LAB – TABULATE AND PRINT ROUTES PER INTERFACE

OBJECTIVE

In this lab, you will read IP route information from a device, then allow a user to enter IP address prefixes that will display route information if the route was learned via OSPF.

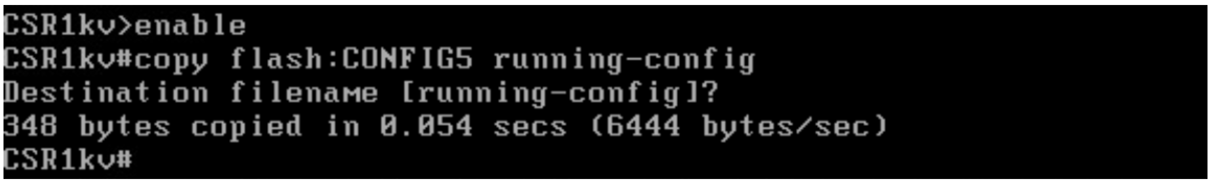
PART 1

Copy the required configuration for this lab from flash into the running-configuration of the CSR1000v and R2 routers.

STEP 1: COPY CONFIGURATION ON CSR1000V (R1)

In the CSR1000v router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG5** file to the running-configuration using the commands:

```
CSR1kv> enable
CSR1kv# copy flash:CONFIG5 running-config
```



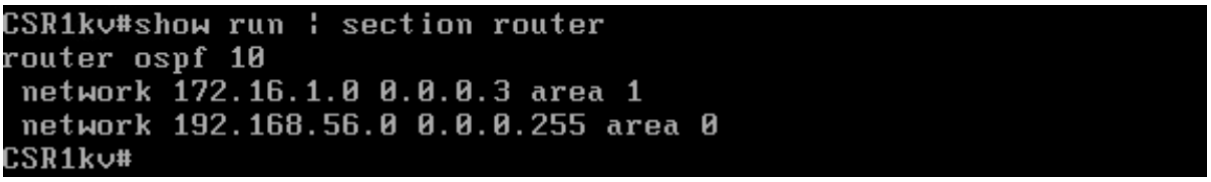
```
CSR1kv>enable
CSR1kv#copy flash:CONFIG5 running-config
Destination filename [running-config]?
348 bytes copied in 0.054 secs (6444 bytes/sec)
CSR1kv#
```

STEP 2: VERIFY CONFIGURATION ON CSR1000V (R1)

Verify the new running-configuration by using the command:

```
CSR1kv# show run | section router
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.

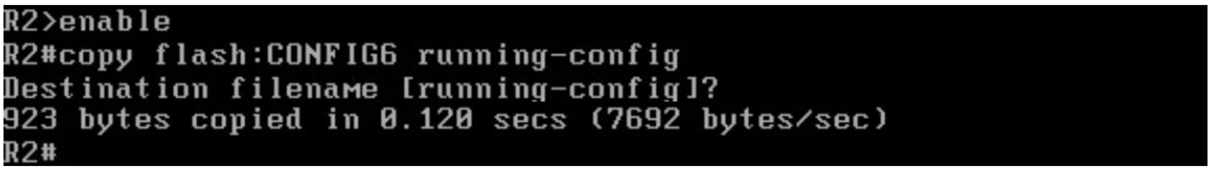


```
CSR1kv#show run ¦ section router
router ospf 10
 network 172.16.1.0 0.0.0.3 area 1
 network 192.168.56.0 0.0.0.255 area 0
CSR1kv#
```

STEP 3: COPY CONFIGURATION ON R2

In the R2 router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG6** file to the running-configuration using the commands:

