Netconf for Peering Automation

NANOG 64 – San Francisco

Tom Paseka

Old Ways

Old Ways

- ***Manual input***
- Very time consuming / manpower heavy
- Prone to human error:
  - Typo
  - Invalid and inconsistent input
  - Route leaks

Old Ways

- ***Manual input – with templates***
- Still prone to human error
- Lacks validation
- Copy and paste error still prone to all the errors from manual input

Old Ways

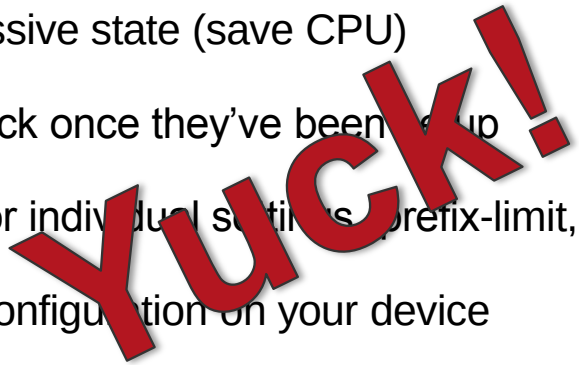
- ***Expect***
- Inelegant solution, though tried and tested
- Screen scraping, slow
- Security can be an issue
  - (Where are you keeping your password?)
- Scripting anything more complicated becomes very time consuming
  - expect router#

Old Ways

- ***Preconfiguration***
- Pre-configure every peer on an internet exchange
  - Set up peers in passive state (save CPU)
  - But you have to track once they've been setup
  - Doesn't help you for individual settings (prefix-limit, md5)
- LOADS of irrelevant configuration on your device
- Quality of data is an issue (peeringdb)
- Without a better way to input, still prone to human error.

Old Ways

- ***Preconfiguration***
- Pre-configure every peer on an internet exchange
  - Set up peers in passive state (save CPU)
  - But you have to track once they've been set up
  - Doesn't help you for individual settings (prefix-limit, md5)
- LOADS of irrelevant configuration on your device
- Quality of data is an issue (peeringdb)
- Without a better way to input, still prone to human error.



New Recipe

New Recipe

- NetConf (RFC 4741, RFC 6241, et. al)
- Programming language of choice
- Jump/Bastion Host
- Many different ways to cook it all up

Intro to NetConf

Intro to NetConf

- SNMP was thought to be used for configuration
  - It failed and was never adopted
- XML configuration base
- Transactional changes (backup, restore, etc)
- Configuration validation

Intro to NetConf

- Request to return the “running-configuration”

- <rpc message-id=“1” xmlns="urn:ietf:params:xml:ns:netconf:base:1.0”>
  <**get-config**>
  <source>
  <**running/**>
  </source>
  </get-config>
  </rpc>

Intro to NetConf

- Juniper includes a NetConf handler and examples
  - Its on GitHub!
    https://github.com/Juniper/netconf-perl
    https://github.com/Juniper/netconf-php
    https://github.com/Juniper/netconf-java
- A lot of examples are available there

Intro to NetConf

- $ ./arp.pl –h <router.hostname> –l <username> -p <password>

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos=http://xml.juniper.net/junos/12.3R6/junos>
<arp-table-information xmlns="http://xml.juniper.net/junos/12.3R6/junos-arp" style="no-resolve”>
        <arp-table-entry>
                <mac-address>
                        64:0e:94:28:02:c0
                </mac-address>
                <ip-address>
                        10.10.10.50
                </ip-address>
                <interface-name>
                        ae0.900
                </interface-name>
                <arp-table-entry-flags>
                        <none/>
                </arp-table-entry-flags>
        </arp-table-information>
</rpc-reply>
```

Router CLI Output

```
tom@router> show arp no-resolve
MAC Address          Address       Interface      Flags
64:0e:94:28:02:c0    10.10.10.50   ae0.900        none

tom@router>
```

Intro to NetConf

- This script sends a netconf request, asking for the ARP table on the router
- my $res = $jnx->get_arp_table_information(no_resolve => 1);
- In the examples from Juniper, you can change the request, this one is “get_arp_table_information”.
- “get_route_information” in the Juniper Libraries will show you the routing table

Intro to NetConf

- But XML is ugly.
- Your favorite scripting language saves the day!
- A very basic script can convert from ugly XML, to pretty format
- Going back to the ARP script…

Intro to NetConf

```
$ cat arp-parse.php
<?php

$dom = simplexml_load_file('php://stdin');

echo "MAC Address \t \t IP Address \t Interface \n";

foreach($dom->{'arp-table-information'}->{'arp-table-entry'} as $record){
        $mac = str_replace("\n", "", $record->{'mac-address'});
        $ip = str_replace("\n", "", $record->{'ip-address'});
        $interface = str_replace("\n", "", $record->{'interface-name'});
        echo "$mac \t $ip \t $interface \n";
}


?>
```

Intro to NetConf

```
$ ./arp.pl –h <router.hostname> -l <username> –p  <password>| php arp-parse.php
MAC Address             IP Address      Interface
64:0e:94:28:02:c0       10.10.10.50     ae0.900
```

Getting XML Configuration

Getting XML configuration

- Juniper has a useful command to see what the XML configuration looks like for beginner

- “show configuration | display XML”

Getting XML configuration

