National University of Singapore

School of Computing

CS4226 **Programming Assignment 2** Semester 1 AY23/24

Submission Deadline

19th November 2023 (Sunday) 11:59pm sharp. A penalty of 2 marks will be imposed on late submission (late submission refers to submission or re-submission after the deadline). The submission folder will be closed on **26th November 2023 (Sunday) 11:59pm** sharp and no late submission will be accepted afterwards.

Objectives

Welcome to CS4226 Programming Assignment 2. In this assignment, you will use Mininet and FRR to simulate the distribution of routing information between autonomous systems through BGP. This assignment will help you to gain familiarity with configuring and debugging BGP peering sessions.

This assignment is worth **10 marks** in total and they are split across **four exercises**. All the work in this assignment shall be completed **individually**.

Plagiarism Warning

You are free to discuss this assignment with your friends. **However, you should not share your answers, screenshots, or network logs.** We highly recommend that you attempt this assignment on your own. There are many resources available online to help you along the way, and the troubleshooting skills you learn will be very useful.

We employ zero-tolerance policy against plagiarism. If a suspicious case is found, the student would be asked to explain their answers to the evaluator in person. Any confirmed breach may result in zero marks being awarded for the assignment and further disciplinary action from the school.

Question & Answer

The tutorial covers most features required in this assignment. If you encounter any problems, we suggest you to lookup the examples shown in the tutorial. You can find the features provided by FRR and their API in its official documentation at https://docs.frrouting.org/en/latest/.

If you still have doubts about this assignment, please post your questions in the "Discussion" section of Canvas or consult the teaching team. However, the teaching team will NOT debug issues for students. The intention of Q&A is to help clarify misconceptions or give you necessary directions.

Submission

You are required to submit a .zip file containing the files required for your implementation. The name of your project folder should be cs4226_bgp_<STUDENT ID>.zip. Replace <STUDENT ID> with your matric number (starting with A), for example, cs4226_bgp_A0123456A.zip.

Note: We have released the **Template folder that contains** both **auto-test scripts** and **programming assignment folder (i.e., A01234567)**, and you will only need to submit **programming assignment folder**.

The archive should have the following structure:

```
.
├── topology.py
├── router.py
├── r110/
│   └── frr.conf
├── r120/
│   └── frr.conf
├── r130/
│   └── frr.conf
├── r210/
│   └── frr.conf
├── r310/
│   └── frr.conf
└── r410/
    └── frr.conf
```

Task 1 – Building the Virtual Network (3 Marks)

Build a virtual network in Mininet according to Figure 1. The router names, hosts names and interface numbers should be consistent with those given in Figure 1. The IP addresses and subnet masks for each interface can be found in Table 1.

Use the provided `FRRRouter` class to bootstrap the router nodes and configure their loopback addresses.

