

COM2022: Computer Network Report

Contents

Task A: Network topology implementation in NetKit [32 marks]	2
T-A0: Created folder "nk00374"	2
T-A1: Configuring startup files [9 marks]	2
T-A2: Lab configuration [2 marks]	3
T-A3: Setting up routing service [12 marks]	4
T-A4: Setting up apache by editing the server.startup file. [3 marks]	4
T-A5: Adding html file to Apache server [2 marks]	4
T-A6: Testing the network [2 marks]	5
T-A7: Graph [2 marks]	6
Task B: Network extension [32 marks]	7
T-B1: Network Topology [4 marks]:	7
T-B2: Questions [12 marks]	8
T-B3: Implement changes in Netkit [14 marks]	9
T-B4/T-B3 Continued: Testing [2 marks]	11
Task C: Routing experiments [8 marks]	15
T-C1: OSPF vs RIP [4 marks]	15
T-C2: The preferred route in the network [4 marks]	18
Task D: DNS server [8 marks]	19
T-D1: Add DNS servers on to the network [4 marks]	20
T-D2: DNS functions and settings [4 marks]	21
Task E: Network analysis [20 marks]	22
T-E1: Test connectivity [4 marks]	22
T-E2: Routing Procedure [4 marks]	23
T-E3: Effect of shutting down r1 [4 marks]	24
T-E4: Effect of changing interface cost [4 marks]	25
T-E5: Connecting Server to network C [4 marks]	26
References:	29

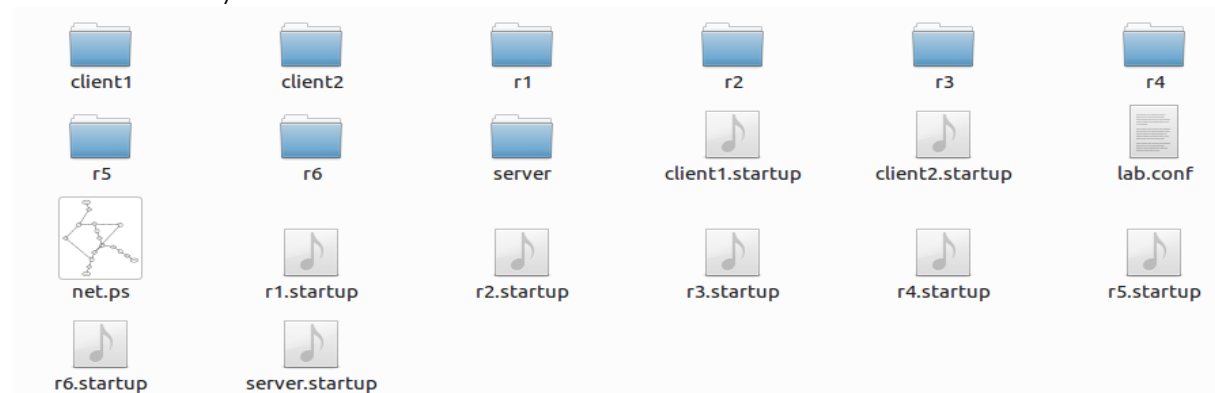
NOTE: Used 74 for both routing cost and IP address

Username: nk00374 URN: 6474079

Task A: Network topology implementation in NetKit [32 marks]

T-A0: Created folder "nk00374"

Screenshot of my final folder contents.



T-A1: Configuring startup files [9 marks]

```
client2.startup
1 ifconfig eth0 151.2.74.2 netmask 255.255.255.0 broadcast 151.2.74.255 up
2 route add default gw 151.2.74.1 dev eth0
-

client1.startup
1 ifconfig eth0 151.4.74.2 netmask 255.255.255.0 broadcast 151.4.74.255 up
2 route add default gw 151.4.74.1 dev eth0

r1.startup
1 ifconfig eth0 151.0.74.1 netmask 255.255.255.0 broadcast 151.0.74.255 up
2 ifconfig eth1 151.5.74.1 netmask 255.255.255.0 broadcast 151.5.74.255 up
3 /etc/init.d/zebra start

r2.startup
1 ifconfig eth0 151.0.74.2 netmask 255.255.255.0 broadcast 151.0.74.255 up
2 ifconfig eth1 151.1.74.1 netmask 255.255.255.0 broadcast 151.1.74.255 up
3 ifconfig eth2 110.2.74.2 netmask 255.255.255.252 broadcast 110.2.74.15 up
4 /etc/init.d/zebra start

r3.startup
1 ifconfig eth0 151.5.74.3 netmask 255.255.255.0 broadcast 151.1.74.255 up
2 ifconfig eth1 151.1.74.3 netmask 255.255.255.0 broadcast 151.5.74.255 up
3 /etc/init.d/zebra start
```

```

r4.startup
1 ifconfig eth0 151.5.74.2 netmask 255.255.255.0 broadcast 151.5.74.255 up
2 ifconfig eth1 151.4.74.1 netmask 255.255.255.0 broadcast 151.4.74.255 up
3 ifconfig eth2 151.3.74.1 netmask 255.255.255.0 broadcast 151.3.74.255 up
4 /etc/init.d/zebra start

r5.startup
1 ifconfig eth0 151.1.74.2 netmask 255.255.255.0 broadcast 151.1.74.255 up
2 ifconfig eth1 151.3.74.2 netmask 255.255.255.0 broadcast 151.3.74.255 up
3 ifconfig eth2 151.2.74.1 netmask 255.255.255.0 broadcast 151.2.74.255 up
4 /etc/init.d/zebra start

r6.startup
1 ifconfig eth0 210.2.74.2 netmask 255.255.255.252 broadcast 151.1.74.3 up
2 ifconfig eth1 110.2.74.1 netmask 255.255.255.252 broadcast 110.2.74.3 up
3 /etc/init.d/zebra start

server.startup
1 ifconfig eth0 210.2.74.1 netmask 255.255.255.252 broadcast 210.2.74.3 up
2 route add default gw 210.2.74.2 dev eth0
3 a2enmod userdir
4 /etc/init.d/apache2 start
5
```

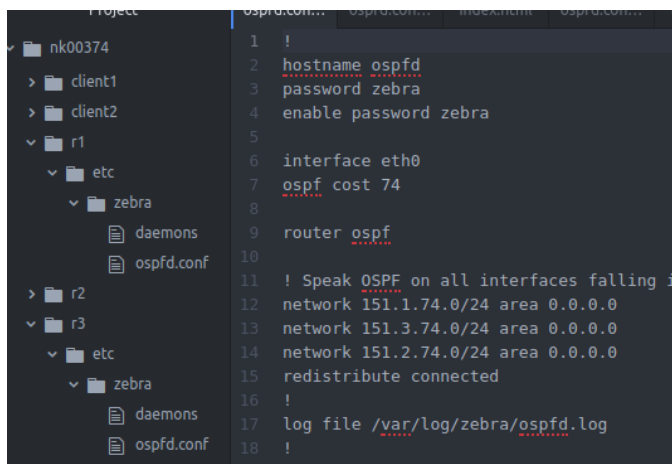
T-A2: Lab configuration [2 marks]

```

lab.conf
1
2 server[0]=H
3
4 client1[0]=D
5 client2[0]=F
6
7 r1[0]=A
8 r1[1]=B
9
10 r2[0]=A
11 r2[1]=C
12 r2[2]=G
13
14 r3[0]=B
15 r3[1]=C
16
17 r4[0]=B
18 r4[1]=D
19 r4[2]=E
20
21 r5[0]=C
22 r5[1]=E
23 r5[2]=F
24
25 r6[0]=H
26 r6[1]=G
```

T-A3: Setting up routing service [12 marks]

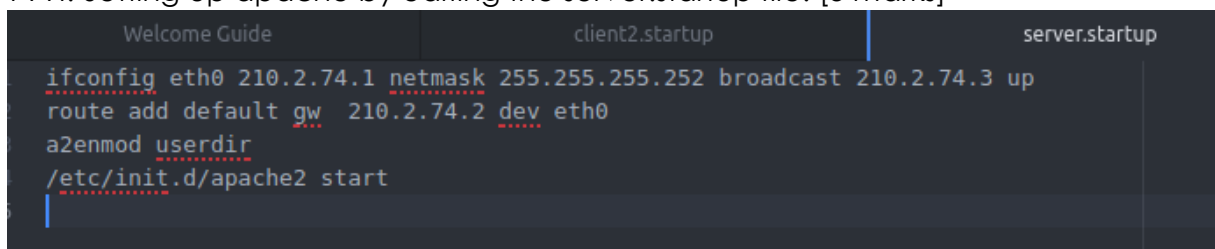
Proof of configuring routing service. I have done this for all routers, adding the appropriate interface cost to each.



The image shows a file explorer on the left with a project structure for 'nk00374'. It includes folders for 'client1', 'client2', 'r1', 'r2', and 'r3'. Under 'r1', there is an 'etc' folder containing a 'zebra' folder with 'daemons' and 'ospfd.conf' files. The main terminal window shows the configuration for 'ospfd' on router 'r1'.

```
1 !
2 hostname ospfd
3 password zebra
4 enable password zebra
5
6 interface eth0
7   ospf cost 74
8
9 router ospf
10
11 ! Speak OSPF on all interfaces falling i
12 network 151.1.74.0/24 area 0.0.0.0
13 network 151.3.74.0/24 area 0.0.0.0
14 network 151.2.74.0/24 area 0.0.0.0
15 redistribute connected
16 !
17 log file /var/log/zebra/ospfd.log
18 !
```

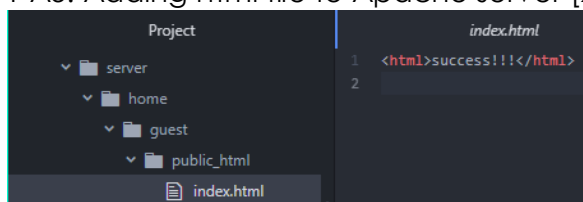
T-A4: Setting up apache by editing the server.startup file. [3 marks]



The image shows a terminal window with three tabs: 'Welcome Guide', 'client2.startup', and 'server.startup'. The 'server.startup' tab is active, showing the following commands:

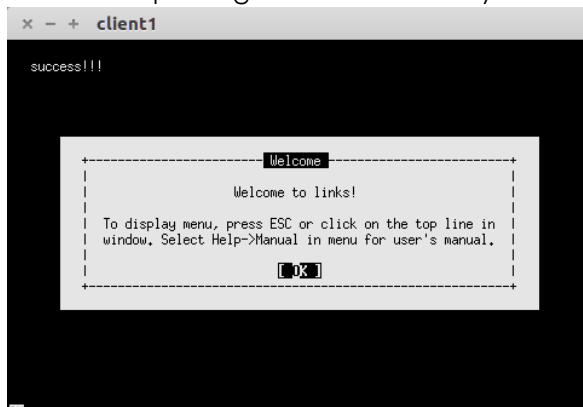
```
ifconfig eth0 210.2.74.1 netmask 255.255.255.252 broadcast 210.2.74.3 up
route add default gw 210.2.74.2 dev eth0
a2enmod userdir
/etc/init.d/apache2 start
```

T-A5: Adding html file to Apache server [2 marks]



The image shows a file explorer with a project structure. The 'server' folder is expanded, showing 'home' and 'guest' folders. The 'public_html' folder is expanded, showing the 'index.html' file.

