

Use Cases: Application Hosting

This chapter describes use cases for running applications on IOS XR.

- Running a Telemetry Receiver in a Linux Container (LXC) , on page 1

Running a Telemetry Receiver in a Linux Container (LXC)

For telemetry to work on Cisco IOS XR, it must use GPB (Google Protocol Buffer) over UDP, instead of TCP.

The procedure consists of the following steps:

1. Create a telemetry policy file.
2. Generate and compile a .proto file.
3. Configure the GPB encoder.
4. Launch a third-party container (LXC).
5. Configure the telemetry receiver.

Creating a Telemetry Policy File

A telemetry policy file is used to specify the kind of data to be generated and pushed to the telemetry receiver. The following steps describe how you can create the policy file for telemetry:

1. Determine the schema paths to stream data.

```
RP/0/RP0/CPU0:ios# schema-describe show interface
Wed Aug 26 02:24:40.556 PDT
RootOper.InfraStatistics.Interface(*).Latest.GenericCounters
```

2. Create a policy file that contains these paths:

```
{
"Name": "Test",
"Metadata": {
"Version": 25,
"Description": "This is a sample policy",
"Comment": "This is the first draft",
"Identifier": "<data that may be sent by the encoder to the mgmt stn"
},
"CollectionGroups": {
```

```
"FirstGroup": {
"Period": 30,
"Paths": [
"RootOper.InfraStatistics.Interface(*).Latest.GenericCounters"
   ]
  }
 }
}
```

3. Enter the XR Linux bash shell, and copy the policy file to IOS XR by using Secure Copy Protocol (SCP).

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash


[XR-vm_node0_RP0_CPU0:~]$ scp Test.policy cisco@10.0.0.1:/telemetry/policies
cisco@10.0.0.1's password:
Test.policy
100% 779 0.8KB/s 00:00
Connection to 10.0.0.1 closed by remote host.
```

   Where 10.0.0.1 is the IP address of the device on which you are copying the policy file.

4. Navigate to the IOS XR prompt and verify if the policy file has been successfully installed.

```
RP/0/RP0/CPU0:ios# show telemetry policies brief
Wed Aug 26 02:24:40.556 PDT
Name |Active?| Version | Description
---------------------------|-------|---------|------------------------------
Test N 1 This is a sample policy
```

Generating and Compiling a .proto File

The path in a policy file that you created needs a `.proto` file associated with it. The .proto file describes the GPB message format used to stream data. The following steps describe how you can generate and compile a .proto file for a telemetry receiver:

The `.proto` file is complied into a `.map` file. The compilation is done on a server.

1. Generate a .proto file.

```
telemetry generate gpb-encoding path
"RootOper.InfraStatistics.Interface(*).Latest.GenericCounters" file
disk0:generic_counters.proto
```

   The .proto file is generated by an on-box tool. The tool ignores naming parameters, and are hence optional.



Note The tool ignores text within quotes; therefore, the path should not contain quotes.

2. Compile the .proto file off the box.
   a. Cisco provides a telemetry compiler on Dev Hub. You can copy the directory to your Linux box, and run it, as shown here:

```
telemetry_protoc -f generic_counters.proto -o generic_counters.map
```

b. Access the copy of the .proto file from Dev Hub, and run the standard compiler on your Linux box, as shown here:

```
protoc python_out . -I=/
sw/packages/protoc/current/google/include/:.
generic_counters.proto ipv4_counters.proto
```

3. Copy the map file to IOS XR at `/telemetry/gpb/maps`.

Configuring the GPB Encoder

Configure the GPB encoder to activate the telemetry policy and stream data as outlined in the following steps:

1. Configure a loopback interface address for mapping the telemetry receiver to IOS XR, as shown here:

```
RP/0/RP0/CPU0:ios(config)# interface Loopback2
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 2.2.2.2/32
RP/0/RP0/CPU0:ios(config-if)# no shut
RP/0/RP0/CPU0:ios(config-if)# commit
Fri Oct 30 07:51:14.785 UTC
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# exit
RP/0/RP0/CPU0:ios# show ipv4 interface brief
Fri Oct 30 07:51:48.996 UTC

Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
Loopback2                      2.2.2.2         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

2. Configure the encoder to stream the policy to the loopback interface of IOS XR that was just configured.

```
telemetry
          encoder gpb
            policy group alpha
              policy demo
              destination ipv4 2.2.2.2 port 5555
             !
      !
   !
```

Launching a Third-Party Container (LXC)

This section describes how you can launch a third-party container (LXC) on IOS XR.

