BGP

The Border Gateway Protocol (BGP) is an inter-Autonomous System routing protocol.

NYC Mesh no longer uses BGP within the mesh between neighbors / members. Use of BGP within the mesh was too static for the changing network, lacked some "automatic" properties, and made it difficult to train new people.

Use outside NYC Mesh

BGP is a popular dynamic routing protocol as it is relatively simple to configure, scales well and enjoys support across multiple hardware and software vendors. The internet uses BGP to interconnect all the thousands of companies, ISPs, and Exchanges that make up the internet.

Use within NYC Mesh

NYC Mesh uses BGP to connect to the internet at Supernodes. In fact .. it is a requirement of a Supernode.
With BGP, we are able to connect our "Autonomous System" (our organization's ID) with other organizations, such as ISPs, and participate in the internet. By doing this, we carry our organization's IP addresses along with us and create many redundant connections to the internet.

Supernodes require BGP to connect to the internet because NYC Mesh has dedicated IP Addresses for use by our organization. In order to maintain proper functionality with the internet, we need to ensure we maintain a BGP connection at critical internet connection points, or, Supernodes.

Old Instructions

It is still possible to connect a BGP-only speaking device to the mesh, please coordinate with the larger group to do so, as it requires some planning.

Expand to see some older BGP information that is kept for informational purposes and the eventuality that BGP is needed for some devices

Expand to view details...

Communities

BGP communities can be used to classify routes that are imported or exported by an AS. Some definitions generally agreed upon by BGP speakers within the mesh are listed below. They are primarily used for interpreting the "quality" of various routes to the internet.

Community	Meaning	Suggested interpretation
65000:1001	Internet connected by NYC Mesh	Set local preference to 130
65000:1002	Internet connected by a fast, neutral 3rd party	Set local preference to 110
65000:1003	Internet connected by a fast, non-neutral 3rd party	Set local preference to 100
65000:1004	Internet connected by a slow, non-neutral 3rd party	Set local preference to 90
65000:1005	Internet connected by a slow, NATed or possibly compromised 3rd party	Set local preference to 80

Prefix lists

IPv4 and IPv6 prefix lists that BGP speakers within the mesh commonly filter on (for import and export) are listed below:

IPv4

Prefix (Bird notation)	Action
199.167.59.0/24{24,32}	Allow
10.0.0.0/8{22,32}	Allow
0.0.0.0/0	Allow
All others	Deny

IPv6

Prefix (Bird notation)	Action
2620:12d:400d::/48{48,64}	Allow

Prefix (Bird notation)	Action
fdff:1508:6410::/48{64,128}	Allow
::/0	Allow
All others	Deny

How to get an ASN or IP allocation

Currently the mesh uses a spreadsheet to keep track of allocated resources. The process will be automated soon, but in the mean time please contact an existing member via Slack or email to have them help you acquire an ASN and IPv4 and/or IPv6 resources.

Examples

Some configuration examples for BGP implementations known to be in use within NYC Mesh today are listed below in no particular order.

Bird

Bird is an open source routing daemon with support for a number of different routing protocols including BGP.

****Expand Bird Example****

```
log stderr all;

router id 10.70.x.1;

function is_mesh_prefix_v4 () { return net ~ [ 199.167.59.0/24{24,32}, 10.0.0.0/8{22,32}, 0.0.0.0/0 ]; }

function is_mesh_prefix_v6 { return net ~ [ 2620:12d:400d::/48{48,64}, fdff:1508:6410::/48{64,128}, ::/0 ]; }
```

function set_local_pref () { if (65000,1001) ~ bgp_community then bgp_local_pref = 130; if (65000,1002) ~ bgp_community then bgp_local_pref = 110; if (65000,1003) ~ bgp_community then bgp_local_pref = 100; if (65000,1004) ~ bgp_community then bgp_local_pref = 90; if (65000,1005) ~ bgp_community then bgp_local_pref = 80; }

filter is_not_deviceroute { if source = RTS_DEVICE then reject; accept; }

filter mesh_import_v4 { if ! is_mesh_prefix_v4() then reject; set_local_pref(); accept; }