Proof of requesting HTML file made by client



The image shows a terminal window titled 'client1'. It displays the output of a request to the Apache server, showing 'success!!!' and a 'Welcome' message.

```
success!!!

Welcome

Welcome to links!

To display menu, press ESC or click on the top line in
window. Select Help->Manual in menu for user's manual.

[ESC]
```

T-A6: Testing the network [2 marks]

Ensure that network works by correctly pinging from one node to another:

Pinging from 'client1' to 'r2' [both nodes in the same network area]:

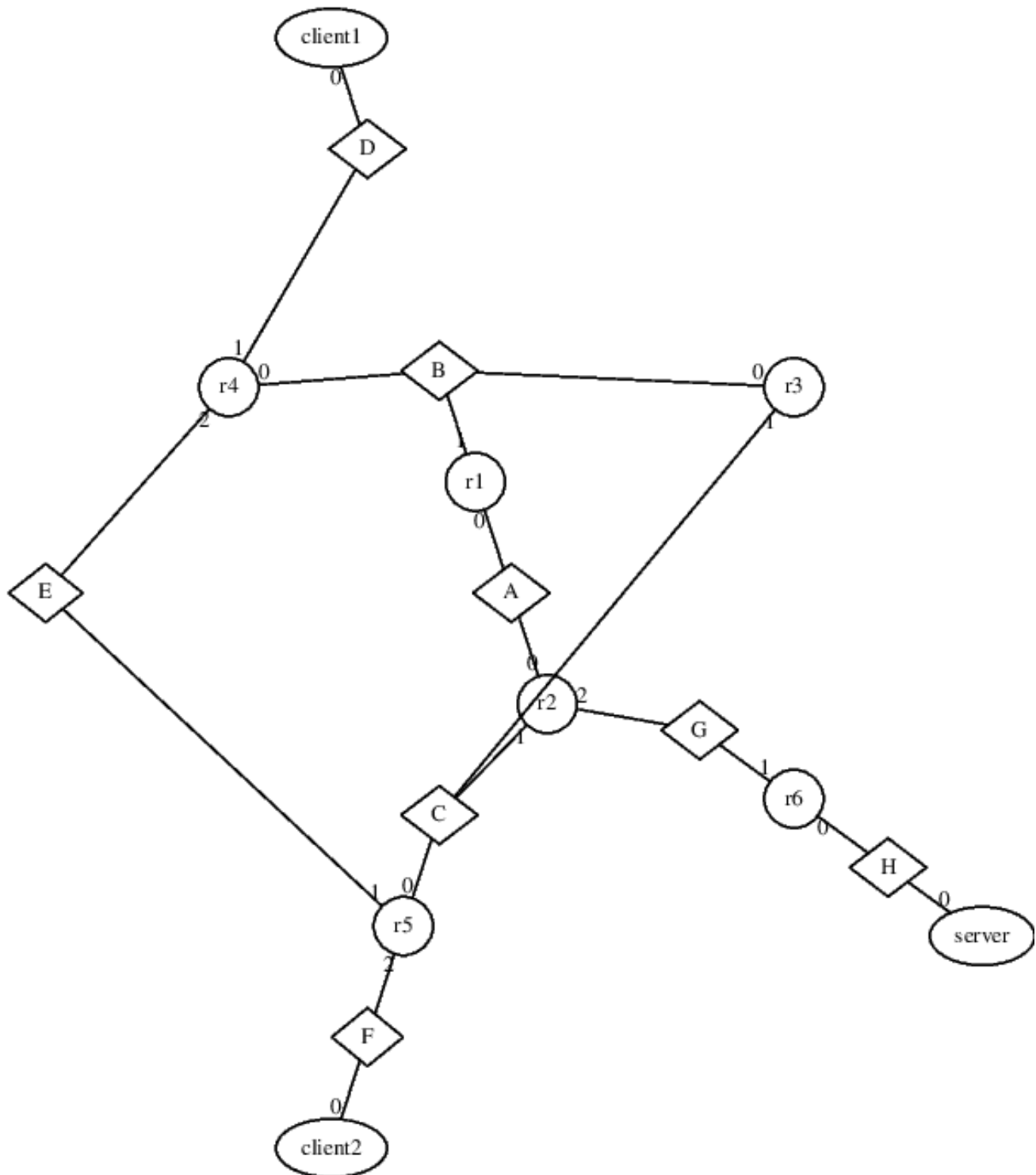
x - + client1	x - + r2
64 bytes from 151.1.74.1: icmp_seq=52 ttl=62 time=1.78 ms	14:03:52.852485 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 62, length 64
64 bytes from 151.1.74.1: icmp_seq=53 ttl=62 time=1.85 ms	14:03:53.862762 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 63, length 64
64 bytes from 151.1.74.1: icmp_seq=54 ttl=62 time=1.84 ms	14:03:54.872475 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 64, length 64
64 bytes from 151.1.74.1: icmp_seq=55 ttl=62 time=1.53 ms	14:03:55.881927 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 65, length 64
64 bytes from 151.1.74.1: icmp_seq=56 ttl=62 time=1.50 ms	14:03:56.892573 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 66, length 64
64 bytes from 151.1.74.1: icmp_seq=57 ttl=62 time=1.56 ms	14:03:57.256877 IP 151.1.74.1 > 224.0.0.5: OSPFv2, Hello, length: 52
64 bytes from 151.1.74.1: icmp_seq=58 ttl=62 time=1.71 ms	14:03:57.258191 IP 151.1.74.3 > 224.0.0.5: OSPFv2, Hello, length: 52
64 bytes from 151.1.74.1: icmp_seq=59 ttl=62 time=1.38 ms	14:03:57.819977 IP 151.1.74.2 > 224.0.0.5: OSPFv2, Hello, length: 52
64 bytes from 151.1.74.1: icmp_seq=60 ttl=62 time=1.50 ms	14:03:57.902465 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 67, length 64
64 bytes from 151.1.74.1: icmp_seq=61 ttl=62 time=0.854 ms	14:03:58.911901 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 68, length 64
64 bytes from 151.1.74.1: icmp_seq=62 ttl=62 time=1.53 ms	14:03:59.922766 IP 151.4.74.2 > 151.1.74.1: ICMP echo request, id 5378, seq 69, length 64
64 bytes from 151.1.74.1: icmp_seq=63 ttl=62 time=2.23 ms	23 packets captured
64 bytes from 151.1.74.1: icmp_seq=64 ttl=62 time=1.48 ms	23 packets received by filter
64 bytes from 151.1.74.1: icmp_seq=65 ttl=62 time=1.07 ms	0 packets dropped by kernel
64 bytes from 151.1.74.1: icmp_seq=66 ttl=62 time=1.65 ms	r2:~#
64 bytes from 151.1.74.1: icmp_seq=67 ttl=62 time=1.48 ms	
64 bytes from 151.1.74.1: icmp_seq=68 ttl=62 time=0.999 ms	
64 bytes from 151.1.74.1: icmp_seq=69 ttl=62 time=1.79 ms	
64 bytes from 151.1.74.1: icmp_seq=70 ttl=62 time=1.57 ms	
^C	
--- 151.1.74.1 ping statistics ---	
70 packets transmitted, 70 received, 0% packet loss, time 69661ms	
rtt min/avg/max/mdev = 0.283/1.911/39.098/4.490 ms	
client1:~#	

Pinging from 'client1' to 'r6' [where both nodes are in different network locations]