1. Log into IOS XR.

```
RP/0/RP0/CPU0:ios# run
[xr-vm_node0_RP0_CPU0:~]$
```

2. Launch the third-party container.

```
[xr-vm_node0_RP0_CPU0:~]$ virsh -c lxc+tcp://10.11.12.15:16509/ -e ^Q console demo1
```

3. Log into the container when prompted.

```
Connected to domain demo
Escape character is ^Q
```

```
Kernel 3.14.23-WR7.0.0.2_standard on an x86_64
host login: Password:
```

You have successfully launched a third-party container.

Configuring the Telemetry Receiver

A telemetry receiver listens for streamed data on the specified interface IP address and port number, and it prints the header of the received packets. If .proto files are provided, they are compiled using the protoc compiler and the message contents are also printed. By default, only the first row of each table is printed, though the `print-all` option can be used to print the complete output.

To run a telemetry receiver within the container you launched, use the following steps:

1. Download all the receiver files to the third-party container. The receiver files are available on IOS XR at https://github.com/cisco/bigmuddy-network-telemetry-collector.
2. Run the receiver to stream and print data.

   ```
   python gpb_receiver.py ipaddress 2.2.2.2 port 5555 proto
   generic_counters.proto ipv4_counters.proto
   ```

   You can see data on the telemetry receiver, as shown here:

   ```
   Waiting for message
   Got message of length:1036bytes from address:('10.1.1.1', 5555)
   Encoding:2271560481
   Policy Name:demo
   Version:25
   Identifier:<data that may be sent by the encoder to the mgmt stn>
   Start Time:Wed Jan 21 09:54:33 1970
   End Time:Wed Aug 26 09:28:37 2015
   # Tables:1
   Schema
   Path:RootOper.InfraStatistics.Interface.Latest.GenericCounters
   # Rows:6
   Row 0:
   applique:0
   availability_flag:0
   broadcast_packets_received:0
   broadcast_packets_sent:0
   bytes_received:0
   bytes_sent:0
   carrier_transitions:0
   crc_errors:0
   framing_errors_received:0
   giant_packets_received:0
   input_aborts:0
   input_drops:0
   input_errors:0
   input_ignored_packets:0
   input_overruns:0
   input_queue_drops:0
   interface_name:Null0
   last_data_time:1440606516
   last_discontinuity_time:1440498130
   multicast_packets_received:0
   multicast_packets_sent:0
   output_buffer_failures:0
   output_buffers_swapped_out:0
   output_drops:0
   output_errors:0
   output_queue_drops:0
   ```

```
output_underruns:0
packets_received:0
packets_sent:0
parity_packets_received:0
resets:0
runt_packets_received:0
seconds_since_last_clear_counters:0
seconds_since_packet_received:4294967295
seconds_since_packet_sent:4294967295
throttled_packets_received:0
unknown_protocol_packets_received:0
Waiting for message
Got message of length:510bytes from address:('2.2.2.2', 5555)
Encoding:2271560481
Policy Name:demo
Version:25
Identifier:<data that may be sent by the encoder to the mgmt stn>
Start Time:Wed Jan 21 09:54:33 1970
End Time:Wed Aug 26 09:28:38 2015
# Tables:1
Schema Path:RootOper.InfraStatistics.Interface.Latest.Protocol
# Rows:5
Row 0:
bytes_received:0
bytes_sent:0
input_data_rate:0
input_packet_rate:0
interface_name:Loopback2
last_data_time:1440606517
output_data_rate:0
output_packet_rate:0
packets_received:0
packets_sent:0
protocol:24
protocol_name:IPV4_UNICAST
```

The telemetry receiver runs successfully within the third-party container (LXC).

Use Cases on Vagrant: Container Application Hosting

This section describes how you can use vagrant to run use cases for container application hosting.

Pre-requisites for Using Vagrant

Before you can start using vagrant, ensure that you have fulfilled the following requirements on your host device.

