and your customers prefixes to all upstreams and peers

- Announce prefixes you receive from upstream and peers to your customer only.

For this you need to:

1. Define a list of communities to control announcement of prefixes (keep in mind that any combination of announcing to peering, upstream or customer must be possible)
2. Tag all prefixes in your routing table with the needed community/ies.
3. Filter all announcements to the outside via eBGP according to the communities set.

8 Task: Allow your customer to control your announcement

As a good transit provider you offer your customers the control over their prefixes. Define and implement the following using BGP communities:

- Allow customers to control announcing their prefixes to upstream, peering or other customers (or and combination of them)
- Prevent customers from sending you any other communities except the ones you allow them for announcement control.

A Initial Router Configuration

```
hostname rXX
!
ip route 10.XX.0.0/16 Null0
!
interface dummy0
 ip address 10.XX.1.1/32
!
router bgp 645XX
 neighbor customer peer-group
 neighbor peering peer-group
 neighbor upstream peer-group
 neighbor 10.XX.2.1 remote-as 64499
 neighbor 10.XX.2.1 peer-group customer
 neighbor 80.81.192.1 remote-as 286
 neighbor 80.81.192.1 peer-group peering
 neighbor 80.81.192.102 remote-as 64502
 neighbor 80.81.192.102 peer-group peering
 neighbor 80.81.192.103 remote-as 64503
 neighbor 80.81.192.103 peer-group peering
 neighbor 80.81.192.104 remote-as 64504
 neighbor 80.81.192.104 peer-group peering
 neighbor 80.81.192.105 remote-as 64505
 neighbor 80.81.192.105 peer-group peering
 neighbor 80.81.192.106 remote-as 64506
 neighbor 80.81.192.106 peer-group peering
 neighbor 10.200.XX.1 remote-as 65550
 neighbor 10.200.XX.1 peer-group upstream
 neighbor 10.230.XX.1 remote-as 64496
 neighbor 10.230.XX.1 peer-group upstream
 !
 address-family ipv4 unicast
  network 10.XX.0.0/16
  neighbor customer next-hop-self
  neighbor customer soft-reconfiguration inbound
```

```
    neighbor customer route-map customer-in in
    neighbor customer route-map customer-out out
    neighbor peering next-hop-self
    neighbor peering soft-reconfiguration inbound
    neighbor peering route-map peering-in in
    neighbor peering route-map peering-out out
    neighbor upstream next-hop-self
    neighbor upstream soft-reconfiguration inbound
    neighbor upstream route-map upstream-in in
    neighbor upstream route-map upstream-out out
 exit-address-family
!
bgp as-path access-list my-as permit ^$
!
route-map upstream-out permit 100
 match as-path my-as
!
route-map upstream-in permit 100
 set local-preference 10
!
route-map peering-out permit 100
 match as-path my-as
!
route-map peering-in permit 100
 set local-preference 1000
!
route-map customer-out permit 100
!
route-map customer-in permit 100
 set local-preference 10000
!
end
```