```
R2> enable
R2# copy flash:CONFIG6 running-config
```



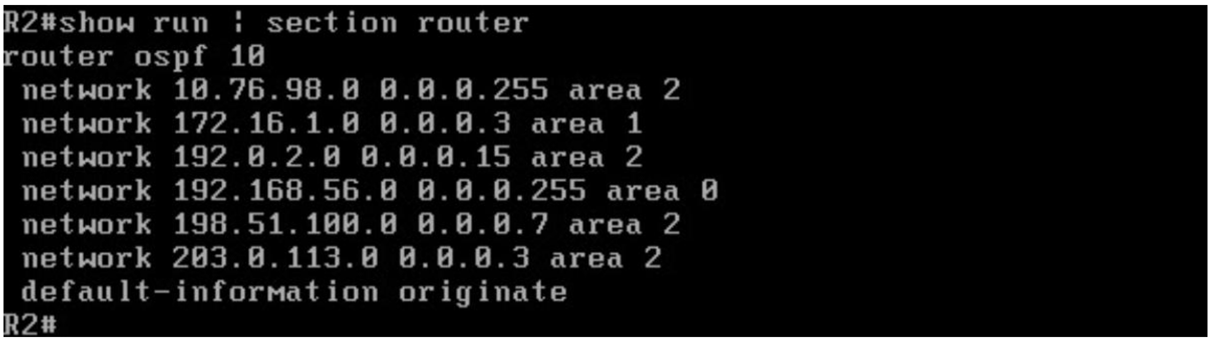
```
R2>enable
R2#copy flash:CONFIG6 running-config
Destination filename [running-config]?
923 bytes copied in 0.120 secs (7692 bytes/sec)
R2#
```

STEP 4: VERIFY CONFIGURATION ON R2

Verify the new running-configuration by using the command:

```
CSR1kv# show run | section router
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.



```
R2#show run | section router
router ospf 10
 network 10.76.98.0 0.0.0.255 area 2
 network 172.16.1.0 0.0.0.3 area 1
 network 192.0.2.0 0.0.0.15 area 2
 network 192.168.56.0 0.0.0.255 area 0
 network 198.51.100.0 0.0.0.7 area 2
 network 203.0.113.0 0.0.0.3 area 2
 default-information originate
R2#
```

PART 2

Open a terminal and switch to the lab directory

STEP 1: OPEN A TERMINAL WINDOW

Double-click the Terminal icon on the desktop to open the terminal window for use in this lab.

STEP 2: CHANGE DIRECTORY

Change to the directory **labs/prne/** in the user home directory, which holds the files for the course labs.

```
~$ cd labs/prne/
```

PART 3

Open **Visual Studio Code**, create a new file and save it with a filename of **tabulate-and-print-routes-per-interface.py**, ensuring to save the file in the **~/labs/prne/** directory.
This python application will connect to a router to read the IP route information from a device.

STEP 1: IMPORT MODULES/PACKAGES/LIBRARY

Import the pexpect library to allow connection to the router and the re module to allow the use of regular expressions.

```
# Import required modules/packages/library
import pexpect
import re
```

STEP 2: CREATE EMPTY LIST

Create an empty list called routes_list to store the route information.

```
# Create the list of routes
routes_list = []
```

STEP 3: CONNECT TO THE ROUTER

Connect to a router in the lab environment and run the **show ip route** command.

```
# Connect to the device
print('--- connecting telnet 192.168.56.101 with prne/cisco123!')

session = pexpect.spawn('telnet 192.168.56.101', encoding='utf-8',
                        timeout=20)
result = session.expect(['Username:', pexpect.TIMEOUT, pexpect.EOF])

# Check for failure
if result != 0:
    print('Timout or unexpected reply from device')
    exit()

# Login with username
session.sendline('prne')
result = session.expect('Password:')

# Send password
session.sendline('cisco123!')
result = session.expect('>')

# Must set terminal length to zero for long replies
print('--- setting terminal length to 0')
session.sendline('terminal length 0')
result = session.expect('>')

# Execute the 'show ip route' command to get routing info
print('--- executing: show ip route')
session.sendline('show ip route')
result = session.expect('>')

# Get output from ip route command
print('--- getting ip route command output')
show_ip_route_output = session.before

# Display routing information
print('')
print('IP route output')
print('---------------------------------------------------')
print(session.before)
print('---------------------------------------------------')
print('')
```

STEP 4: CREATE LIST OF DESTINATION PREFIXES

Using the output of the above **show ip route** command, create a list of destination IP address prefixes. For each destination, you will create a list of potential next hops, including next hop IP address, interface, and any other information you choose.

```
# Get routing information into list
routes_list = show_ip_route_output.splitlines()
```

STEP 5: ALLOW IP PREFIX ENTRY

Allow the user to input a destination IP address prefix.

```
# Loop forever, until user terminates program
while True:

    # Request user input the IP destination route prefix for searching
    try:
        ip_address = input('Enter IP destination address to find (Ctrl-C to exit): ')
    except KeyboardInterrupt:
        break
```