filter mesh_export_v4 { if ! is_mesh_prefix_v4() then reject; if ifname = "eth0" then bgp_community.add((65000,1005)); accept; }

filter mesh_import_v6 { if ! is_mesh_prefix_v6() then reject; set_local_pref(); accept; }

filter mesh_export_v6 { if ! is_mesh_prefix_v6() then reject; if ifname = "eth0" then bgp_community.add((65000,1005)); accept; }

protocol device { scan time 10; }

protocol direct { ipv4; interface "br0" "eth0"; }

protocol kernel { scan time 10; ipv4 { export filter is_not_deviceroute; }; }

protocol kernel { scan time 10; ipv6 { export filter is_not_deviceroute; }; }

template bgp meshpeer { local 10.70.x.1 as 65xxx; hold time 15; keepalive time 5; ipv4 { next hop self; import filter mesh_import_v4; export filter mesh_export_v4; }; ipv6 { next hop self; import filter mesh_import_v6; export filter mesh_export_v6; }; }

protocol bgp n1234 from meshpeer { neighbor 10.70.x.y as 65yyy; }

```
</details>


### [UBNT/EdgeOS](https://www.ubnt.com/products/#edgemax)
UBNT's EdgeOS was forked from Vyatta, which in turn borrows from
[Quagga](https://www.nongnu.org/quagga/).
<details>
<summary>**Expand for UBNT/EdgeOS Example**</summary>
```

protocols { bgp 65xxx { neighbor 10.70.x.y { description n1234 nexthop-self remote-as 65yyy route-map { export nycmeshexport import nycmeshimport } soft-reconfiguration { inbound } } network 10.70.x.0/24 { } network 199.167.59.x/32 { } parameters { router-id 10.70.x.1 } timers { holdtime 15 keepalive 5 } redistribute { static { route-map nycmeshexportIspDefault } } } static { route 10.70.x.0/24 { blackhole { } } } } policy {

community-list 101 { rule 10 { action permit regex 65000:1001 } } community-list 102 { rule 10 { action permit regex 65000:1002 } } community-list 103 { rule 10 { action permit regex 65000:1003 } } community-list 104 { rule 10 { action permit regex 65000:1004 } } community-list 105 { rule 10 { action permit regex 65000:1005 } } prefix-list nycmeshprefixes { rule 10 { action permit ge 22 le 32 prefix 10.0.0.0/8 } rule 20 { action permit ge 24 le 32 prefix 199.167.56.0/22 } rule 30 { action permit prefix 0.0.0.0/0 } } route-map nycmeshexport { rule 10 { action permit match { ip { address { prefix-list nycmeshprefixes } } } } rule 20 { action deny } } route-map nycmeshexportIspDefault { rule 10 { action permit match { interface eth0 } set { community "65000:1005 additive" } } rule 20 { action deny } } route-map nycmeshimport { rule 10 { action permit match { community { community-list 101 } } set { local-preference 130 } } rule 20 { action permit match { community { community-list 102 } } set { local-preference 110 } } rule 30 { action permit match { community { community-list 103 } } set { local-preference 100 } } rule 40 { action permit match { community { community-list 104 } } set { local-preference 90 } } rule 50 { action permit match { community { community-list 105 } } set { local-preference 80 } } rule 60 { action permit match { ip { address { prefix-list nycmeshprefixes } } } } rule 70 { action deny } } }

```
</details>

### [Mikrotik/RouterOS](https://wiki.mikrotik.com/wiki/Manual:TOC)
Mikrotik's RouterOS has its own closed source BGP implementation.
**TODO**

### [OpenBGPD](http://www.openbgpd.org/)
An example of a working configuration, abeit without BGP community rules, is available
[here](https://github.com/bongozone/kibble/blob/master/src/etc/bgpd.conf).

<details>
```

Revision #2
Created 9 December 2023 04:39:47 by Willard Nilges
Updated 15 December 2023 20:45:24 by Andy