NOCTION
NETWORK INTELLIGENCE

Using PeeringDB to set up your Internet Exchange peering

www.noction.com

Introduction

When networks exchange traffic without having a customer-provider relationship, this is called peering. There's private and public peering. Private peering happens over a direct interconnect between the two networks involved.

Public peering happens over an internet exchange (IX). An IX is nothing more than a shared subnet that all the members/customers connect to. So, each connected network connects one or more routers to the shared Ethernet network. Peering then happens by setting up BGP sessions to routers belonging to other networks that are also connected to the IX.

Once you start looking at connecting to one or more IXes, you'll soon find that the larger ones have ***many*** members. Fortunately, most IXes have route servers. When you peer with the IX's route server(s), you automatically peer with all other members who also peer with the route server(s). So that's a good start. But typically, you'll also want to peer with other networks that don't peer with route servers. This involves sending out large numbers of emails to potential peering partners as outlined in the Peering Request Etiquette blog post. Then, if everything happens according to plan, you'll get a message back that the other network also wants to peer with you, and peering can commence.

At that point, you'll have to configure one or more routers with the right information to set up a BGP session towards your new peering partner's router. It is of course perfectly possible to find the contact info of prospective peering partners on the website of the IX or IXes you're connected to, and then exchange the BGP session details through email. However, in practice this is a lot of work because contact info on the IX websites is often incomplete, and the BGP session details in email are unstructured, so there's a lot of copy/paste involved.

Finding peering partners through PeeringDB.com

A better way to handle this is through PeeringDB.com.

PeeringDB is a website that has information about internet exchanges and the networks that connect to those IXes. For each network, there's a lot of information that is relevant to prospective peering partners:

- Mostly inbound traffic (access ISP) or outbound traffic (content network)
- Numbers of IPv4 and IPv6 prefixes announced

- Geographic scope: global, regional or smaller
- (Sometimes) traffic levels
- At which IXes the network is present
- Peering policy: open, selective or restrictive, in the latter cases often with a description of the policy
- Contact information

And, once you've agreed to peer, for each IX there's the AS number (yes, some networks use different AS numbers in different locations!) as well as their router's IPv4 and IPv6 addresses.

So, if a peering partner has their correct information filled in on PeeringDB, you can use the website to find all the information you need to configure your BGP sessions. Well, except for your BGP MD5 passwords. You find all this information on PeeringDB without registering an account, but obviously, it's a good idea to sign up and fill in your own information for others to find. Then, rather than list the relevant information in your peering request emails, you can simply list a link to your PeeringDB information.

However, searching PeeringDB for information and then copying that information to a router configuration in order to set up BGP is still inefficient and error-prone. A better way to do this is to retrieve the desired information directly from PeeringDB using SQL queries or API calls. Unfortunately, PeeringDB no longer supports querying the database using SQL, so it's necessary to interact with the database through the PeeringDB REST API, which requires more steps to reach the same results.

Setting up a system that queries PeeringDB requires a good amount of work up front, but once that's done, creating router configurations becomes much easier. As a service to the community, this ebook contains some simple PHP scripts to accomplish this.

The API

The PeeringDB database can be queried using a REST API. REST allows a client to request information from a server over HTTP or HTTPS. The server then returns the requested information in JSON format.

Our example script is written in PHP. PHP is mostly a web scripting language, but it can also be used on the command line. Our script uses the CURL, the Client URL Library, to perform HTTPS requests. On Linux/Unix systems installing PHP and the CURL library is usually quite easy, and on macOS they are installed by default. Please download the script here, and save it with the extension .php:

Download peerlist.php.txt script

Once downloaded, you can run the script as follows:

```
php peerlist.php 1200
```

The script will then query PeeringDB to find out at which internet exchanges AS1200 is present. As AS1200 is the Autonomous System of AMS-IX, AS1200 is present at AMS-IX with two routers, so the script shows:

```
Exchange    Mbps  RS  IPv4 address      IPv6 address
AMS-IX      1000      80.249.208.1      2001:7f8:1::a500:1200:1
AMS-IX      1000      80.249.209.1      2001:7f8:1::a500:1200:2

Total available bandwidth: 2 Gbps at 2 internet exchange
locations.
```

(Obviously you'll want to use your own AS number—you do have your AS registered with PeeringDB, don't you?)