x - + client1	x - + r6
64 bytes from 110.2.74.1: icmp_seq=11 ttl=61 time=2.01 ms	r6:~# tcpdump -i eth1
64 bytes from 110.2.74.1: icmp_seq=12 ttl=61 time=1.41 ms	tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
64 bytes from 110.2.74.1: icmp_seq=13 ttl=61 time=1.57 ms	listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
64 bytes from 110.2.74.1: icmp_seq=14 ttl=61 time=1.79 ms	14:40:54.652686 IP 151.4.74.2 > 110.2.74.1: ICMP echo request, id 1282, seq 23, length 64
64 bytes from 110.2.74.1: icmp_seq=15 ttl=61 time=1.45 ms	14:40:54.659816 IP 110.2.74.1 > 151.4.74.2: ICMP echo reply, id 1282, seq 23, length 64
64 bytes from 110.2.74.1: icmp_seq=16 ttl=61 time=1.86 ms	14:40:55.662898 IP 151.4.74.2 > 110.2.74.1: ICMP echo request, id 1282, seq 24, length 64
64 bytes from 110.2.74.1: icmp_seq=17 ttl=61 time=1.96 ms	14:40:55.662936 IP 110.2.74.1 > 151.4.74.2: ICMP echo reply, id 1282, seq 24, length 64
64 bytes from 110.2.74.1: icmp_seq=18 ttl=61 time=1.40 ms	14:40:56.672678 IP 151.4.74.2 > 110.2.74.1: ICMP echo request, id 1282, seq 25, length 64
64 bytes from 110.2.74.1: icmp_seq=19 ttl=61 time=2.00 ms	14:40:56.672715 IP 110.2.74.1 > 151.4.74.2: ICMP echo reply, id 1282, seq 25, length 64
64 bytes from 110.2.74.1: icmp_seq=20 ttl=61 time=1.39 ms	14:40:57.682183 IP 151.4.74.2 > 110.2.74.1: ICMP echo request, id 1282, seq 26, length 64
64 bytes from 110.2.74.1: icmp_seq=21 ttl=61 time=1.55 ms	14:40:57.682211 IP 110.2.74.1 > 151.4.74.2: ICMP echo reply, id 1282, seq 26, length 64
64 bytes from 110.2.74.1: icmp_seq=22 ttl=61 time=1.57 ms	^C
64 bytes from 110.2.74.1: icmp_seq=23 ttl=61 time=1.49 ms	8 packets captured
64 bytes from 110.2.74.1: icmp_seq=24 ttl=61 time=1.95 ms	8 packets received by filter
64 bytes from 110.2.74.1: icmp_seq=25 ttl=61 time=1.73 ms	0 packets dropped by kernel
64 bytes from 110.2.74.1: icmp_seq=26 ttl=61 time=1.37 ms	r6:~#
64 bytes from 110.2.74.1: icmp_seq=27 ttl=61 time=2.02 ms	
64 bytes from 110.2.74.1: icmp_seq=28 ttl=61 time=1.81 ms	
64 bytes from 110.2.74.1: icmp_seq=29 ttl=61 time=1.95 ms	
^C	
--- 110.2.74.1 ping statistics ---	
29 packets transmitted, 29 received, 0% packet loss, time 28286ms	
rtt min/avg/max/mdev = 1.017/1.965/10.792/1.690 ms	
client1:~#	

T-A7: Graph [2 marks]

Save an image (net.ps) of the graph.



Linfo generates a topology map of our network. We can see the clients, virtual machines, collision domains, servers outputted on the graph. The numbers represent the ethernet connections at a node.

Task B: Network extension [32 marks]

T-B1: Network Topology [4 marks]:

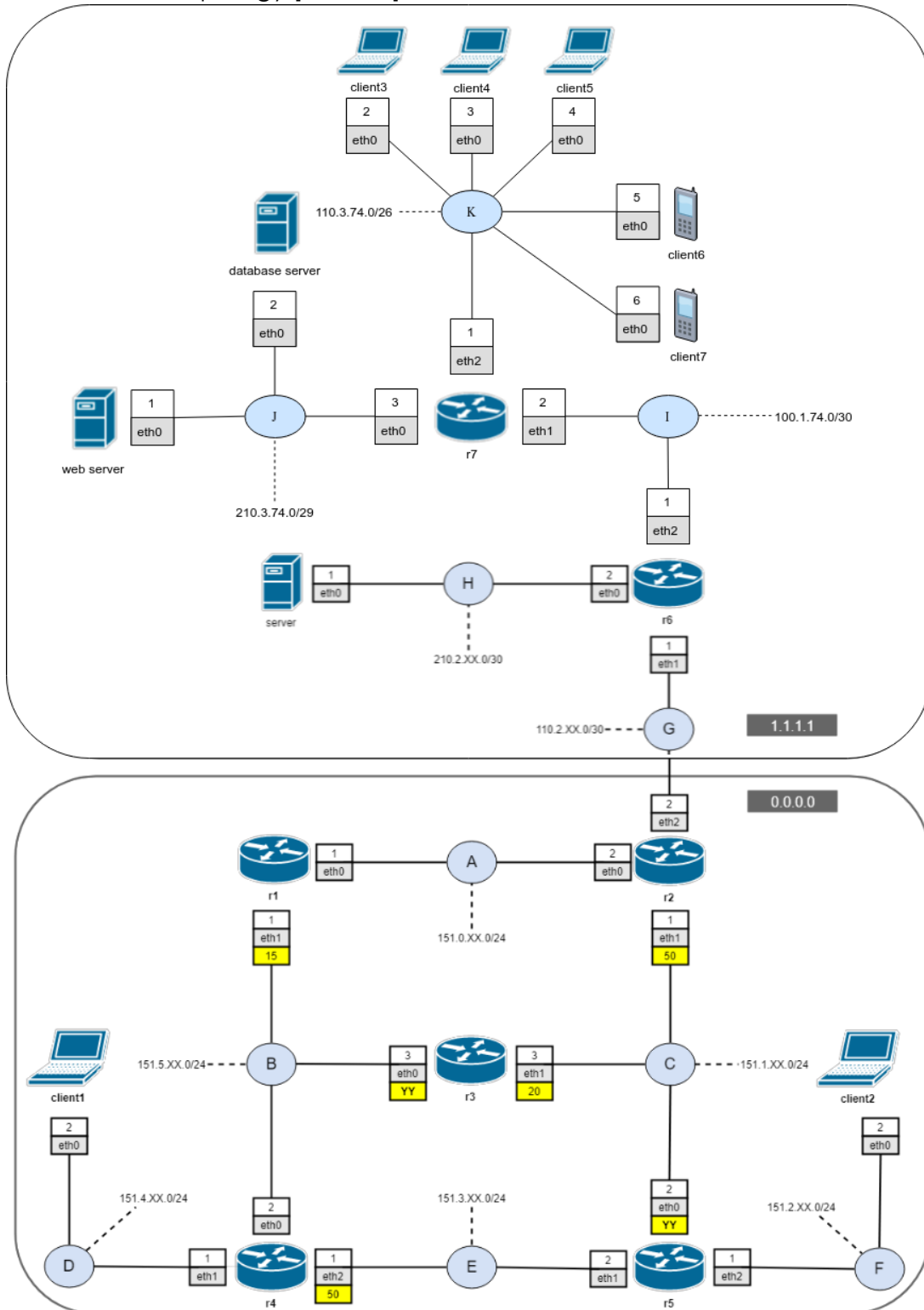


Diagram Description: I added r7 to network I so that I can create 2 subnets. I have selected the networks carefully and appropriately so they can hold the correct number of hosts. Subnet J will connect r7 to the servers (web server and database server) while subnet K will connect the clients (the three laptops and 2 smart phones) to r7. I have also assumed that the network extension belongs to the same network area 1.1.1.1. For network K, its subnet will be able to hold up to 62 devices whilst network J will hold up to 6 devices. For the ip addresses for J and K I simply followed the same naming process as network G and H, where it goes 110.2.73.0/30 for H but then it goes to 210.2.74.0/30. Through this logic I decided to keep J as 210.3.74.0/29 and K as 110.3.74.0/26. I was not sure if this was correct, though upon implementation I found that this still works.

UPDATE: after completing my coursework and looking back at Task B perhaps subnet J should be called 100.2.74.0/29 and subnet K be called 100.3.74.0/29 since subnetting is where we essentially split the network.

T-B2: Questions [12 marks]

a. I designed the network as it is due to the constraint that the business should have two subnets set up. In order to do this, I connected a new router to network so that I can add the two subnets to the new router (r7). I then gave each subnet an appropriate, valid IP address such that for network J it will hold up to 6 hosts so we can add the web server, database server and r7 to network J. For network K the IP allows the network to hold up to 62 devices allowing for the 50 clients to be added to the subnet. After adding the subnets, I connected the business's equipment to the corresponding subnet. I added both the servers to subnet J and then added all the clients (3 laptops and 2 smart phones) on to subnet K.

b.

Calculating IP for network I:

Address space: 100.1.74.0/30

Address space in Binary: 01100100.00000001.01001010.00000000

Subnet mask: 255.255.255.252

Subnet mask in Binary: 11111111.11111111.11111111.11111100

Network address: 100.1.74.0, calculated by setting all the bits to 0 after the 30th bit.

Network address in Binary: 01100100.00000001.01001010.000000 | 00

Broadcast address: 100.1.74.3, calculated by setting all the bits to 1 after the 30th bit

Broadcast address in Binary: 01100100.00000001.01001010.000000 | 11

Host Range is: 100.1.74.1 -----100.1.74.2,

Min Host: 100.1.74.1

Max Host: 100.1.74.2

From this we can calculate we can only have 2 hosts.

Calculating IP for network J:

Address space: 210.3.74.0/29

Address space in Binary: 11010010.00000011.01001010.00000000

Subnet mask: 255.255.255.248

Subnet mask in Binary: 11111111.11111111.11111111.11111000

Network address: 210.3.74.0, calculated by setting all the bits to 0 after the 29th bit.

Network address in Binary: 11010010.00000011.01001010.000000 | 000

Broadcast address: 210.3.74.7, calculated by setting all the bits to 1 after the 29th bit.

Broadcast address in binary: 11010010.00000011.01001010.000000 | 111

Host Range is: 210.3.74.0 ----- 210.3.74.6,

Min Host: 210.3.74.1

Max Host: 210.3.74.6

From this we can calculate we can only have 6 hosts on network J. This is for the web server and database server to connect to the database.

Calculating IP for network K:

Address space: 110.3.74.0/26

Address space in Binary: 01101110.00000011.01001010.00000000

Subnet mask: 255.255.255.192

Subnet mask in Binary: 11111111.11111111.11111111.11000000

Network address: 110.3.74.0, calculated by setting all the bits to 0 after the 26th bit.