```
{master}
tom@re0.router> show configuration protocols bgp group 4-PUBLIC-PEERS neighbor 206.223.116.102 | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/12.3R6/junos">
    <configuration junos:commit-seconds="1425340126" junos:commit-localtime="2015-03-02 23:48:46 UTC" junos:commit-user="user">
            <protocols>
                <bgp>
                    <group>
                        <name>4-PUBLIC-PEERS</name>
                        <neighbor>
                            <name>206.223.116.102</name>
                            <description>Akamai [WW CDN]</description>
                            <family>
                                <inet>
                                    <unicast>
                                        <prefix-limit>
                                            <maximum>250</maximum>
                                        </prefix-limit>
                                    </unicast>
                                </inet>
                            </family>
                            <peer-as>20940</peer-as>
                        </neighbor>
                    </group>
                </bgp>
            </protocols>
    </configuration>
    <cli>
        <banner>{master}</banner>
    </cli>
</rpc-reply>

{master}
tom@re0.router>
```

Getting XML configuration

```
<configuration>
        <protocols>
           <bgp>
              <group>
                 <name>4-PUBLIC-PEERS</name>
                 <neighbor>
                    <name>$ipaddr</name>
                    <description>$descr</description>
                    <family>
                       <inet>
                          <unicast>
                             <prefix-limit>
                                <maximum>$pfxcnt</maximum>
                             </prefix-limit>
                          </unicast>
                       </inet>
                    </family>
                    <peer-as>$ASN</peer-as>
                 </neighbor>
              </group>
           </bgp>
        </protocols>
  </configuration>
```

Getting XML configuration

```
<configuration>
        <protocols>
           <bgp>
              <group>
                 <name>4-PUBLIC-PEERS</name>
                 <neighbor>
                    <name>$ipaddr</name>
                    <description>$descr</description>
                    <family>
                       <inet>
                          <unicast>
                             <prefix-limit>
                                <maximum>$pfxcnt</maximum>
                             </prefix-limit>
                          </unicast>
                       </inet>
                    </family>
                    <peer-as>$asn</peer-as>
                 </neighbor>
              </group>
           </bgp>
        </protocols>
  </configuration>
```

Getting XML configuration

- Template is very simple
- Build out your group (be it Juniper “group”, Cisco “peer-group”, whatever)
- Peer config drops inside that group.
- There are really only 4 values you need to insert into the template

Pushing Peering Configuration

Pushing Peering Configuration

- Beginners way – write config to a file then push it.
- Using template above, save it in a text file, Juniper has a handler to push inside :

  ./edit_configuration.pl -l <username> –p <password> <config file> <router>

- Once this is pushed, this handler validates configuration and applies to the running config on the router

Pushing Peering Configuration

- Intermediate mode – write a handler around your own push

- $ php peer_push.php -h router –d “Testing Session" –ip 1.1.1.1 –as 1234 –max 250
  REQUEST succeeded !! - 0
  $

- Extra steps
  - Validatie the peer IP/ASN details from external sources
  - Don’t troll me for using PHP ☺

Pushing Peering Configuration

- Advanced Mode:

- Pulling peer configuration from PeeringDB and other sources for configuration
- A front end for configuration, validation status
- Anything you want to build!

Pushing Peering Configuration - Validate

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="http://xml.juniper.net/junos/12.3R6/junos">
        <bgp-information xmlns="http://xml.juniper.net/junos/12.3R6/junos-routing">
                <bgp-peer style="detail">
                <peer-address>1.1.1.1+65403</peer-address>
                <peer-as>1234</peer-as>
                <local-address>2.2.2.2+179</local-address>
                <local-as>13335</local-as>
                <description>Testing Session</description>
                <peer-type>External</peer-type>
                <peer-state>Established</peer-state>
                <peer-flags>Sync</peer-flags>
                <last-state>OpenConfirm</last-state>
                <last-event>RecvKeepAlive</last-event>
                <last-error>Cease</last-error>
        <bgp-option-information xmlns="http://xml.juniper.net/junos/12.3R6/junos-routing">
                <export-policy>
                        REJECT-ALL
                </export-policy>
                <import-policy>
                        REJECT-ALL
                </import-policy>
                <bgp-options>Multihop Preference LocalAddress HoldTime Ttl LogUpDown AddressFamily PeerAS PrefixLimit LocalAS Refresh</bgp-options>
                <bgp-options2></bgp-options2>
                <bgp-options-extended></bgp-options-extended>
                <address-families>inet-unicast inet-flow</address-families>
.........
```

Pushing Peering Configuration - Validate

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:junos="http://xml.juniper.net/junos/12.3R6/junos">
        <bgp-information xmlns="http://xml.juniper.net/junos/12.3R6/junos-routing">
                <bgp-peer style="detail">
                <peer-address>1.1.1.1+65403</peer-address>
                <peer-as>1234</peer-as>
                <local-address>2.2.2.2+179</local-address>
                <local-as>13335</local-as>
                <description>Testing Session</description>
                <peer-type>External</peer-type>
                <peer-state>Established</peer-state>
                <peer-flags>Sync</peer-flags>
                <last-state>OpenConfirm</last-state>
                <last-event>RecvKeepAlive</last-event>
                <last-error>Cease</last-error>
        <bgp-option-information xmlns="http://xml.juniper.net/junos/12.3R6/junos-routing">
                <export-policy>
                        REJECT-ALL
                </export-policy>
                <import-policy>
                        REJECT-ALL
                </import-policy>
                <bgp-options>Multihop Preference LocalAddress HoldTime Ttl LogUpDown AddressFamily PeerAS PrefixLimit LocalAS Refresh</bgp-options>
                <bgp-options2></bgp-options2>
                <bgp-options-extended></bgp-options-extended>
                <address-families>inet-unicast inet-flow</address-families>
.........
```

Summary

Summary

- It’s easy to build the base for automation
  - Even for non-programmers like myself
- Make the system well, validate and its easy to build on top of it.
- Using the script, even at the intermediate mode can save minutes per peering session turn up, saving hours and days for large peering deployments

Questions?