- Latest version of Vagrant for your operating system. We recommend Version 1.8.6.
- Latest version of a virtual box for your operating system. We recommend Version 5.1+.
- Minimum of 5 GB of RAM with two cores.
- (Optional) If you are using the Windows Operating System, we recommend that you download the Git bash utility for running the commands.

OSPF Path Failover by Running iPerf with Netconf on Vagrant

This section describes a use case for solving a path remediation problem by using iPerf and Netconf applications on vagrant.

Topology

The topology used for OSPF path remediation is illustrated in the following figure.

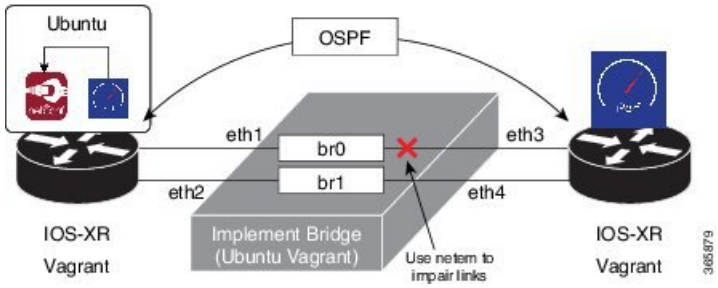
The router on the left is rtr1 and is the source of traffic. We run the pathchecker application inside an LXC on this router. Pathchecker uses an iPerf client to determine the health of the path.

The router on the right is rtr2 and is the destination for traffic. We run the pathchecker application inside an LXC on this router. Pathchecker uses an iPerf server that talks to the iPerf client on rtr1.

`devbox` serves two purposes in this topology:

- To create an LXC tar ball with pathchecker before being deployed to the routers.
- To bridge the two networks between the two routers over the parallel paths.

Figure 1: OSPF Path Failover with iPerf and Netconf on Vagrant



This example uses the following process for OSPF path failover:

1. Configure and establish OSPF neighbor relationship between two routers over two parallel paths.
2. Increase the cost of one path so that the other path is the preferred active path.
3. Use the pathchecker python application to monitor the OSPF active path by determining the bandwidth, jitter, packet loss and other parameters. Pathchecker uses the iPerf application to measure health of the active traffic path.
4. Use pathchecker to simulate network degradation by changing the OSPF active path cost during a Netconf session.

Procedure

Use the following steps to use iPerf with Netconf for OSPF path failover.

1. Generate an API key and a CCO ID by using the steps described on Github.
2. Download the latest stable version of the IOS-XRv vagrant box.

```
$ curl <cco-id>:<API-KEY>

$ BOXURL --output ~/iosxrv-fullk9-x64.box
```

```
$ vagrant box add --name IOS-XRv ~/iosxrv-fullk9-x64.box
```

3. Verify if the vagrant box has been successfully installed.

```
 AKSHSHAR-M-K0DS:~ akshshar$ vagrant box list
IOS-XRv (virtualbox, 0)
```

4. Create a working directory.

```
 AKSHSHAR-M-K0DS:~ akshshar$ mkdir ~/iosxrv
 AKSHSHAR-M-K0DS:~ akshshar$cd ~/iosxrv
```

5. Initialize the vagrant file with the new vagrant box.

```
 AKSHSHAR-M-K0DS:~ akshshar$ vagrant init IOS-XRv
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

6. Clone the repository containing the pathchecker code.

```
AKSHSHAR-M-K0DS:~ akshshar$ git clone https://github.com/ios-xr/pathchecker.git
Cloning into 'pathchecker'...
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 46 (delta 8), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (46/46), done.
Checking connectivity... done.
```

7. Navigate to the `pathchecker/vagrant` directory and launch `devbox`.

```
AKSHSHAR-M-K0DS:~ akshshar$ cd pathchecker/
AKSHSHAR-M-K0DS:pathchecker akshshar$ cd vagrant/
AKSHSHAR-M-K0DS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant

AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant up devbox
Bringing machine 'devbox' up with 'virtualbox' provider...
==> devbox: Importing base box 'ubuntu/trusty64'...