B Final Router Configuration

```
hostname rXX
!
ip route 10.XX.0.0/16 Null0
!
interface dummy0
 ip address 10.XX.1.1/32
!
router bgp 645XX
 neighbor customer peer-group
 neighbor peering peer-group
 neighbor upstream peer-group
 neighbor 10.XX.2.1 remote-as 64499
 neighbor 10.XX.2.1 peer-group customer
 neighbor 80.81.192.1 remote-as 286
 neighbor 80.81.192.1 peer-group peering
 neighbor 80.81.192.102 remote-as 64502
 neighbor 80.81.192.102 peer-group peering
 neighbor 80.81.192.103 remote-as 64503
 neighbor 80.81.192.103 peer-group peering
 neighbor 80.81.192.104 remote-as 64504
 neighbor 80.81.192.104 peer-group peering
 neighbor 80.81.192.105 remote-as 64505
 neighbor 80.81.192.105 peer-group peering
 neighbor 80.81.192.106 remote-as 64506
 neighbor 80.81.192.106 peer-group peering
 neighbor 10.200.XX.1 remote-as 65550
 neighbor 10.200.XX.1 peer-group upstream
 neighbor 10.230.XX.1 remote-as 64496
 neighbor 10.230.XX.1 peer-group upstream
 !
 address-family ipv4 unicast
  network 10.XX.0.0/16 route-map own-networks
  neighbor customer next-hop-self
  neighbor customer soft-reconfiguration inbound
  neighbor customer route-map customer-in in
  neighbor customer route-map customer-out out
  neighbor peering next-hop-self
  neighbor peering soft-reconfiguration inbound
  neighbor peering route-map peering-in in
  neighbor peering route-map peering-out out
  neighbor upstream next-hop-self
  neighbor upstream soft-reconfiguration inbound
  neighbor upstream route-map upstream-in in
  neighbor upstream route-map upstream-out out
 exit-address-family
!
bgp as-path access-list my-as permit ^$
!
bgp community-list expanded announce-to-customer permit _645XX:4[1357]000
bgp community-list expanded announce-to-peering permit _645XX:4[2367]000
bgp community-list expanded announce-to-upstream permit _645XX:4[4567]000
bgp community-list expanded my-communities permit _645XX:.*
bgp large-community-list expanded my-largecommunities permit _645XX:.*:.*
!
route-map upstream-in permit 50
 match community my-communities
 set comm-list my-communities delete
 on-match next
!
route-map upstream-in permit 60
 match large-community my-largecommunities
 set large-comm-list my-largecommunities delete
 on-match next
!
route-map upstream-in permit 100
 set community 64501:41000 additive
 set extcommunity rt 64501:21000
 set large-community 64501:2:1000 additive
 set local-preference 10
!
route-map peering-in permit 50
 match community my-communities
 set comm-list my-communities delete
 on-match next
!
route-map peering-in permit 60
 match large-community my-largecommunities
 set large-comm-list my-largecommunities delete
 on-match next
```

```
!
route-map peering-in permit 100
 set community 64501:41000 additive
 set extcommunity rt 64501:23000
 set large-community 64501:2:3000 additive
 set local-preference 1000
!
route-map customer-in permit 50
 match community my-communities
 set comm-list my-communities delete
 on-match next
!
route-map customer-in permit 60
 match large-community my-largecommunities
 set large-comm-list my-largecommunities delete
 on-match next
!
route-map customer-in permit 100
 set community 64501:47000 additive
 set extcommunity rt 64501:24000
 set large-community 64501:2:4000 additive
 set local-preference 10000
!
route-map customer-out permit 100
 match community announce-to-customer
!
route-map peering-out permit 100
 match community announce-to-peering
!
route-map upstream-out permit 100
 match community announce-to-upstream
!
route-map own-networks permit 100
 set community 64501:47000
!
line vty
!
end
```

C Slides

BGP Communities - informational

Set communities that show from where you have received a prefix:

Received from	Community value
Upstream	645xx:21000
Peer	645xx:23000
Customer	645xx:24000

```
route-map upstream-in permit 100
  set community 645xx:21000 additive

route-amp peering-in permit 100
  set community 645xx:23000 additive

route-map customer-in permit 100
  set community 645xx:24000 additive
```

Show commands:

- show bgp ipv4 129.13.0.0/16
- show bgp ipv4 community 645*xx*:23000

BGP Extended Communities

Set communities that show from where you have received a prefix:

Received from	Community value
Upstream	rt:645xx:21000
Peer	rt:645xx:23000
Customer	rt:645xx:24000

```
route-map upstream-in permit 100
  set extcommunity rt 645xx:21000

route-map peering-in permit 100
  set extcommunity rt 645xx:23000

route-map customer-in permit 100
  set extcommunity rt 645xx:24000
```

- show bgp ipv4 129.13.0.0/16
- there is no "show bgp ipv4 extcommunity" command!

BGP Large Communities

Set communities that show from where you have received a prefix:

Received from	Community value
Upstream	645xx:2:1000
Peer	645xx:2:3000
Customer	645xx:2:4000

```
route-map upstream-in permit 100
  set large-community 645xx:2:1000 additive

route-amp peering-in permit 100
  set large-community 645xx:2:3000 additive

route-map customer-in permit 100
  set large-community 645xx:2:4000 additive
```

- show bgp ipv4 129.13.0.0/16
- show bgp ipv4 large-community 645xx:2:4000

Community Lists - Standard Community Lists

Standard Community Lists

- Match on (complete) community numbers
- Exist for all three kinds of Communities
- Have permit/deny lines - first match terminates with result

Define three lists:

```
bgp community-list standard from-peering permit 645xx:23000

bgp community-list standard from-upstream permit 645xx:21000

bgp community-list standard from-customer permit 645xx:24000
```

Large and extended community lists look very similar (no need to define them):

```
  bgp extcommunity-list standard ...

  bgp large-community-list standard ...
```