Network address in binary: 01101110.00000011.01001010.00 | 000000

Broadcast address: 110.3.74.63, calculated by turning everything after the 29th bit to 1's

Broadcast address in binary: 01101110.00000011.01001010.00 | 111111

Host Range is: 110.3.74.0 ----- 110.3.74.63,

Min Host: 110.3.74.1

Max Host: 110.3.74.62

From this we can calculate we can only have 62 hosts on network J. This satisfies the constraint allowing 50 devices to connect to the network

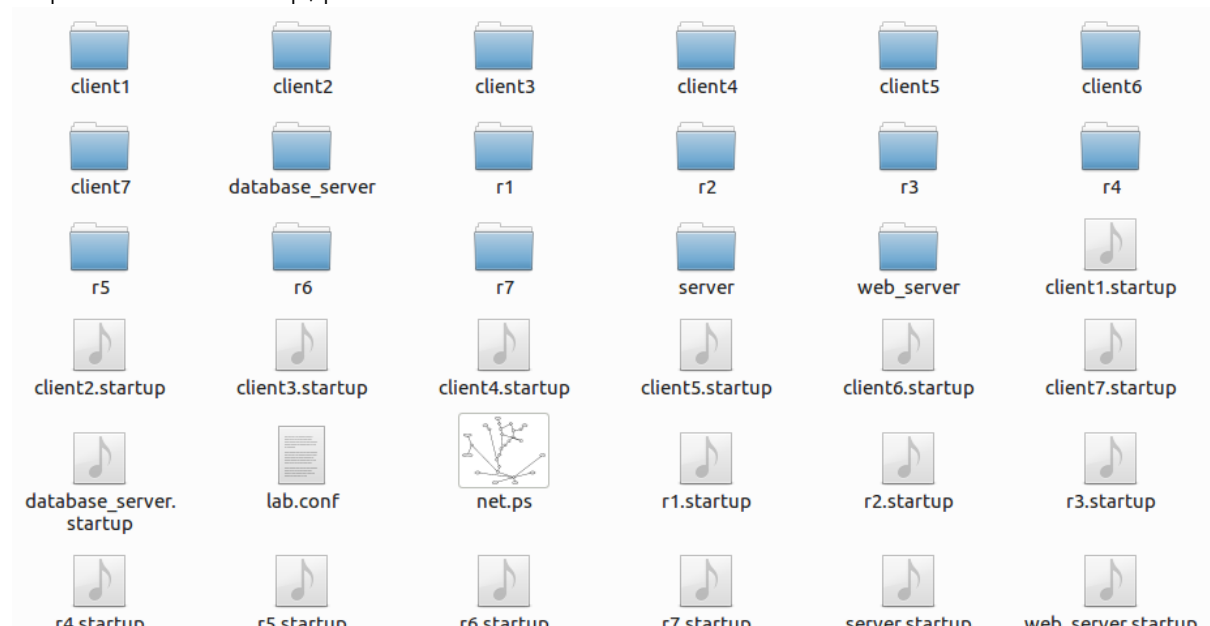
c. If I was to implement three subnets, I would add another subnet to r7. Then I would appropriately give it an IP address 100.4.74.0/29 following the naming convention in network area 0.0.0.0 (ignore ip address of subnet J and K in diagram as I am conflicted on what the ip address should be). I would simply edit the lab config files to extend the network by adding the third subnet, and then add the subnet to the ospf conf file for r7.

d. In order to support 100 laptops we would need to change the netmask to 25 or less. With a netmask of 25 we can have up to 126 hosts allowing the network to support 100 laptops. If the network was required to support 300 laptops then a netmask of 23 would be appropriate, allowing the network to support up to 510 devices.

T-B3: Implement changes in Netkit [14 marks]

Extending the network; Added all the appropriate files. And added all the necessary commands

required for the startup, proof shown below.



The screenshot displays a file explorer window showing a directory structure for a network lab. The directory contains folders for clients (client1 to client7), routers (r1 to r7), servers (server, web_server, database_server), and various startup scripts (e.g., client1.startup, r1.startup, server.startup). There are also configuration files like lab.conf and net.ps.

Project	client4.startup	client7.startup	database_server.startup
1	ifconfig eth0 110.3.74.3 netmask 255.255.255.255	1 ifconfig eth0 110.3.74.6 netmask 255.255.255.255	1 ifconfig eth0 210.3.74.2 netmask 255.255.255.255
2	route add default gw 110.3.74.1 dev eth0	2 route add default gw 110.3.74.1 dev eth0	2 route add default gw 210.3.74.3 dev eth0
3			3
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			

```
ospfd.conf
!
hostname ospfd
password zebra
enable password zebra

router ospf

! Speak OSPF on all interfaces falling in the
network 100.1.74.0/30 area 1.1.1.1
network 210.3.74.0/29 area 1.1.1.1
network 110.3.74.0/26 area 1.1.1.1
area 1.1.1.1 stub
redistribute connected
!
log file /var/log/zebra/ospfd.log
!

!
hostname ospfd
password zebra
enable password zebra

router ospf

! Speak OSPF on all interfaces falling in the
network 210.2.74.0/30 area 1.1.1.1
network 110.2.74.0/30 area 1.1.1.1
network 100.1.74.0/30 area 1.1.1.1
area 1.1.1.1 stub
redistribute connected
!
log file /var/log/zebra/ospfd.log
!
```

The above shows a screen shot of all the files added in order to extend the network. The left ospfd.conf file is for r7 and the right ospfd.conf file is for r6. Changes was required in order to extend the network in the same network area. I began by drawing the diagram for the problem described in Task B. This visual presentation made it much easier for me to refer to when extending the network, quite similar to how task A was tackled. I duplicated the task A lab and worked over the duplication just in case any problem occurs allowing me. If any errors occurred then I can go back and work from task A again. Task B is found in the B folder. More thought process described in the testing section below.

T-B4/T-B3 Continued: Testing [2 marks]

In order to minimize error (by adding it one by one it is much easier to pin point the error), I decided to add the router first and ping between router 6 to router 7 then client 1 to router 7 the results are shown below.

```
r6
Lab directory (host): /scratch/C
Version: <none>
Author: <none>
Email: <none>
Web: <none>
Description:
<none>

*****
--- Netkit phase 2 initialization terminated ---

r6 login: root (automatic login)
r6:~# ping 100.1.74.2
PING 100.1.74.2 (100.1.74.2) 56(84) bytes of data:
64 bytes from 100.1.74.2: icmp_seq=1 ttl=64 time=9.92 ms
64 bytes from 100.1.74.2: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 100.1.74.2: icmp_seq=3 ttl=64 time=0.443 ms
64 bytes from 100.1.74.2: icmp_seq=4 ttl=64 time=0.781 ms
64 bytes from 100.1.74.2: icmp_seq=5 ttl=64 time=0.603 ms
64 bytes from 100.1.74.2: icmp_seq=6 ttl=64 time=0.606 ms

r7
length 64
16:17:28.363196 IP 100.1.74.2 > 100.1.74.1: ICMP echo reply, id 11010, seq 2, length 64
16:17:29.345597 IP 100.1.74.1 > 224.0.0.5: OSPFv2, Hello, length: 48
16:17:29.362026 IP 100.1.74.1 > 100.1.74.2: ICMP echo request, id 11010, seq 3, length 64
16:17:29.362075 IP 100.1.74.2 > 100.1.74.1: ICMP echo reply, id 11010, seq 3, length 64
16:17:29.379349 IP 100.1.74.2 > 224.0.0.5: OSPFv2, Hello, length: 48
16:17:30.363481 IP 100.1.74.1 > 100.1.74.2: ICMP echo request, id 11010, seq 4, length 64
16:17:30.363530 IP 100.1.74.2 > 100.1.74.1: ICMP echo reply, id 11010, seq 4, length 64
16:17:31.372966 IP 100.1.74.1 > 100.1.74.2: ICMP echo request, id 11010, seq 5, length 64
16:17:31.373014 IP 100.1.74.2 > 100.1.74.1: ICMP echo reply, id 11010, seq 5, length 64
16:17:32.348074 arp who-has 100.1.74.1 tell 100.1.74.2
16:17:32.348588 arp reply 100.1.74.1 is-at f2:date7:51:1b:05 (oui Unknown)
16:17:32.372870 IP 100.1.74.1 > 100.1.74.2: ICMP echo request, id 11010, seq 6, length 64
16:17:32.372925 IP 100.1.74.2 > 100.1.74.1: ICMP echo reply, id 11010, seq 6, length 64
```

client1	r7
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time	16:21:50.902523 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 12, length 64
client1:~#	16:21:51.090389 IP 100.1.74.2 > 224.0.0.5: OSPFv2, LS-Ack, length: 44
client1:~#	16:21:51.911940 IP 151.4.74.2 > 100.1.74.2: ICMP echo request, id 1538, seq 13, length 64
client1:~# ping 100.1.74.2	16:21:51.911989 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 13, length 64
PING 100.1.74.2 (100.1.74.2) 56(84) bytes of data.	16:21:52.922001 IP 151.4.74.2 > 100.1.74.2: ICMP echo request, id 1538, seq 14, length 64
64 bytes from 100.1.74.2: icmp_seq=1 ttl=60 time=31.3 ms	16:21:52.922060 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 14, length 64
64 bytes from 100.1.74.2: icmp_seq=2 ttl=60 time=2.74 ms	16:21:53.931819 IP 151.4.74.2 > 100.1.74.2: ICMP echo request, id 1538, seq 15, length 64
64 bytes from 100.1.74.2: icmp_seq=3 ttl=60 time=1.09 ms	16:21:53.931832 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 15, length 64
64 bytes from 100.1.74.2: icmp_seq=4 ttl=60 time=2.46 ms	16:21:54.942038 IP 151.4.74.2 > 100.1.74.2: ICMP echo request, id 1538, seq 16, length 64
64 bytes from 100.1.74.2: icmp_seq=5 ttl=60 time=2.37 ms	16:21:54.942078 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 16, length 64
64 bytes from 100.1.74.2: icmp_seq=6 ttl=60 time=2.24 ms	16:21:55.951840 IP 151.4.74.2 > 100.1.74.2: ICMP echo request, id 1538, seq 17, length 64
64 bytes from 100.1.74.2: icmp_seq=7 ttl=60 time=2.72 ms	16:21:55.951874 IP 100.1.74.2 > 151.4.74.2: ICMP echo reply, id 1538, seq 17, length 64
64 bytes from 100.1.74.2: icmp_seq=8 ttl=60 time=2.27 ms	
64 bytes from 100.1.74.2: icmp_seq=9 ttl=60 time=1.91 ms	
64 bytes from 100.1.74.2: icmp_seq=10 ttl=60 time=2.06 ms	
64 bytes from 100.1.74.2: icmp_seq=11 ttl=60 time=2.02 ms	
64 bytes from 100.1.74.2: icmp_seq=12 ttl=60 time=1.91 ms	
64 bytes from 100.1.74.2: icmp_seq=13 ttl=60 time=1.58 ms	
64 bytes from 100.1.74.2: icmp_seq=14 ttl=60 time=1.58 ms	
64 bytes from 100.1.74.2: icmp_seq=15 ttl=60 time=1.00 ms	
64 bytes from 100.1.74.2: icmp_seq=16 ttl=60 time=1.79 ms	
64 bytes from 100.1.74.2: icmp_seq=17 ttl=60 time=1.12 ms	