STEP 6: MATCH PREFIX

Go through the list of destination IP prefixes, searching for a match learned via OSPF. Print the route information if there is a match. If there is no match, display **--- Given route prefix not found ---**.

```
    # Set the pattern for matching OSPF routes
    route_pattern = re.compile(r'O.\D+(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})')

    # Loop through our devices looking for a match on IP address
    for route in routes_list:

        # Search for our route string,
        # and continue to next iteration if not found
        route_match = route_pattern.search(route)
        if not route_match:
            continue

        # Found our IP address, display route information
        if route_match.group(1) == ip_address:
            route_info = route.split(',')
            print('  ---- Route:     ', route_info[0][5:].strip())
            print('  ---- Time:      ', route_info[1].strip())
            print('  ---- Interface: ', route_info[2].strip())
            print('')
            break

    else:  # We get here if we exhausted the device list, IP not found
        print('--- Given route prefix not found ---')

# Display a blank line to make easier to read
print('')
print('Route search terminated.\n')
```

STEP 6: CLOSE

Close the session and file.

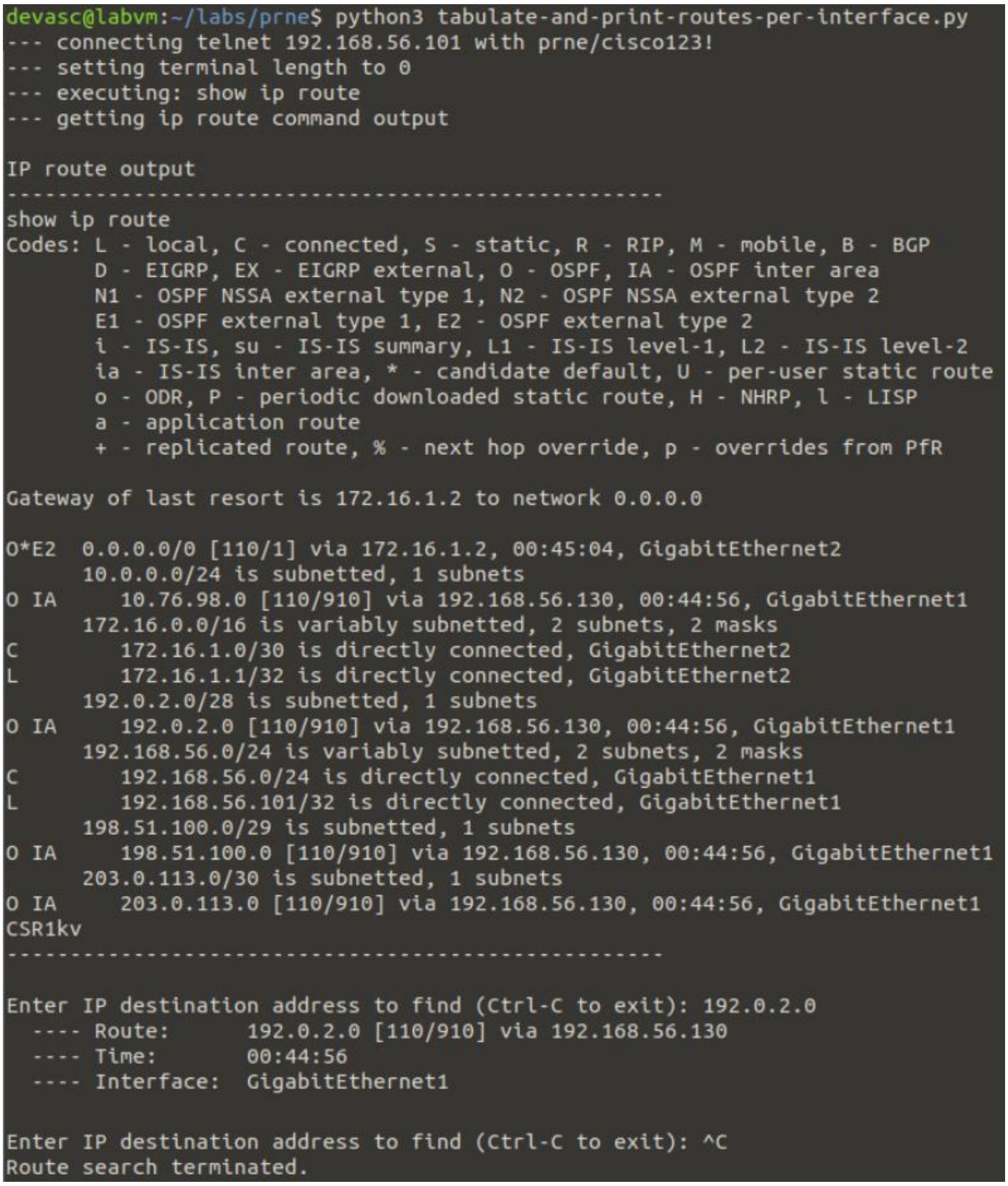
```
# Close the session
session.sendline('quit')
session.kill(0)
```