The next step is for the script to obtain a list of all networks present at the internet exchanges found above. These are the potential peers, and it's usually a long list. This is the relatively modest output for Phyxia, AS35627:

```
php peerlist.php 35627

AS35627 has a presence at:

Exchange    Mbps  RS  IPv4 address      IPv6 address
FreeBIX     1000      195.85.203.23     2001:7f8:1b::3:5627:1
GN-IX       100       193.111.172.30    2001:7f8:31:0:5:3:5627:1

Total available bandwidth: 1.1 Gbps at 2 internet exchange
locations.

Potential peers:

FreeBIX, Packet Clearing House, 3856, Open
FreeBIX, Trance Nation, 34806, Open
FreeBIX, Connexeon, 35821, Open
FreeBIX, MAC Telecom, 50857, Open
FreeBIX, Netlog N.V., 41471, Open
FreeBIX, FR-NIC-DNS (AFNIC / NIC-France), 2484, Open
FreeBIX, HousingCenter, 28707, Open
FreeBIX, Hermes Telecom Group, 6824, Selective
GN-IX, De Kooi, 16318, Open
GN-IX, OpenPeering, 20562, Open
GN-IX, MetaMicro Automatisering BV, 41037, Open
GN-IX, Duocast BV, 31477, Open
GN-IX, CJ2 Hosting, 39704, Open
GN-IX, Plusine ICT, 198508, Open
GN-IX, gnTel, 41153, Open
```

The RS field indicates whether a network is connected to the route servers of the internet exchange, with 1 meaning yes and empty or 0 meaning no.

The script can also be run with the additional keyword csv, and then it will output a comma separated values file which can easily be imported into a database or spreadsheet. The output then looks like this:

```
php peerlist.php 35627 csv

ix,name,website,asn,info_traffic,info_ratio,info_scope,policy_
url,policy_general,peeringdb_url

“FreeBIX”,”Packet Clearing House”,”http://www.pch.
net/”,3856,”1-5Gbps”,”Balanced”,”Global”,””,”Open”,”https://
www.peeringdb.com/net/286”

“FreeBIX”,”Trance Nation”,”http://www.trancenation.
be”,34806,”0-20 Mbps”,”Balanced”,”Europe”,”http://www.
trancenation.be/?page=PeeringPolicy”,”Open”,”https://www.
peeringdb.com/net/905”
```

This script should provide a simple example of how to interact with the PeeringDB API and it can provide a start for managing peering relationships.

Generating router configurations: base configuration

Once you’ve found a network willing to peer, you have to configure your routers. Doing this manually takes time and easily leads to mistakes. So here is a script to generate router configurations to generate peering configurations.

However, our script only generates the configuration lines that are required to set up a peering. Before that can happen, a lot of configuration needs to be in place already. This is the base configuration in our example:

```
!
router bgp 65065
 neighbor gnix-peers-ipv4 peer-group
 neighbor gnix-peers-ipv4 description IPv4 peers on GN-IX
 neighbor gnix-peers-ipv6 peer-group
 neighbor gnix-peers-ipv6 description IPv6 peers on GN-IX
!
 address-family ipv4 unicast
 neighbor gnix-peers-ipv4 activate
 neighbor gnix-peers-ipv4 maximum-prefix 100000
 neighbor gnix-peers-ipv4 prefix-list import in
 neighbor gnix-peers-ipv4 prefix-list export out
 neighbor gnix-peers-ipv4 route-map localpref110 in
 no neighbor gnix-peers-ipv6 activate
 exit-address-family
!
 address-family ipv6
 no neighbor gnix-peers-ipv4 activate
 neighbor gnix-peers-ipv6 activate
```

```
 neighbor gnix-peers-ipv6 maximum-prefix 10000
 neighbor gnix-peers-ipv6 prefix-list import in
 neighbor gnix-peers-ipv6 prefix-list export out
 neighbor gnix-peers-ipv6 route-map localpref110 in
 exit-address-family
!
route-map localpref110 permit 10
 set local-preference 110
!
```

What we do here is define two peer groups for peers on the GNIX exchange: one for IPv4 and one for IPv6. The peergroup name must match the name of the internet exchange in PeeringDB, but converted to lower case and with non-alphanumeric characters removed and followed with -peers- and then the IP version. So GN-IX becomes gnix-peers-ipv4 and gnix-peers-ipv6.

As is usual with eBGP, the IPv4 sessions are only activated for IPv4 and the IPv6 sessions only for IPv6. Note that there are no issues using the same name for the IPv4 and IPv6 prefix lists, as those are define with ip prefix-list import … and ipv6 pref-list import … respectively. Both peer groups refer to the same route map localpref110 that is used to increase the local preference for peers to 110, so peering routes are preferred over routes learned from transit providers, which presumably have the default local preference of 100.