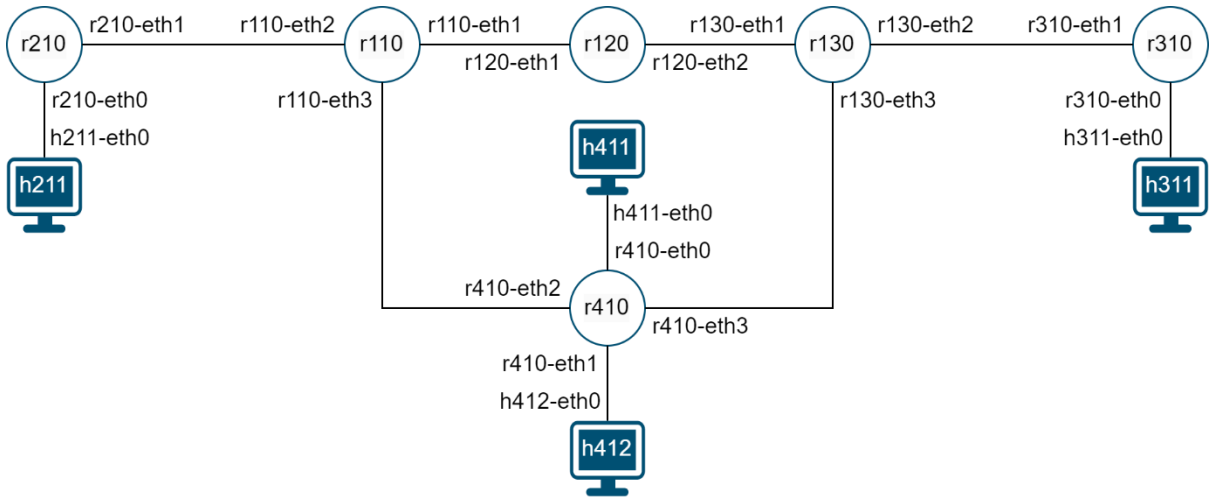

Figure 1: Network Topology

Node	Interface	IP Address
r110	loopback	100.100.1.1/32
	r110-eth1	192.168.1.0/31
	r110-eth2	172.17.1.0/31
	r110-eth3	172.17.3.0/31
r120	loopback	100.100.1.2/32
	r120-eth1	192.168.1.1/31
	r120-eth2	192.168.1.2/31
r130	loopback	100.100.1.3/32
	r130-eth1	192.168.1.3/31
	r130-eth2	172.17.2.0/31
	r130-eth3	172.17.4.0/31
r210	loopback	100.100.2.1/32
	r210-eth0	10.2.1.254/24
	r210-eth1	172.17.1.1/31
r310	loopback	100.100.3.1/32
	r310-eth0	10.3.1.254/24
	r310-eth1	172.17.2.1/31
r410	loopback	100.100.4.1/32
	r410-eth0	10.4.1.126/25
	r410-eth1	10.4.1.254/25
	r410-eth2	172.17.3.1/31
	r410-eth3	172.17.4.1/31

Node	Interface	IP Address
h211	h211-eth0	10.2.1.1/24
h311	h311-eth0	10.3.1.1/24
h411	h411-eth0	10.4.1.1/25
h412	h412-eth0	10.4.1.129/25

Table 1: Interface IP Address Assignment

Task 2 – Setting Up Routing Protocols (3 Marks)

This task is comprised of three sub tasks.

1. Configure the routers in AS100, as shown in Figure 2, to use RIP as the intra-domain routing protocol. The routers should be able to reach each other through their loopback addresses. The routers should only advertise their loopback addresses using RIP.
2. Configure all routers to run BGP and set up the necessary BGP peering sessions between the routers.
   iBGP sessions should be formed between the loopback addresses of routers.
3. Configure all routers to advertise their respective IP subnets to their BGP neighbors. All nodes in the network should be able to ping each other.
   Routers should advertise the subnet of any connected interface, except for their loopback addresses.

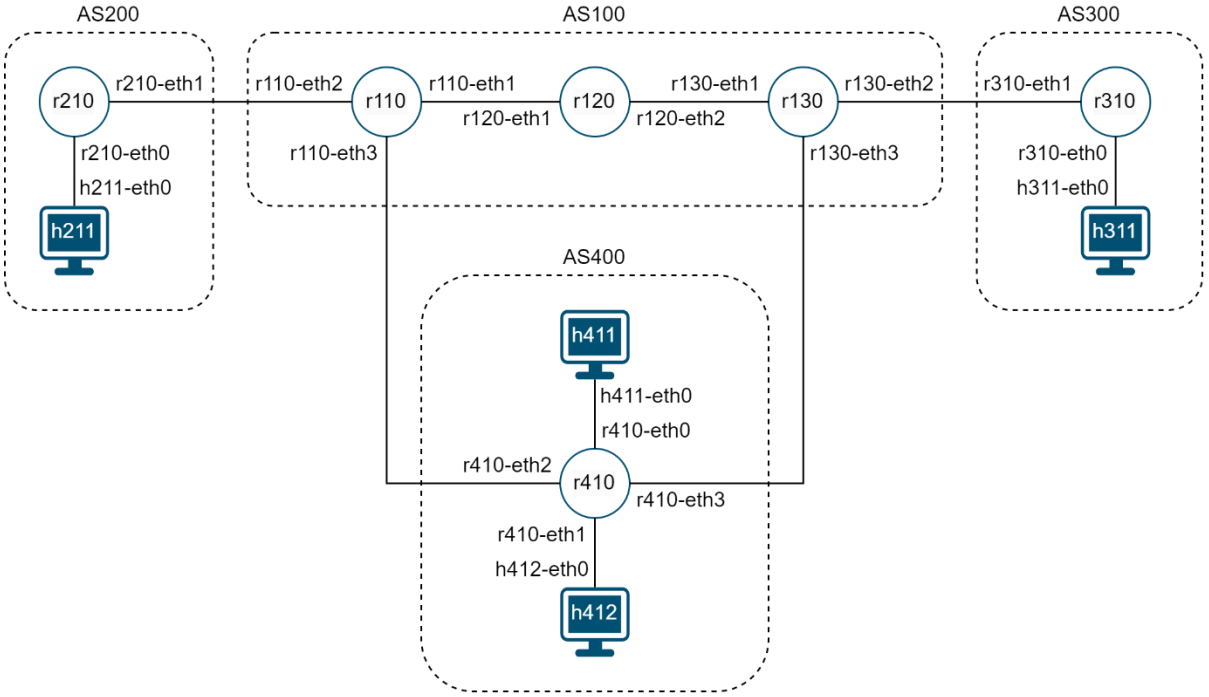

Figure 2: Autonomous Systems

Points to Note:

- Routers in AS200, AS300, and AS400 should not run RIP.
- All IP subnets should be advertised using the `network` command. Do not use the `redistribute` command.
- You may use the `no bgp ebgp-requires-policy` command, if necessary.

Task 3 – Multi Exit Discriminator (2 Marks)

AS100 is connected to AS400 by two links, one via r110 and the other via r130. This task aims to configure the two links in a failover configuration for traffic originating from AS400. The link going through r110, as highlighted in Figure 3, should be the primary link for traffic originating from AS400. The link going through r130 should be configured to be the backup link that is to be used when the link between r110 and r410 fails.

This setup must be achieved by using the Multi-Exit Discriminator (MED) attribute in BGP advertisements. Configure r110 and r130 to set MED values when announcing routes to r410. (You **must** configure both r110 and r130 to set MED values on routes announced to r410.)

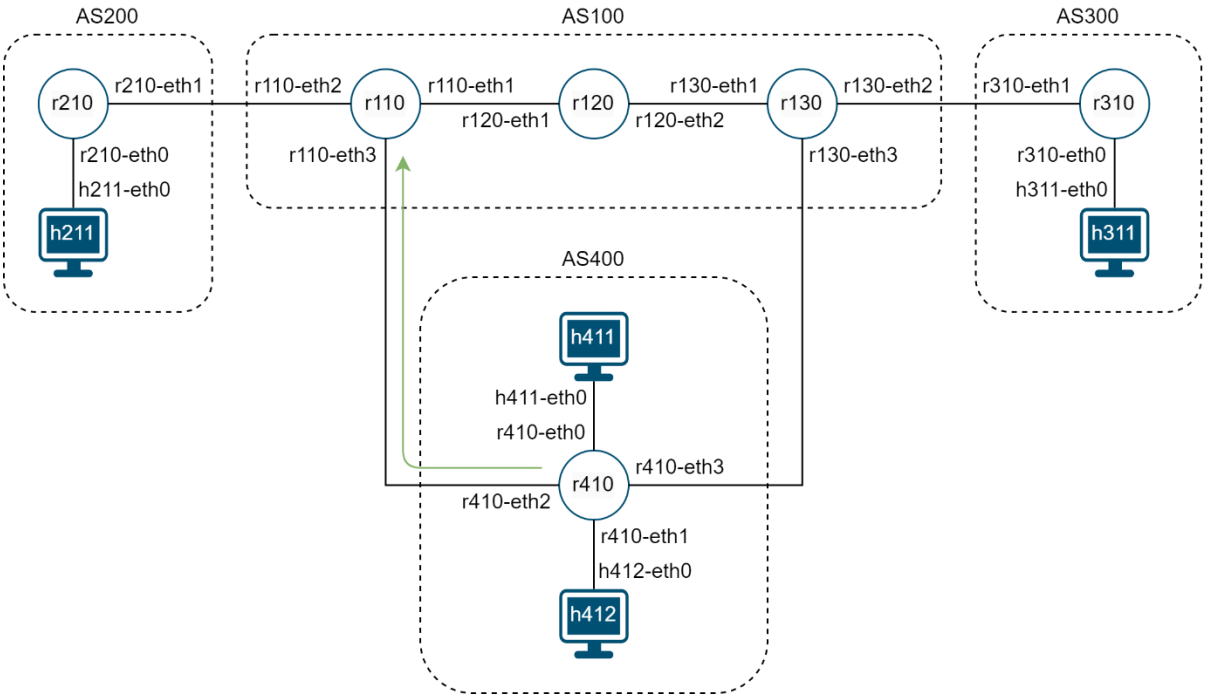

Figure 3: Failover Setup

After r110 and r130 have been configured, all packets originating from AS400 should be routed through r110, if the link between r110 and r410 is up. However, if the link between r110 and r410 is down, nodes in AS400 should still be able to reach all other nodes by routing their packets through r130.