After that was working, I decided to add both servers to a collision domain with subnet mask 29.29 to allow 3 hosts since a subnet mask of 29 allows a max of 6 hosts. I edited the following files: lab.conf, r7.startup, database_server.startup, web_server.startup and added the server folders to accommodate the network extension. After that I had to test the network by pinging again just like in Task A between 2 nodes within the same network area and 2 nodes in different network area. In this case I tested between router 6 to database_server and from client 1 to database_server (the same was also applied to web_server). The results are shown below.

r6	database_server
64 bytes from 100.1.74.2: icmp_seq=123 ttl=64 time=0.405 ms	16:37:51.712084 IP 100.1.74.1 > 210.3.74.2: ICMP echo request, id 11266, seq 4, length 64
64 bytes from 100.1.74.2: icmp_seq=124 ttl=64 time=0.373 ms	16:37:51.712123 IP 210.3.74.2 > 100.1.74.1: ICMP echo reply, id 11266, seq 4, length 64
64 bytes from 100.1.74.2: icmp_seq=125 ttl=64 time=0.362 ms	16:37:52.722228 IP 100.1.74.1 > 210.3.74.2: ICMP echo request, id 11266, seq 5, length 64
64 bytes from 100.1.74.2: icmp_seq=126 ttl=64 time=0.397 ms	16:37:52.722271 IP 210.3.74.2 > 100.1.74.1: ICMP echo reply, id 11266, seq 5, length 64
64 bytes from 100.1.74.2: icmp_seq=127 ttl=64 time=0.420 ms	16:37:53.691729 arp who-has 210.3.74.3 tell 210.3.74.2
64 bytes from 100.1.74.2: icmp_seq=128 ttl=64 time=0.096 ms	16:37:53.692099 arp reply 210.3.74.3 is-at be:00:48:aa:07:b6 (oui Unknown)
--- 100.1.74.2 ping statistics ---	16:37:53.732164 IP 100.1.74.1 > 210.3.74.2: ICMP echo request, id 11266, seq 6, length 64
128 packets transmitted, 128 received, 0% packet loss, time 127475ms	16:37:53.732204 IP 210.3.74.2 > 100.1.74.1: ICMP echo reply, id 11266, seq 6, length 64
rtt min/avg/max/mdev = 0.090/0.560/9.924/0.845 ms	16:37:54.742414 IP 100.1.74.1 > 210.3.74.2: ICMP echo request, id 11266, seq 7, length 64
r6:~#	16:37:54.742445 IP 210.3.74.2 > 100.1.74.1: ICMP echo reply, id 11266, seq 7, length 64
r6:~#	16:37:55.752233 IP 100.1.74.1 > 210.3.74.2: ICMP echo request, id 11266, seq 8, length 64
r6:~# ping 210.3.74.2	16:37:55.752279 IP 210.3.74.2 > 100.1.74.1: ICMP echo reply, id 11266, seq 8, length 64
PING 210.3.74.2 (210.3.74.2) 56(84) bytes of data.	
64 bytes from 210.3.74.2: icmp_seq=1 ttl=63 time=11.1 ms	
64 bytes from 210.3.74.2: icmp_seq=2 ttl=63 time=0.931 ms	
64 bytes from 210.3.74.2: icmp_seq=3 ttl=63 time=0.372 ms	
64 bytes from 210.3.74.2: icmp_seq=4 ttl=63 time=0.899 ms	
64 bytes from 210.3.74.2: icmp_seq=5 ttl=63 time=0.957 ms	
64 bytes from 210.3.74.2: icmp_seq=6 ttl=63 time=0.885 ms	
64 bytes from 210.3.74.2: icmp_seq=7 ttl=63 time=0.898 ms	
64 bytes from 210.3.74.2: icmp_seq=8 ttl=63 time=0.870 ms	

client1	database_server
client1 login: root (automatic login)	10:45:46.102577 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 13, length 64
client1:~# ping 210.3.74.2	10:45:46.453569 IP 210.3.74.3 > 224.0.0.5: OSPFv2, Hello, length: 44
PING 210.3.74.2 (210.3.74.2) 56(84) bytes of data.	10:45:47.112322 IP 151.4.74.2 > 210.3.74.2: ICMP echo request, id 1282, seq 14, length 64
64 bytes from 210.3.74.2: icmp_seq=1 ttl=59 time=42.1 ms	10:45:47.112336 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 14, length 64
64 bytes from 210.3.74.2: icmp_seq=2 ttl=59 time=0.588 ms	10:45:48.112487 IP 151.4.74.2 > 210.3.74.2: ICMP echo request, id 1282, seq 15, length 64
64 bytes from 210.3.74.2: icmp_seq=3 ttl=59 time=0.966 ms	10:45:48.112500 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 15, length 64
64 bytes from 210.3.74.2: icmp_seq=4 ttl=59 time=0.754 ms	10:45:49.122815 IP 151.4.74.2 > 210.3.74.2: ICMP echo request, id 1282, seq 16, length 64
64 bytes from 210.3.74.2: icmp_seq=5 ttl=59 time=0.623 ms	10:45:49.122831 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 16, length 64
64 bytes from 210.3.74.2: icmp_seq=6 ttl=59 time=0.967 ms	10:45:50.132396 IP 151.4.74.2 > 210.3.74.2: ICMP echo request, id 1282, seq 17, length 64
64 bytes from 210.3.74.2: icmp_seq=7 ttl=59 time=0.633 ms	10:45:50.132410 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 17, length 64
64 bytes from 210.3.74.2: icmp_seq=8 ttl=59 time=0.794 ms	10:45:51.132451 IP 151.4.74.2 > 210.3.74.2: ICMP echo request, id 1282, seq 18, length 64
64 bytes from 210.3.74.2: icmp_seq=9 ttl=59 time=0.982 ms	10:45:51.132470 IP 210.3.74.2 > 151.4.74.2: ICMP echo reply, id 1282, seq 18, length 64
64 bytes from 210.3.74.2: icmp_seq=10 ttl=59 time=0.741 ms	
64 bytes from 210.3.74.2: icmp_seq=11 ttl=59 time=0.512 ms	
64 bytes from 210.3.74.2: icmp_seq=12 ttl=59 time=1.18 ms	
64 bytes from 210.3.74.2: icmp_seq=13 ttl=59 time=1.32 ms	
64 bytes from 210.3.74.2: icmp_seq=14 ttl=59 time=0.539 ms	
64 bytes from 210.3.74.2: icmp_seq=15 ttl=59 time=0.948 ms	
64 bytes from 210.3.74.2: icmp_seq=16 ttl=59 time=1.08 ms	
64 bytes from 210.3.74.2: icmp_seq=17 ttl=59 time=0.647 ms	
64 bytes from 210.3.74.2: icmp_seq=18 ttl=59 time=0.843 ms	

Again, checking that I was able to ping between the nodes I began adding the clients onto the network. After adding the clients, I double checked to ensure I was able to ping from the same & different network area. I ping from r7 to client 6, then client 1 to client 6. I done this for all