---------------------------- snip output ---------------------------------

==> devbox: Running provisioner: file...
AKSHSHAR-M-K0DS:vagrant akshshar$
AKSHSHAR-M-K0DS:vagrant akshshar$
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant status
Current machine states:


rtr1                      not created (virtualbox)
devbox                    running (virtualbox)
rtr2                      not created (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

8. Launch an LXC within `devbox`.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh devbox

vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-create -t ubuntu --name pathchecker
Checking cache download in /var/cache/lxc/trusty/rootfs-amd64 ...
Installing packages in template: ssh,vim,language-pack-en
Downloading ubuntu trusty minimal ...
I: Retrieving Release
I: Retrieving Release.gpg
I: Checking Release signature
...
vagrant@vagrant-ubuntu-trusty-64:~$ sudo lxc-start --name pathchecker
<4>init: hostname main process (3) terminated with status 1
<4>init: plymouth-upstart-bridge main process (5) terminated with status 1
<4>init: plymouth-upstart-bridge main process ended, respawning


Ubuntu 14.04.4 LTS nc_iperf console


pathchecker login: ubuntu
Password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

...
```

9. Install all the required iPerf and Netconf application dependencies within the LXC.

```
ubuntu@pathchecker:~$ sudo apt-get -y install python-pip python-lxml
 python-dev libffi-dev libssl-dev iperf git

ubuntu@pathchecker:~$ sudo pip install ncclient jinja2 cryptography==1.2.1
```

10. Retrieve the iPerf and Netconf application code from Github.

```
ubuntu@pathchecker:~$ git clone https://github.com/ios-xr/pathchecker.git
Cloning into 'pathchecker'...
remote: Counting objects: 46, done.
remote: Compressing objects: 100% (28/28), done.
remote: Total 46 (delta 8), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (46/46), done.
Checking connectivity... done.
ubuntu@pathchecker:~$
```

11. Change the SSH port inside the LXC.

When a container is deployed on XR, it shares the network namespace of XR. Since XR uses ports 22 and 57722 for internal processes, we change the port number to 58822 in this example.

```
ubuntu@pathchecker:~$ sudo sed -i s/Port\ 22/Port\ 58822/ /etc/ssh/sshd_config

ubuntu@pathchecker:~$ cat /etc/ssh/sshd_config | grep Port
Port 58822
```

12. Create the LXC tar ball.

a. Shut down the LXC.

```
ubuntu@pathchecker:~$ sudo shutdown -h now
ubuntu@pathchecker:~$
Broadcast message from ubuntu@pathchecker
 (/dev/lxc/console) at 10:24 ...

The system is going down for halt NOW!
```

b. Assume the root user role.

```
vagrant@vagrant-ubuntu-trusty-64:~$ sudo -s
root@vagrant-ubuntu-trusty-64:~# whoami
root
```

c. Navigate to the `/var/lib/lxc/pathchecker/rootfs/` directory and package the `rootfs` into a tar ball.

```
root@vagrant-ubuntu-trusty-64:~# cd /var/lib/lxc/pathchecker/rootfs/
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/pathchecker/rootfs/# tar -czvf
/vagrant/pathchecker_rootfs.tar.gz *
tar: dev/log: socket ignored
root@vagrant-ubuntu-trusty-64:/var/lib/lxc/pathchecker/rootfs/# exit
vagrant@vagrant-ubuntu-trusty-64:~$ exit
logout
Connection to 127.0.0.1 closed.

AKSHSHAR-M-K0DS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant
AKSHSHAR-M-K0DS:vagrant akshshar$ ls -l pathchecker_rootfs.tar.gz
-rw-r--r--  1 akshshar  staff  301262995 Jul 18 07:57 pathchecker_rootfs.tar.gz
AKSHSHAR-M-K0DS:vagrant akshshar$
```

13. Launch the two router topology.

a. Navigate to the `pathchecker/vagrant` directory and launch the vagrant instance.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ pwd
/Users/akshshar/pathchecker/vagrant

AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant up
Bringing machine 'rtr1' up with 'virtualbox' provider...
Bringing machine 'devbox' up with 'virtualbox' provider...
Bringing machine 'rtr2' up with 'virtualbox' provider...
```

b. Verify if the topology has been launched.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant status
Current machine states:


rtr1                      running (virtualbox)
devbox                    running (virtualbox)
rtr2                      running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

14. Verify if OSPF is running on rtr1 and check the path state.

You can also see the cost of the OSPF path.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant port rtr1
The forwarded ports for the machine are listed below. Please note that
these values may differ from values configured in the Vagrantfile if the
provider supports automatic port collision detection and resolution.