You may wish to use the `traceroute` and `link ... down` commands to test your implementation.

(Ungraded) Exploration: In task 2, the failover configuration was achieved using the MED attribute. This attribute is used by the upstream AS to communicate its preference of primary and backup routes to the downstream AS. How can downstream ASes overwrite this preference?

Task 4 – Community & Local Preference (2 Marks)

In the previous task, the two links between AS100 and AS400 were configured in a failover configuration for traffic originating from AS400.

In this task, the two links shall be configured in a load balance configuration for traffic being sent to AS400, as shown in Figure 4. Traffic sent to h411 should be routed over the link between r110 and r410, while traffic being sent to h412 should be sent over the link between r130 and r410.

This setup must be achieved using community values, local preference settings, and route-maps using the following steps:

1. Configure r110 and r130 to map the following community values to their local preference values.

Community	Local Preference
400:100	100
400:300	300

2. Configure r410 to set the correct community values on advertised routes.

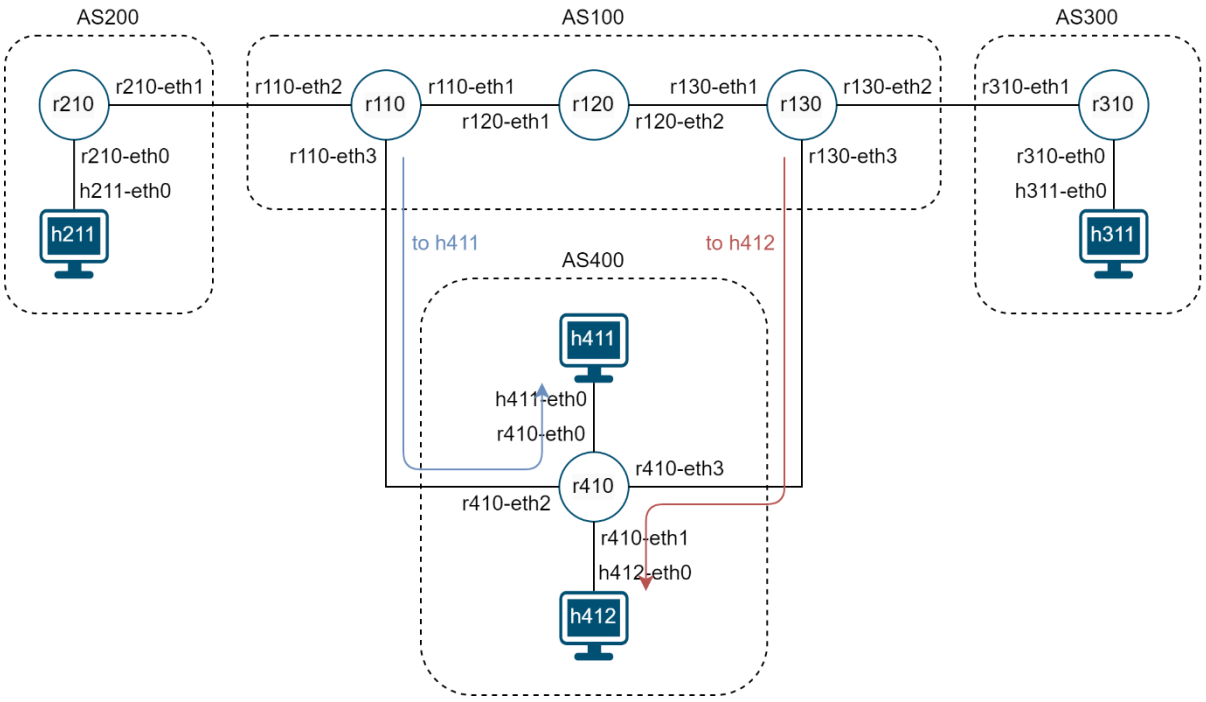

Figure 4: Load Balanced Setup

After the necessary routers have been configured, the routing table in r120 should show that the best path to h411 has r110 as its next hop while the best path to h412 has r130 as its next hop.

You may wish to use the `show ip route` command to test your implementation.

(Ungraded) Exploration: In task 3, the load balancing configuration was achieved using community values, local preference settings, and route-maps. This requires coordination between ASes. Can the same effect be achieved by only configuring the router in AS400?

Testing your code

You will need to set up a **Mininet VM** and install **FRRRouter** related environment following the instructions in this section.