clients to ensure validity in the implementation.

```

x - + r7
*****
--- Netkit phase 2 initialization terminated ---

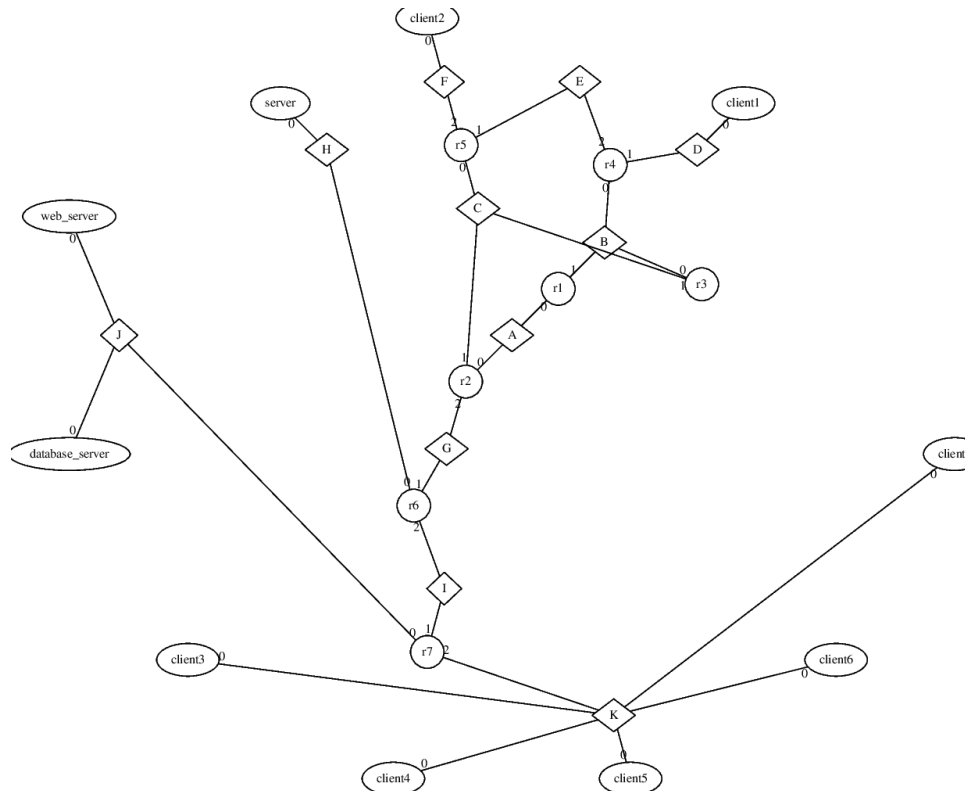
r7 login: root (automatic login)
r7:~# ping 110.3.74.5
PING 110.3.74.5 (110.3.74.5) 56(84) bytes of data.
64 bytes from 110.3.74.5: icmp_seq=1 ttl=64 time=0.355 ms
64 bytes from 110.3.74.5: icmp_seq=2 ttl=64 time=0.183 ms
64 bytes from 110.3.74.5: icmp_seq=3 ttl=64 time=0.257 ms
64 bytes from 110.3.74.5: icmp_seq=4 ttl=64 time=0.163 ms
64 bytes from 110.3.74.5: icmp_seq=5 ttl=64 time=0.183 ms
64 bytes from 110.3.74.5: icmp_seq=6 ttl=64 time=0.145 ms
64 bytes from 110.3.74.5: icmp_seq=7 ttl=64 time=0.169 ms
64 bytes from 110.3.74.5: icmp_seq=8 ttl=64 time=0.140 ms
64 bytes from 110.3.74.5: icmp_seq=9 ttl=64 time=0.170 ms
64 bytes from 110.3.74.5: icmp_seq=10 ttl=64 time=0.182 ms
64 bytes from 110.3.74.5: icmp_seq=11 ttl=64 time=0.237 ms
64 bytes from 110.3.74.5: icmp_seq=12 ttl=64 time=0.241 ms
64 bytes from 110.3.74.5: icmp_seq=13 ttl=64 time=0.165 ms
64 bytes from 110.3.74.5: icmp_seq=14 ttl=64 time=0.184 ms
64 bytes from 110.3.74.5: icmp_seq=15 ttl=64 time=0.130 ms
^C
^C

x - + client6
length 64
10:51:47.302926 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 10,
length 64
10:51:48.302969 IP 110.3.74.1 > 110.3.74.5: ICMP echo request, id 8962, seq 11,
length 64
10:51:48.302987 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 11,
length 64
10:51:49.303015 IP 110.3.74.1 > 110.3.74.5: ICMP echo request, id 8962, seq 12,
length 64
10:51:49.303035 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 12,
length 64
10:51:50.302861 IP 110.3.74.1 > 110.3.74.5: ICMP echo request, id 8962, seq 13,
length 64
10:51:50.302889 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 13,
length 64
10:51:51.302913 IP 110.3.74.1 > 110.3.74.5: ICMP echo request, id 8962, seq 14,
length 64
10:51:51.302934 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 14,
length 64
10:51:52.302912 IP 110.3.74.1 > 110.3.74.5: ICMP echo request, id 8962, seq 15,
length 64
10:51:52.302928 IP 110.3.74.5 > 110.3.74.1: ICMP echo reply, id 8962, seq 15,
length 64

x - + client1
client1:~# ping 110.3.74.5
PING 110.3.74.5 (110.3.74.5) 56(84) bytes of data.
64 bytes from 110.3.74.5: icmp_seq=1 ttl=59 time=0.579 ms
64 bytes from 110.3.74.5: icmp_seq=2 ttl=59 time=0.468 ms
64 bytes from 110.3.74.5: icmp_seq=3 ttl=59 time=0.637 ms
64 bytes from 110.3.74.5: icmp_seq=4 ttl=59 time=0.963 ms
64 bytes from 110.3.74.5: icmp_seq=5 ttl=59 time=0.698 ms
64 bytes from 110.3.74.5: icmp_seq=6 ttl=59 time=0.668 ms
64 bytes from 110.3.74.5: icmp_seq=7 ttl=59 time=0.802 ms
64 bytes from 110.3.74.5: icmp_seq=8 ttl=59 time=0.768 ms
64 bytes from 110.3.74.5: icmp_seq=9 ttl=59 time=0.736 ms
^C
^C

x - + client6
gth 64
10:56:50.952422 IP 151.4.74.2 > 110.3.74.5: ICMP echo request, id 2050, seq 5,
length 64
10:56:50.952434 IP 110.3.74.5 > 151.4.74.2: ICMP echo reply, id 2050, seq 5, l
gth 64
10:56:51.908082 arp who-has 110.3.74.1 tell 110.3.74.5
10:56:51.908272 arp reply 110.3.74.1 is-at 92:37:e3:de:45:a9 (oui Unknown)
10:56:51.952389 IP 151.4.74.2 > 110.3.74.5: ICMP echo request, id 2050, seq 6,
length 64
10:56:51.952402 IP 110.3.74.5 > 151.4.74.2: ICMP echo reply, id 2050, seq 6, l
gth 64
10:56:52.952609 IP 151.4.74.2 > 110.3.74.5: ICMP echo request, id 2050, seq 7,
length 64
10:56:52.952623 IP 110.3.74.5 > 151.4.74.2: ICMP echo reply, id 2050, seq 7, l
gth 64
10:56:53.952607 IP 151.4.74.2 > 110.3.74.5: ICMP echo request, id 2050, seq 8,
length 64
10:56:53.952621 IP 110.3.74.5 > 151.4.74.2: ICMP echo reply, id 2050, seq 8, l
gth 64
10:56:54.952463 IP 151.4.74.2 > 110.3.74.5: ICMP echo request, id 2050, seq 9,
length 64
10:56:54.952476 IP 110.3.74.5 > 151.4.74.2: ICMP echo reply, id 2050, seq 9, l
gth 64
```

Generated another image of the graph using the linfo command. If the image was incomplete, ie if there were lines missing between nodes then we would know that there is an error with the files. Using this method, we are actually able to pinpoint the cause of the error since if a node was isolated (no lines to or from the node) then we know that there is an error associated with the file, or perhaps an error in the lab.conf file.



Further Testing using vtysh. Using the command `show ip route` (or `show ip route ospf`) and `show ip rip` will give us the list of routes to network destinations. If each router should display 11 routes since we have 11 networks in our topology. The image below shows the list of paths for r2. We can see that if we filter through the routes, we find that there are 11 unique networks thus matching with our topology. We can also identify the directly connected nodes and compare it with our network topology diagram and see if there are any missing nodes.

```
x - + r5
0>* 151.5.74.0/24 [110/20] via 151.3.74.1, eth1, 00:22:28
0>* 210.2.74.0/30 [110/50] via 151.3.74.1, eth1, 00:22:21
0>* 210.3.74.0/29 [110/60] via 151.3.74.1, eth1, 00:22:16
r5#
r5#
r5# show ip route ospf
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

0>* 100.1.74.0/30 [110/50] via 151.3.74.1, eth1, 00:22:24
0>* 110.2.74.0/30 [110/40] via 151.3.74.1, eth1, 00:22:31
0>* 110.3.74.0/26 [110/60] via 151.3.74.1, eth1, 00:22:19
0>* 151.0.74.0/24 [110/30] via 151.3.74.1, eth1, 00:22:31
0 151.1.74.0/24 [110/40] via 151.3.74.1, eth1, 00:22:31
0 151.2.74.0/24 [110/10] is directly connected, eth2, 00:23:16
0 151.3.74.0/24 [110/10] is directly connected, eth1, 00:23:16
0>* 151.4.74.0/24 [110/20] via 151.3.74.1, eth1, 00:22:31
0>* 151.5.74.0/24 [110/20] via 151.3.74.1, eth1, 00:22:31
0>* 210.2.74.0/30 [110/50] via 151.3.74.1, eth1, 00:22:24
0>* 210.3.74.0/29 [110/60] via 151.3.74.1, eth1, 00:22:19
r5#
```

Task C: Routing experiments [8 marks]

T-C1: OSPF vs RIP[4 marks]

Examine and compare the routing algorithms and see if they have calculated the shortest paths correctly and how they do that. Screenshots below shows results when tracerouting between two nodes using OSPF and RIP.

RIP: Client 1 to Client 2

```
x - + client1
client1:~# traceroute 151.2.74.2
traceroute to 151.2.74.2 (151.2.74.2), 64 hops max, 40 byte packets
 1 151.4.74.1 (151.4.74.1)  0 ms  0 ms  0 ms
 2 151.3.74.2 (151.3.74.2)  0 ms  0 ms  0 ms
 3 151.2.74.2 (151.2.74.2)  0 ms  0 ms  0 ms
client1:~#
```

This test confirms that RIP considers the smallest hop count (3) to client 2. The routing information protocol only considers hop count to calculate the best path of a network and does not consider the cost unlike OSPF. RIP works by transmitting updates every 30 seconds. These transmissions hold the current router's information containing a list of routes to network destinations and are transmitted to adjacent routers. In the event that two paths have the same number of hops the router will send the packet files through both routes at the same time or by any other forwarding means. The forwarding mechanism used is default Cisco Forwarding Mechanism (CFM). Upon further research

(https://www.cisco.com/en/US/products/hw/modules/ps2033/prod_technical_reference09186a00800a_feb7.html) I found that a good router would distribute its traffic across multiple paths making 'efficient use of the available bandwidth'.

Here I am using the commands 'show ip rip' and 'show ip route' which outputs the router's information containing a list of routes to network destinations. We are looking at route 4 since it is client 1's route gateway. From the list of routes, we identify the path with the smallest number of hops needed to get to client 2. In this case we find that in order to get to 151.2.74.0/24 we need to hop to 151.3.74.0/24 and then hop to the destination, that is a total of 2 hops.

```
x - + r4
Password:
Password:
ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
(n) - normal, (s) - static, (d) - default, (r) - redistribute,
(i) - interface

Network        Next Hop        Metric From      Tag Time
R(n) 100.1.74.0/30 151.5.74.1      4 151.5.74.1      0 02:41
R(n) 110.2.74.0/30 151.5.74.1      3 151.5.74.1      0 02:41
R(n) 110.3.74.0/26 151.5.74.3      5 151.5.74.3      0 02:51
R(n) 151.0.74.0/24 151.5.74.1      2 151.5.74.1      0 02:41
R(n) 151.1.74.0/24 151.5.74.3      2 151.5.74.3      0 02:51
R(n) 151.2.74.0/24 151.3.74.2      2 151.3.74.2      0 02:43
C(i) 151.3.74.0/24 0.0.0.0         1 self           0
C(i) 151.4.74.0/24 0.0.0.0         1 self           0
C(i) 151.5.74.0/24 0.0.0.0         1 self           0
R(n) 210.2.74.0/30 151.5.74.1      4 151.5.74.1      0 02:41
R(n) 210.3.74.0/29 151.5.74.3      5 151.5.74.3      0 02:51
ripd>
Vty connection is timed out.
Connection closed by foreign host.
r4:~#

x - + r4
r4:~# vtysh
-bash: vtysh: command not found
r4:~# vtysh

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

r4# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 100.1.74.0/30 [120/4] via 151.5.74.1, eth0, 01:23:53
R>* 110.2.74.0/30 [120/3] via 151.5.74.1, eth0, 01:24:06
R>* 110.3.74.0/26 [120/5] via 151.5.74.3, eth0, 01:23:48
C>* 127.0.0.0/8 is directly connected, lo
R>* 151.0.74.0/24 [120/2] via 151.5.74.1, eth0, 01:24:06
R>* 151.1.74.0/24 [120/2] via 151.5.74.3, eth0, 01:24:06
R>* 151.2.74.0/24 [120/2] via 151.3.74.2, eth2, 01:24:00
C>* 151.3.74.0/24 is directly connected, eth2
C>* 151.4.74.0/24 is directly connected, eth1
C>* 151.5.74.0/24 is directly connected, eth0
R>* 210.2.74.0/30 [120/4] via 151.5.74.1, eth0, 01:23:53
R>* 210.3.74.0/29 [120/5] via 151.5.74.3, eth0, 01:23:48
r4#
```