 22 (guest) => 2223 (host)
 57722 (guest) => 2200 (host)
 58822 (guest) => 58822 (host)
AKSHSHAR-M-K0DS:vagrant akshshar$ ssh -p 2223 vagrant@localhost
The authenticity of host '[localhost]:2223 ([127.0.0.1]:2223)' can't be established.
RSA key fingerprint is b1:c1:5e:a5:7e:e7:c0:4f:32:ef:85:f9:3d:27:36:0f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2223' (RSA) to the list of known hosts.
vagrant@localhost's password:



RP/0/RP0/CPU0:rtr1# show  running-config  router ospf
Mon Jul 18 15:25:53.875 UTC
router ospf apphost
 area 0
  interface Loopback0
  !
  interface GigabitEthernet0/0/0/0
  !
  interface GigabitEthernet0/0/0/1
    cost 20
  !
 !
!

RP/0/RP0/CPU0:rtr1# show route 2.2.2.2
Mon Jul 18 15:26:03.576 UTC

Routing entry for 2.2.2.2/32
  Known via "ospf apphost", distance 110, metric 2, type intra area
  Installed Jul 18 15:18:28.218 for 00:07:35
  Routing Descriptor Blocks
   10.1.1.20, from 2.2.2.2, via GigabitEthernet0/0/0/0
      Route metric is 2
  No advertising protos.
RP/0/RP0/CPU0:rtr1#
```

15. Start the iPerf server on `rtr2` and configure it for receiving packets from `rtr1`.

Note iPerf was launched as a native application on `rtr2` while launching the vagrant instance.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh rtr2
Last login: Mon Jul 18 15:57:05 2016 from 10.0.2.2
xr-vm_node0_RP0_CPU0:~$
xr-vm_node0_RP0_CPU0:~$ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 64.0 MByte (default)
```

16. Launch the pathchecker application within the LXC on `rtr1`.

a. Log in to the LXC on `rtr1`.

Password for user `ubuntu` is **ubuntu**.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ ssh -p 58822 ubuntu@localhost
The authenticity of host '[localhost]:58822 ([127.0.0.1]:58822)' can't be
established.
RSA key fingerprint is 19:54:83:a9:7a:9f:0a:18:62:d1:f3:91:87:3c:e9:0b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:58822' (RSA) to the list of known hosts.
ubuntu@localhost's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.14.23-WR7.0.0.2_standard x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Mon Jul 18 15:19:45 2016 from 10.0.2.2
ubuntu@pathchecker:~$
```

b. Navigate to the pathchecker repository within the LXC, and check the contents of the pathchecker script.

```
ubuntu@pathchecker:~$ cd pathchecker/
ubuntu@pathchecker:~/pathchecker$ cat pc_run.sh
#!/bin/bash

./pathchecker.py --host 6.6.6.6 -u vagrant -p vagrant --port 830 -c 10 -o
apphost -a 0 -i GigabitEthernet0/0/0/0 -s 2.2.2.2  -j 4 -l 5 -f -t 10
```

`-I` represents the threshold for packet loss and has been set to 5% for this run. `-j` represents the jitter threshold that has a value of 4.

c. Start the pathchecker application by running the script.

```
ubuntu@pathchecker:~/pathchecker$ ./pc_run.sh
Error while opening state file, let's assume low cost state
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run.....
20160718162513,1.1.1.1,62786,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718162513,1.1.1.1,62786,2.2.2.2,5001,6,0.0-10.0,1312710,1048474
20160718162513,2.2.2.2,5001,1.1.1.1,62786,6,0.0-10.0,1312710,1048679,2.453,0,892,0.000,1

bw is
1025.5546875
jitter is
2.453
pkt_loss is
0.000
verdict is
False
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run.....
```

The pathchecker application is running on the path from `GigabitEthernet0/0/0/0` interface.

17. Open a parallel Git bash window and simulate impairment on the active path.

a. Access `devbox` through SSH.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ cd pathchecker/vagrant

AKSHSHAR-M-K0DS:vagrant akshshar$ vagrant ssh devbox
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-87-generic x86_64)

 ...
```