Step 1: Environment setup

Before using the test script, make sure you have installed all required tools:

```
sudo apt-get update
sudo apt-get install mininet openvswitch-testcontroller frr traceroute
-y
```

Note: You can also choose to use the Mininet VM to skip the environment configuration for Mininet. But it still needs to set up other dependencies, i.e., **openvswitch-testcontroller, frr, and traceroute**.

During the installation of the OpenVSwitch controller and FRR, some services will be configured to start on boot. We must disable these services so that they will not conflict when running Mininet.

```
sudo systemctl stop openvswitch-testcontroller frr
sudo systemctl disable openvswitch-testcontroller frr
```

Step 2: Move your code to the VM

You can simply move the released project folder to the VM root folder using any file transmission tools, such as **scp** or **PuTTY**.

We show an example of using scp that is natively supported by Linux, MacOS, and WSL here.

Before using scp, you may need to:

1. Check whether SSH service is enabled, you can simply install the SSH daemon by:

```
sudo apt-get install ssh
```

2. Make sure the **host-only network adapter** is enabled for the VM to ensure the supervisor (i.e., your localhost can access the VM).

Then, you can use the following command to transfer the project folder to the VM.

```
scp -r PROJECT_FOLDER user_name@VM_IP_ADDRESS:~/
```

Note: the **`user_name`** is the user name configured for the Mininet VM, and **`VM_IP_ADDRESS`** is the associated IP address. You will need to enter your VM password after executing the command.

Step 3: Build Mininet topology and try functionalities

Go to the **programming assignment folder (**i.e., **Template/A01234567/)**, and then, you can build the Mininet topology and launch the Mininet CLI using the following command:

```
sudo python3 topology.py
```

Then, you may try the associated Mininet CLI commands in **Appendix** to verify the implementation.

Step 4: Scripts for testing

Go back to the project root folder (i.e., **Template/**), and then you can test all functionalities using our testing script **autotester.py**.

You can run the **autotester.py** by using:

```
sudo python3 autotester.py PROJECT_FOLDER_NAME [--show-output]
```

Note: you can use `--show-output` to get detailed command execution results.

It will check the implementation following three parts :

Task 1

- Check Mininet Topology.

Task 2

- Check active protocols.
- Check BGP ASN.
- Check connectivity.
- Check BGP neighbors.

Task 3

- Check fault tolerance
- Check metric values, i.e., show BGP ipv4 unicast info.

Task 4

- Check route.

Appendix

The expected execution log can be shown as follows, and you can also find the commands to test each individual task accordingly.

```
/home/myc/BGP-Assignment/Grading
(A01234567) GRADING

############################## Starting test cases for Task 1 ##############################
(A01234567) PASSED: Topology check

############################## Starting test cases for Task 2 ##############################
++++++ check_active_protocols: r110 command: vtysh -c "show ip rip" expected (is instance
found): True
++++++ check_active_protocols: r110 command: vtysh -c "show ip bgp" expected (is instance
found): True
++++++ check_active_protocols: r120 command: vtysh -c "show ip rip" expected (is instance
found): True
++++++ check_active_protocols: r120 command: vtysh -c "show ip bgp" expected (is instance
found): True
++++++ check_active_protocols: r130 command: vtysh -c "show ip rip" expected (is instance
found): True
++++++ check_active_protocols: r130 command: vtysh -c "show ip bgp" expected (is instance
found): True
++++++ check_active_protocols: r210 command: vtysh -c "show ip rip" expected (is instance
found): False
++++++ check_active_protocols: r210 command: vtysh -c "show ip bgp" expected (is instance
found): True
++++++ check_active_protocols: r310 command: vtysh -c "show ip rip" expected (is instance
found): False
++++++ check_active_protocols: r310 command: vtysh -c "show ip bgp" expected (is instance
found): True
++++++ check_active_protocols: r410 command: vtysh -c "show ip rip" expected (is instance
found): False
++++++ check_active_protocols: r410 command: vtysh -c "show ip bgp" expected (is instance
found): True
(A01234567) PASSED: Protocol check
++++++ check_bgp_asn: r110 command: vtysh -c "show bgp summary" expected: local AS number
100
++++++ check_bgp_asn: r120 command: vtysh -c "show bgp summary" expected: local AS number
100
++++++ check_bgp_asn: r130 command: vtysh -c "show bgp summary" expected: local AS number
100
++++++ check_bgp_asn: r210 command: vtysh -c "show bgp summary" expected: local AS number
200
++++++ check_bgp_asn: r310 command: vtysh -c "show bgp summary" expected: local AS number
300
++++++ check_bgp_asn: r410 command: vtysh -c "show bgp summary" expected: local AS number
400
(A01234567) PASSED: ASN check
++++++ check_connectivity: pingall expected: all sucess
```