OSPF: Client 1 to Client 2

```
x - + client1
client1:~# traceroute 151.2.74.2
traceroute to 151.2.74.2 (151.2.74.2), 64 hops max, 40 byte packets
 1 151.4.74.1 (151.4.74.1) 0 ms 0 ms 0 ms
 2 151.1.74.3 (151.1.74.3) 0 ms 0 ms 0 ms
 3 151.3.74.2 (151.3.74.2) 0 ms 0 ms 0 ms
 4 151.2.74.2 (151.2.74.2) 0 ms 0 ms 0 ms
client1:~#
```

In the test above we discovered that OSPF takes a different route to RIP. This is because a differential factor between the two is that OSPF takes in parameters such as speed, cost and traffic and would calculate the shortest path using Dijkstra algorithm. Upon research, (open shortest path first) uses a link state routing protocol which can transmit routing information to all other routes running the same protocol.

The algorithm works as follows: [<https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html>] setting the routing information or changing the information will generate a link-state advertisement which will be used to update the routing information across the other routers (in other words the link-state database is synced across all routers in the network). When a router receives the link-state advertisement it should store a copy into its 'link-state database' and distribute the update to other routers. After the LSDB (linked-state database) is synced across the routers, the router will calculate a list of shortest paths to all destinations, using Dijkstra's shortest path algorithm. The router now contains a IP routing table containing a list of destination IP with its associated cost and the next hop to reach the destination. Lastly if there is an update to the cost or change to the topology of the network link-state packets will be passed around communicating the change that occurred. The routers will then perform Dijkstra algorithm again in order to find the shortest path.

In the image below, we see the list of possible routes from r4. We need to get to 151.2.74.2 (client 2), from the table we see that we can get there through 151.5.74.3. When we get to r3, the routing table is checked to see how to get to the destination. From the list we see that we can get to client 2 through 151.1.74.2 (R5). At r5 again looking at the routing table we see that to get to client 2 we see that the network is directly connected to r5, and thus we have reached our destination. A similar command that I could have used is show ip ospf route or show ip route ospf. The routing table for each router has been generated using the process explained above.

```
x - + r4
r4# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

O* 100.1.74.0/30 [110/40] via 151.5.74.1, eth0, 00:40:56
O* 110.2.74.0/30 [110/30] via 151.5.74.1, eth0, 00:41:09
O* 110.3.74.0/26 [110/50] via 151.5.74.1, eth0, 00:40:51
C* 127.0.0.0/8 is directly connected, lo
O* 151.0.74.0/24 [110/20] via 151.5.74.1, eth0, 00:41:09
O* 151.1.74.0/24 [110/30] via 151.5.74.3, eth0, 00:41:09
O* 151.2.74.0/24 [110/40] via 151.5.74.3, eth0, 00:41:03
O 151.3.74.0/24 [110/40] via 151.5.74.3, eth0, 00:41:03
C* 151.3.74.0/24 is directly connected, eth2
O 151.4.74.0/24 [110/10] is directly connected, eth1, 00:41:54
C* 151.4.74.0/24 is directly connected, eth1
O 151.5.74.0/24 [110/10] is directly connected, eth0, 00:41:54
C* 151.5.74.0/24 is directly connected, eth0
O* 210.2.74.0/30 [110/40] via 151.5.74.1, eth0, 00:40:56
O* 210.3.74.0/29 [110/50] via 151.5.74.1, eth0, 00:40:51

x - + r3
r3:~# vtysh
Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

r3# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

O* 100.1.74.0/30 [110/40] via 151.1.74.1, eth1, 00:00:56
O* 110.2.74.0/30 [110/30] via 151.1.74.1, eth1, 00:01:03
O* 110.3.74.0/26 [110/50] via 151.1.74.1, eth1, 00:00:51
C* 127.0.0.0/8 is directly connected, lo
O* 151.0.74.0/24 [110/30] via 151.1.74.1, eth1, 00:01:03
O 151.1.74.0/24 [110/20] is directly connected, eth1, 00:01:03
C* 151.1.74.0/24 is directly connected, eth1
O* 151.2.74.0/24 [110/30] via 151.1.74.2, eth1, 00:01:03
O* 151.3.74.0/24 [110/30] via 151.1.74.2, eth1, 00:01:03
O* 151.4.74.0/24 [110/40] via 151.1.74.2, eth1, 00:01:03
O 151.5.74.0/24 [110/40] via 151.1.74.2, eth1, 00:01:03
C* 151.5.74.0/24 is directly connected, eth0
O* 210.2.74.0/30 [110/40] via 151.1.74.1, eth1, 00:00:56
O* 210.3.74.0/29 [110/50] via 151.1.74.1, eth1, 00:00:51
```


x - + r5	x - + r5
Copyright 1996-2005 Kunihiro Ishiguro, et al.	Copyright 1996-2005 Kunihiro Ishiguro, et al.
r5# show ip route	r5# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF, I - ISIS, B - BGP, > - selected route, * - FIB route	Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF, I - ISIS, B - BGP, > - selected route, * - FIB route
O>* 110.2.74.0/30 [110/40] via 151.3.74.1, eth1, 00:00:06	O>* 110.2.74.0/30 [110/40] via 151.3.74.1, eth1, 00:00:06
C>* 127.0.0.0/8 is directly connected, lo	C>* 127.0.0.0/8 is directly connected, lo
O>* 151.0.74.0/24 [110/30] via 151.3.74.1, eth1, 00:00:06	O>* 151.0.74.0/24 [110/30] via 151.3.74.1, eth1, 00:00:06
O 151.1.74.0/24 [110/40] via 151.3.74.1, eth1, 00:00:06	O 151.1.74.0/24 [110/40] via 151.3.74.1, eth1, 00:00:06
C>* 151.1.74.0/24 is directly connected, eth0	C>* 151.1.74.0/24 is directly connected, eth0
O 151.2.74.0/24 [110/10] is directly connected, eth2, 00:00:51	O 151.2.74.0/24 [110/10] is directly connected, eth2, 00:00:51
C>* 151.2.74.0/24 is directly connected, eth2	C>* 151.2.74.0/24 is directly connected, eth2
O 151.3.74.0/24 [110/10] is directly connected, eth1, 00:00:51	O 151.3.74.0/24 [110/10] is directly connected, eth1, 00:00:51
C>* 151.3.74.0/24 is directly connected, eth1	C>* 151.3.74.0/24 is directly connected, eth1
O>* 151.4.74.0/24 [110/20] via 151.3.74.1, eth1, 00:00:06	O>* 151.4.74.0/24 [110/20] via 151.3.74.1, eth1, 00:00:06
O>* 151.5.74.0/24 [110/20] via 151.3.74.1, eth1, 00:00:06	O>* 151.5.74.0/24 [110/20] via 151.3.74.1, eth1, 00:00:06

Using the

command "show ip route ospf" we see the list of routes we can take to get to a network along with its associated minimum cost. I have double checked the results and have confirmed that the algorithm did indeed pick the shortest route.

Route Process:

Looking at r4: Using 'show ip route ospf', I identified that to get to 151.2.74.2 (client 2), it will cost 40 and requires going to r3

```
N 151.2.74.0/24 [40] area: 0.0.0.0
via 151.5.74.3, eth0
```

To get from r4 to r3 the cost is 10, we can see this from the routing information in r4.

```
N 151.5.74.0/24 [10] area: 0.0.0.0
directly attached to eth0
```

Looking at r3: To get to 151.2.74.2 we need to get to 151.1.74.2 (r5) which costs 20.

```
N 151.1.74.0/24 [20] area: 0.0.0.0
directly attached to eth1
```

Looking at r5: To get to 151.2.74.2 we need to get to 151.1.74.0/24 which costs 10.

```
N 151.2.74.0/24 [10] area: 0.0.0.0
directly attached to eth2
```

Adding the total, we get $10 + 20 + 10 = 40$.

We can confirm this as the only possible non-cyclical routes from Client 1 to client 2 are;

client1 – r4 – r5 – client2 [Total route = $50 + 10 = 60$]

client1 – r4 – r3 – r5 – client 2 [Total route = $10 + 20 + 10 = 40$] <- shortest route

client1 – r4 – r1 – r2 – r5 – client 2 [Total route = $10 + 10 + 50 + 10 = 80$]

Through research and experimentation, it was quite clear that rip did not consider the cost, although this was obvious since the cost was not specified in the files unlike OSPF. The main difference between the two is that RIP only considers hop as a distance metric whilst OSPF considers parameters such as speed, bandwidth and cost. Upon research I discovered that RIP is used for small networks whilst OSPF is used for large hierarchical networks since RIP only considers the first 15 hops before assuming the distance between the two nodes are infinity, therefore RIP is suited for small, simple networks. Because of this I'd expect networks that use RIP to be cheaper to set up unlike OSPF which is much more complex and CPU intensive.