b. View the impairment script and run it on `devbox`.

```
vagrant@vagrant-ubuntu-trusty-64:~$ ls
impair_backup.sh  impair_reference.sh  stop_impair.sh

vagrant@vagrant-ubuntu-trusty-64:~$ cat impair_reference.sh
#!/bin/bash
echo "Stopping all current impairments"
sudo tc qdisc del dev eth3 root &> /dev/null
sudo tc qdisc del dev eth4 root &> /dev/null
echo "Starting packet loss on reference link"
sudo tc qdisc add dev eth3 root netem loss 7%

vagrant@vagrant-ubuntu-trusty-64:~$ ./impair_reference.sh
Stopping all current impairments
Starting packet loss on reference link
```

The script creates a packet loss of 7% on the reference link.

18. Open the first Git bash window to view the pathchecker application running on `rtr1`.

```
Currently, on reference link GigabitEthernet0/0/0/0
Starting an iperf run.....
20160718164745,1.1.1.1,60318,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164745,1.1.1.1,60318,2.2.2.2,5001,6,0.0-10.0,1312710,1048516
20160718164745,2.2.2.2,5001,1.1.1.1,60318,6,0.0-573.0,1312710,18328,5.215,0,892,0.000,1

bw is
1025.5546875
jitter is
5.215
pkt_loss is
0.000
verdict is
True
Woah! iperf run reported discrepancy, increase cost of reference link !
Increasing cost of the reference link GigabitEthernet0/0/0/0
Currently, on backup link
Starting an iperf run.....
20160718164755,1.1.1.1,61649,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164755,1.1.1.1,61649,2.2.2.2,5001,6,0.0-10.0,1312710,1048577
20160718164755,2.2.2.2,5001,1.1.1.1,61649,6,0.0-583.3,1312710,18002,1.627,0,893,0.000,0

bw is
1025.5546875
jitter is
1.627
pkt_loss is
0.000
verdict is
False
Currently, on backup link
Starting an iperf run.....
20160718164805,1.1.1.1,59343,2.2.2.2,5001,6,0.0-10.0,1311240,1048992
20160718164805,1.1.1.1,59343,2.2.2.2,5001,6,0.0-10.0,1312710,1048520
20160718164805,2.2.2.2,5001,1.1.1.1,59343,6,0.0-593.4,1312710,17697,2.038,0,893,0.000,0
```

Pathchecker has initiated a failover from primary to secondary link.

19. Verify if the failover was successful on `rtr1`.

```
AKSHSHAR-M-K0DS:vagrant akshshar$ ssh -p 2223 vagrant@localhost
vagrant@localhost's password:



RP/0/RP0/CPU0:rtr1# show  running-config  router ospf
Mon Jul 18 17:50:47.851 UTC
router ospf apphost
 area 0
  interface Loopback0
  !
  interface GigabitEthernet0/0/0/0
   cost 30
  !
  interface GigabitEthernet0/0/0/1
   cost 20
  !
 !
!
```

The path cost from the GigabitEthernet0/0/0/0 interface is greater than that from the GigabitEthernet0/0/0/1 interface. Hence, failover takes place to the GigabitEthernet0/0/0/1 interface for traffic from `rt1` to `rtr2`.

20. Verify the OSPF path failover on **`rtr1`**.

The Loopback 0 interface IP address of `rtr1` in this example is 2.2.2.2

```
RP/0/RP0/CPU0:rtr1# show route 2.2.2.2
Mon Jul 18 18:01:49.297 UTC

Routing entry for 2.2.2.2/32
  Known via "ospf apphost", distance 110, metric 21, type intra area
  Installed Jul 18 16:47:45.705 for 01:14:03
  Routing Descriptor Blocks
  11.1.1.20, from 2.2.2.2, via GigabitEthernet0/0/0/1
      Route metric is 21
  No advertising protos.
RP/0/RP0/CPU0:rtr1#
```

The next hop for `rtr1` is 11.1.1.20 through the backup reference link: GigabitEthernet0/0/0/1

You have successfully configured OSPF path failover by using iPerf and Netconf on vagrant.