++++++ check_connectivity: check AS100 RIP connectivity expected: all success
++++++ check_bgp_asn: r110 command: ping 100.100.1.2 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_bgp_asn: r110 command: ping 100.100.1.3 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_bgp_asn: r120 command: ping 100.100.1.1 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_bgp_asn: r120 command: ping 100.100.1.3 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_bgp_asn: r130 command: ping 100.100.1.1 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_bgp_asn: r130 command: ping 100.100.1.2 -W 1 -w 1 -c 1 expected: 0% packet loss
++++++ check_connectivity: check other AS RIP connectivity expected: all fail
++++++ check_bgp_asn: r120 command: ping 100.100.1.4 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r120 command: ping 100.100.1.5 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r120 command: ping 100.100.1.6 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r210 command: ping 100.100.1.1 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r210 command: ping 100.100.1.2 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r210 command: ping 100.100.1.3 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r310 command: ping 100.100.1.1 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r310 command: ping 100.100.1.2 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r310 command: ping 100.100.1.3 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r410 command: ping 100.100.1.1 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r410 command: ping 100.100.1.2 -W 1 -w 1 -c 1 expected: Network is unreachable
++++++ check_bgp_asn: r410 command: ping 100.100.1.3 -W 1 -w 1 -c 1 expected: Network is unreachable
(A01234567) PASSED: Connectivity check
++++++ check_bgp_neighbors: r110 command: vtysh -c "show bgp summary" expected: include 100.100.1.2,100.100.1.3,172.17.1.1,172.17.3.1
++++++ check_bgp_neighbors: r110 command: vtysh -c "show bgp summary" expected: exclude 192.168.1.1
++++++ check_bgp_neighbors: r120 command: vtysh -c "show bgp summary" expected: include 100.100.1.1,100.100.1.3
++++++ check_bgp_neighbors: r120 command: vtysh -c "show bgp summary" expected: exclude 192.168.1.0,192.168.1.3
++++++ check_bgp_neighbors: r130 command: vtysh -c "show bgp summary" expected: include 100.100.1.1,100.100.1.2,172.17.2.1,172.17.4.1
++++++ check_bgp_neighbors: r130 command: vtysh -c "show bgp summary" expected: exclude 192.168.1.2
++++++ check_bgp_neighbors: r210 command: vtysh -c "show bgp summary" expected: include 172.17.1.0
++++++ check_bgp_neighbors: r310 command: vtysh -c "show bgp summary" expected: include 172.17.2.0
++++++ check_bgp_neighbors: r410 command: vtysh -c "show bgp summary" expected: include 172.17.3.0,172.17.4.0

(A01234567) PASSED: Neighbour check

############################## **Starting test cases for Task 3** ##############################
++++++ check_fault_tolerance: r410 command: traceroute 192.168.1.1 expected: 172.17.3.0
++++++ check_fault_tolerance: r410 command (after link r110 r410 down): traceroute 192.168.1.1 expected: 172.17.4.0
++++++ check_fault_tolerance: pingall expected: all sucess
++++++ check_fault_tolerance: r410 command (after link r110 r410 up): traceroute 192.168.1.1 expected: 172.17.3.0
++++++ check_fault_tolerance: pingall expected: all sucess
(A01234567) PASSED: Fault tolerance check
++++++ check_metric_values: r410 command: vtysh -c "show bgp ipv4 unicast all" expected: 172.17.3.0 0
++++++ check_metric_values: r410 command: vtysh -c "show bgp ipv4 unicast all" expected: 172.17.4.0 0
++++++ check_metric_values: r210 command: vtysh -c "show bgp ipv4 unicast all" expected: 172.17.1.0 0
++++++ check_metric_values: r310 command: vtysh -c "show bgp ipv4 unicast all" expected: 172.17.2.0 0
(A01234567) PASSED: Metric value check

############################## **Starting test cases for Task 4** ##############################
++++++ check_route: r120 command: r120 ip route expected: output match reg 10.4.1.0\/25 nhid \d+ via 192.168.1.0
++++++ check_route: r120 command: r120 ip route expected: output match reg 10.4.1.128\/25 nhid \d+ via 192.168.1.3
(A01234567) PASSED: Route check
(A01234567) PASSED: Restricted commands check
(A01234567) *** Summary ***
(A01234567) ALL PASSED