NOTE: When you take a look at the configuration, it may look different, as IPv4-specific configuration is often placed directly under router bgp … rather than under the address-family ipv4 unicast heading.

Generating router configurations: the config file

With the base configuration in place, we're almost ready to start generating peering configurations. This works by combining the details for our own AS with the details of another AS. We don't request our own information from PeeringDB each time we generate a peering configuration, both because the PeeringDB API isn't very fast and also so we can customize our own information. Rather, we set up a configuration file, which looks like this:

```
<?
$rtr_name[0] = "freebix-rtr";
$rtr_ixid[0] = 58;
$rtr_v4[0] = 1;
$rtr_v6[0] = 1;
$rtr_type[0] = "cisco";
#
$rtr_name[1] = "gnix-rtr";
$rtr_ixid[1] = 76;
$rtr_v4[1] = 1;
$rtr_v6[1] = 1;
$rtr_type[1] = "cisco";
```

```
#
$num_rtrs = 2;
$ixes = ":freebix:gnix:";
$local_as = 65065;
```

This file needs to exist under the name *config.txt* in the directory from where we run our script. The *rtr_name* line is the name for the router. In this case, there are two routers connected to two internet exchanges. However, if one router connects to both exchanges, simply change the name so it's the same for both exchanges.

In order to save on typing, you can use the *generateconfig.php* script. Provide your own AS number as an argument and the script will generate the configuration information from PeeringDB. For instance:

```
php generateconfig.php 35627
```

And to create the config file:

```
php generateconfig.php 35627 >config.txt
```

A previous config.txt file is overwritten. Find the generateconfig.php script as well as the routerconfig.php script here:

Download the generateconfig.php script
Download the routerconfig.php script

Generating router the actual configuration snippets

With the configuration in place, it's time to run the routerconfig.php script. This script takes as its input a router name, the AS number of a peer and optionally an MD5 password:

```
php routerconfig.php
Usage: php routerconfig.php <router> <ASN> [MD5 password]
```

If we now run the script with the configuration file above in place and provide the AS number of a peer, this is the result:

```
php routerconfig.php gnix-rtr 39704 5ecret

conf t
!
! configuration for gnix-rtr on 2017-01-29 for peer AS39704
!
router bgp 65065
!
 neighbor 193.111.172.79 remote-as 39704
 neighbor 193.111.172.79 password 5ecret
 neighbor 193.111.172.79 peer-group gnix-peers-ipv4
!
 neighbor 2001:7f8:31:0:3:3:9704:1 remote-as 39704
 neighbor 2001:7f8:31:0:3:3:9704:1 password 5ecret
 address-family ipv6 unicast
 neighbor 2001:7f8:31:0:3:3:9704:1 peer-group gnix-peers-ipv6
 exit-address-family
!
end
```

The script finds all the internet exchanges that we and our peer have in common, and then generates peering configurations for the ones where the indicated router is present. (So it's necessary to run the script for each router separately.)

The script outputs configuration lines that configure the desired peerings based on the IP addresses listed in PeeringDB, which you can copy and paste into the router configuration. It's a good idea to double check the output, sometimes there's outdated or incorrect IP addresses in PeeringDB.

The peer group reference is address family specific, hence we need to specify ***address-family ipv6 unicast*** first before we can set the IPv6 peer group.

It shouldn't be too hard to modify the script to generate configurations for different types of routers. Have a look at the function ***genconfig_cisco*** that starts at line 127. This function is called from the function ***genconfig*** that starts at line 110, where you can add extra lines to refer to additional router types. Note that ***genconfig_cisco*** is called for "router types" ***cisco, quagga*** and ***brocade***. For our purposes, Cisco and Quagga configurations are identical and Brocade only has a slightly different ***router bgp command*** syntax.

So have a look at these scripts and start configuring internet exchange peerings faster than ever.

NOCTION
NETWORK INTELLIGENCE

This ebook was brought to you by Noction.

Noction Intelligent Routing Platform enables enterprises and service providers to maximize end-to-end network performance and safely reduce infrastructure costs. The platform evaluates critical network performance metrics in real-time and responds quickly by automatically rerouting traffic through a better path to avoid outages and congestion.

Request a free trial today and see how IRP can boost your network performance.

Start a Free Trial