STEP 7: SAVE, RUN AND VERIFY APPLICATION

Save you application and then run it from the terminal rather than from within visual studio code.

```
~/labs/prne$ python3 tabulate-and-print-routes-per-interface.py
```

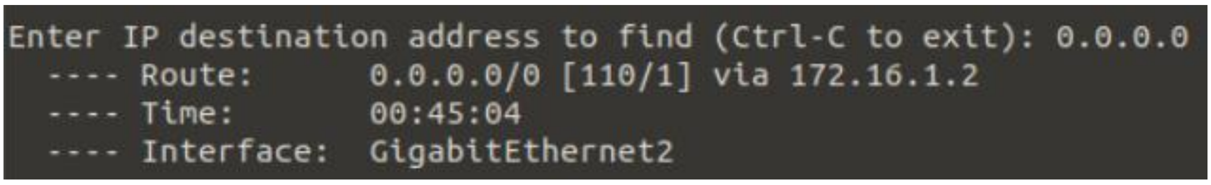
The output from your application will be displayed in your terminal window, verify that it is comparable to below.



```
devasc@labvm:~/labs/prne$ python3 tabulate-and-print-routes-per-interface.py
--- connecting telnet 192.168.56.101 with prne/cisco123!
--- setting terminal length to 0
--- executing: show ip route
--- getting ip route command output

IP route output
----------------------------------------------------
show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is 172.16.1.2 to network 0.0.0.0

O*E2  0.0.0.0/0 [110/1] via 172.16.1.2, 00:45:04, GigabitEthernet2
      10.0.0.0/24 is subnetted, 1 subnets
O IA     10.76.98.0 [110/910] via 192.168.56.130, 00:44:56, GigabitEthernet1
      172.16.0.0/16 is variably subnetted, 2 subnets, 2 masks
C        172.16.1.0/30 is directly connected, GigabitEthernet2
L        172.16.1.1/32 is directly connected, GigabitEthernet2
      192.0.2.0/28 is subnetted, 1 subnets
O IA     192.0.2.0 [110/910] via 192.168.56.130, 00:44:56, GigabitEthernet1
      192.168.56.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.56.0/24 is directly connected, GigabitEthernet1
L        192.168.56.101/32 is directly connected, GigabitEthernet1
      198.51.100.0/29 is subnetted, 1 subnets
O IA     198.51.100.0 [110/910] via 192.168.56.130, 00:44:56, GigabitEthernet1
      203.0.113.0/30 is subnetted, 1 subnets
O IA     203.0.113.0 [110/910] via 192.168.56.130, 00:44:56, GigabitEthernet1
CSR1kv
----------------------------------------------------

Enter IP destination address to find (Ctrl-C to exit): 192.0.2.0
  ---- Route:      192.0.2.0 [110/910] via 192.168.56.130
  ---- Time:       00:44:56
  ---- Interface:  GigabitEthernet1

Enter IP destination address to find (Ctrl-C to exit): ^C
Route search terminated.
```

STEP 8: CHALLENGE (OPTIONAL)

Change the pattern for matching OSPF routes to also return the details for the default (0.0.0.0) route, so the output is comparable to below, whilst also still returning the other OSPF routes.



```
Enter IP destination address to find (Ctrl-C to exit): 0.0.0.0
  ---- Route:      0.0.0.0/0 [110/1] via 172.16.1.2
  ---- Time:       00:45:04
  ---- Interface:  GigabitEthernet2
```

PART 4 (OPTIONAL BUT HIGHLY RECOMMENDED)

As this lab is completed in NETLAB+ and your code files will be erased when the reservation ends, it is advisable to save your files in GitHub under your repository for this course.