Show commands:

- show bgp ipv4 community-list from-peering
- show bgp ipv4 community-list from-customer

Expanded Community Lists

- Use *regular expressions* for matching
- Communities are seen as strings (characters) - not numbers
- See table below

```
bgp community-list expanded my-communities permit _645xx:.*
```

Symbol/Character	Matches
0123456789	number as written
^	beginning of line
$	end of line
.	any character
_	any whitespace
[123]	1 or 2 or 3
*	zero or more occurrences of preceding pattern
+	one or more occurrences of preceding pattern
?	zero or one occurrences of preceding pattern

BGP Communities - Scrubbing

- Delete unwanted communities incoming
- We allow communities from customers
  - But not from upstream or peers
  - And also from customers only *some* communities
- All other (not our) communities should be kept

```
bgp community-list expanded my-communities permit _645xx:.*

route-map upstream-in permit 50
  match community my-communities
  set comm-list my-communities delete
  on-match next

route-map peering-in permit 50
  match community my-communities
  set comm-list my-communities delete
  on-match next

route-map customer-in permit 50
  match community my-communities
  set comm-list my-communities delete
  on-match next
```

- show bgp ipv4 44.5.0.0/16

BGP Communities - Action!

- We *set* communities when we receive prefixes
- These communities steer where prefixes are announced

For this we need an announcement policy:

Received from	Announced to
Customers	Customers, Peers, Upstream
Peers	Customers
Upstream	Customers

We use these communities:

Announce to	Communitiy	Significant Digit
Customers	645xx:41000	1
Peers	645xx:42000	2
Upstream	645xx:44000	4
Peers + Upstream	645xx:46000	6 = 2 + 4
Customers + Peers + Upstream	645xx:47000	7 = 1 + 2 + 4

```
route-map customer-in permit 100
  set community 645xx:47000 additive

route-map upstream-in permit 100
  set community 645xx:41000 additive

route-map peering-in permit 100
  set community 645xx:41000 additive
```

BGP Communities - Action!

- We need to change our outgoing filters
- For this we define community lists
- One list for each class of announcement

See this table for what we announce where:

Announce to	Communitiy	Meaning
Customers	645xx:41000	Customers only
	645xx:43000	Customers+Peers
	645xx:45000	Customers+Upstream
	645xx:47000	All
Peers	645xx:42000	Peers only
	645xx:43000	Peers+Customers
	645xx:46000	Peers+Upstream
	645xx:47000	All
Upstream	645xx:44000	Upstream only
	645xx:45000	Upstream+Customers
	645xx:46000	Upstream+Peers
	645xx:47000	All

Remember *regular expressions*?

We can build nice and short expanded BGP community lists for matching:

```
bgp community-list expanded announce-to-customer permit _645xx:4[1357]000

bgp community-list expanded announce-to-peering permit _645xx:4[2367]000

bgp community-list expanded announce-to-upstream permit _645xx:4[4567]000
```

- show bgp ipv4 community-list announce-to-upstream
- show bgp ipv4 community-list announce-to-peering
- show bgp ipv4 community-list announce-to-customer

BGP Communities - Action!

- We now use these community list in route-map *match* statements
- We replace current route-maps for outgoing announcements

```
route-map customer-out permit 100
  match community announce-to-customer

route-map peering-out permit 100
  match community announce-to-peering
  no match  as-path my-as

route-map upstream-out permit 100
  match community announce-to-upstream
  no match as-path my-as
```

Show commands:

- show bgp ipv4 neighbors 10.230.xx.1 advertised-routes
- show bgp ipv4 neighbors 80.81.192.1yy advertised-routes
- show bgp ipv4 neighbors 10.xx.2.1 advertised-routes

Advertising our own network

- We have a static route for 10.xx.0.0/16
- We inject it into BGP using a network statement
- But it is not announced anywhere
  - Because no communities are attached to it
- We can re-use the customer-in route-map
  - Or define a separate one (to avoid side effects)

```
route-map own-networks permit 100
set community 645xx:47000

router bgp 645xx
  address-family ipv4 unicast
  network 10.xx.0.0/16 route-map own-networks
```

Show commands:

- show bgp ipv4 10.xx.0.0/16
- show bgp ipv4 neighbors 10.230.xx.1 advertised-routes
- show bgp ipv4 neighbors 80.81.192.1yy advertised-routes
- show bgp ipv4 neighbors 10.2.xx.1 advertised-routes