T-C2: The preferred route in the network [4 marks]

RIP: Client 2 to Server

```
x - + client2
client2:~# traceroute 210.2.74.1
traceroute to 210.2.74.1 (210.2.74.1), 64 hops max, 40 byte packets
 1  151.2.74.1 (151.2.74.1)  0 ms  0 ms  0 ms
 2  151.1.74.1 (151.1.74.1)  0 ms  0 ms  0 ms
 3  110.2.74.1 (110.2.74.1)  0 ms  0 ms  0 ms
 4  210.2.74.1 (210.2.74.1)  0 ms  0 ms  0 ms
client2:~#
```

```
x - + r5
Network          Next Hop          Metric From          Tag Time
R(n) 100.1.74.0/30 151.1.74.1         3 151.1.74.1         0 02:45
R(n) 110.2.74.0/30 151.1.74.1         2 151.1.74.1         0 02:45
R(n) 110.3.74.0/26 151.1.74.1         4 151.1.74.1         0 02:45
R(n) 151.0.74.0/24 151.1.74.1         2 151.1.74.1         0 02:45
C(i) 151.1.74.0/24 0.0.0.0            1 self              0
C(i) 151.2.74.0/24 0.0.0.0            1 self              0
C(i) 151.3.74.0/24 0.0.0.0            1 self              0
R(n) 151.4.74.0/24 151.3.74.1         2 151.3.74.1         0 02:36
R(n) 151.5.74.0/24 151.1.74.3         2 151.1.74.3         0 02:46
R(n) 210.2.74.0/30 151.1.74.1         3 151.1.74.1         0 02:45 <- 'show ip rip'
```

For Routing information protocol, when we traceroute between client 2 and server the packets take the route from client 2 to r5 to r2 to r6 to server. This is because rip takes the route with the minimal number of hops. Rip does not concern itself with the route's cost, therefore the optimal route definition for RIP may not give us the most optimal route if cost, traffic etc. were considered. The calculated cost of this route is $74+10+10 = 94$.

Route A: Client 2 – r5 – r2 – r6 – Server <- least number of hops

Route B: Client 2 – r5 – r4 – r3 – r2 – r6 – Server

Route C: Client 2 – r5 – r4 – r1 – r2 – r6 – Server

From the above we can see that route A is the least number of hops. This information is calculated when the lab has been set up. It is only calculated once during the initialization of the router and is updated periodically.

OSPF: Client 2 to Server

```
x - + client2
client2:~# traceroute 210.2.74.1
traceroute to 210.2.74.1 (210.2.74.1), 64 hops max, 40 byte packets
 1  151.2.74.1 (151.2.74.1)  3 ms  0 ms  0 ms
 2  151.5.74.2 (151.5.74.2) 31 ms  0 ms  0 ms
 3  151.5.74.1 (151.5.74.1) 19 ms  0 ms  0 ms
 4  151.0.74.2 (151.0.74.2)  5 ms  0 ms  0 ms
 5  110.2.74.1 (110.2.74.1)  0 ms  0 ms  0 ms
 6  210.2.74.1 (210.2.74.1)  0 ms  0 ms  0 ms
client2:~#
```

```
x - + r5
N IA 210.2.74.0/30    [50] area: 0.0.0.0
                     via 151.3.74.1, eth1 <- show ip ospf route
```

For Open shortest path first protocol, when we traceroute between client 2 and server the

packets take the route from client 2 to r5 to r4 to r1 to r2 to r6 to server. This is because, unlike rip, OSPF calculates the optimal route considering the cost. The route taken has 6 hops, which is greater than rip's route which had 4 hops. The important distance between the two is the cost. Rip's route has a total cost of 94 whilst OSPF has a total cost of 50.

OSPF calculated the optimal route using Dijkstra's shortest path algorithm. Here are all the non-cyclic routes and their associated costs:

Route A: Client 2 – r5 – r2 – r6 - Server [Cost=94]

Route B: Client 2 – r5 – r4 – r3 – r2 – r6 – Server [Cost=50] <- Cheapest

Route C: Client 2 – r5 – r4 – r1 – r2 – r6 – Server [Cost=60]

I explained the process of how OSPF works in the first part of the question. Again, the optimal path is calculated after the initialization of the routers and through updates of the routers using link-state advertisements.

Task D: DNS server [8 marks]

Updated Diagram:



Screenshot below shows that I have added the DNS server to the network:

```
Project
├── D
│   ├── client1
│   ├── client2
│   │   └── etc
│   │       └── resolv.conf
│   ├── client3
│   ├── client4
│   ├── client5
│   ├── client6
│   ├── client7
│   ├── database_server
│   ├── dnscom
│   │   ├── etc
│   │   ├── bind
│   │   ├── db.com
│   │   ├── db.root
│   │   └── named.conf
│   ├── dnsroot
│   │   ├── etc
│   │   ├── bind
│   │   ├── db.root
│   │   └── named.conf
│   ├── r1
│   ├── r2
│   ├── r3
│   ├── r4
│   ├── r5
│   ├── r6
│   ├── r7
│   ├── server
│   ├── web_server
│   ├── client1.startup
│   ├── client2.startup
│   ├── client3.startup
│   ├── client4.startup
│   ├── client5.startup
│   ├── client6.startup
│   ├── client7.disk
│   ├── client7.startup
│   ├── database_server.startup
│   └── dnscom.startup
└── db.root
    1 . IN NS ROOT-SERVER.
    2 ROOT-SERVER. IN A 210.3.74.5
    3
db.com
    1 $TTL 60000
    2 @ IN SOA dnscom.com. root.dnscom.com. (
    3 2006031201 ; serial
    4 28800 ; refresh
    5 14400 ; retry
    6 3600000 ; expire
    7 0 ; negative cache ttl
    8 )
    9 @ IN NS dnscom.com.
    10 dnscom IN A 210.3.74.4
    11 nk00374 IN A 210.2.74.1
    12
dnsroot.startup
    1 ifconfig eth0 210.3.74.5 netmask 255.255.255.248 broadcast 210.3.74.7 up
    2 route add default gw 210.3.74.3 dev eth0
    3 /etc/init.d/bind start
    4
dnscom.startup
    1 ifconfig eth0 210.3.74.4 netmask 255.255.255.248 broadcast 210.3.74.7 up
    2 route add default gw 210.3.74.3 dev eth0
    3 /etc/init.d/bind start
    4
resolv.conf
    1 nameserver 210.3.74.4
    2
named.conf
    1
    2 zone "." {
    3     type master;
    4     file "/etc/bind/db.root";
    5 };
    6
    7 zone "com" {
    8     type master;
    9     file "/etc/bind/db.com";
    10 };
    11
    12 zone "com" {
    13     type master;
    14     file "/etc/bind/db.com";
    15 };
    16
    17
```

T-D2: DNS functions and settings [4 marks]

In order to successfully connect to the server using the address <http://nk00374.com/~guest/> I decided that two DNS servers were required. That is the DNS Root server and the TLD (top level domain) DNS com server. The DNS (domain name system) is a hierarchical database that in response for the query <http://nk00374.com/~guest/> will query the root server which will then respond with the address of the com DNS server. The com DNS server is queried to retrieve the IP address of the domain name server of nk00374.com. The query is then sent to the domain's name server and the IP address of nk00374.com is returned. With the IP returned the client should be able to make a successful connection to the server and render the HTML. The user can render the HTML since the `/~guest/` in the URL is a request that is handled by the apache server, this is done once the nk00374.com has been resolved and has been translated to 210.2.74.1.

The resolv.conf file in our client has the line "nameserver 210.3.74.4" which specifies that the default name server is dnscom.com.

In the named.conf file we specify the location holding information about the root name server which is found in `"/etc/bind/db-root"` and we specify where to find the names in the zone which is found in `"/etc/bind/db.com"`.

In the db.com file we set the following configuration on the name server as follows: The serial number determines how recent information is and influences data within the zone and has the

format YYYYMMDDNN and is set to 2006031201. There is also a refresh interval which periodically checks if the data is up to date. We also set the number of seconds it takes between each attempt to reconnect to the master. An expire time is also set to stop the connection to the master.

```

db.com
1 $TTL 60000
2 @ IN SOA dnscom.com. root.dnscom.com. (
3 2006031201 ; serial
4 28800 ; refresh
5 14400 ; retry
6 3600000 ; expire
7 0 ; negative cache ttl
8 )
9 @ IN NS dnscom.com.
10 dnscom IN A 210.3.74.4
11 nk00374 IN A 210.2.74.1

```

Below the authoritative information, we have associations between the names and the addresses. We can see that there are two machines in this zone, that is dnscom.com and nk00374.com (the origin '.' is automatically appended).

```

db.root
1 $TTL 60000
2 @ IN SOA ROOT-SERVER. root.ROOT-SERVER. (
3 2006031201 ; serial
4 28800 ; refresh
5 14400 ; retry
6 3600000 ; expire
7 0 ; negative cache ttl
8 )
9 @ IN NS ROOT-SERVER.
10 ROOT-SERVER. IN A 210.3.74.5
11 com. IN NS dnscom.com.
12 dnscom.com. IN A 210.3.74.4

```

For the db.root file, we have the same thing however for the DNS root server we need to specify an authority for a subdomain which is dnscom.com. In line 11 it simply shows ROOT-SERVER is the authority for the zone root whilst the dnscom.com is the authority for the com zone.

Task E: Network analysis [20 marks]

T-E1: Test connectivity [4 marks]

client4 to client1: [client4 - r7 - r6 - r2 - r4 - client1] [Two nodes from the same network]

```

x - + client4
client4:~# ping 151.4.74.2
PING 151.4.74.2 (151.4.74.2) 56(84) bytes of data.
64 bytes from 151.4.74.2: icmp_seq=1 ttl=59 time=1.58 ms
64 bytes from 151.4.74.2: icmp_seq=2 ttl=59 time=2.02 ms
64 bytes from 151.4.74.2: icmp_seq=3 ttl=59 time=2.23 ms
64 bytes from 151.4.74.2: icmp_seq=4 ttl=59 time=2.45 ms
64 bytes from 151.4.74.2: icmp_seq=5 ttl=59 time=2.11 ms
64 bytes from 151.4.74.2: icmp_seq=6 ttl=59 time=2.20 ms
64 bytes from 151.4.74.2: icmp_seq=7 ttl=59 time=3.24 ms
64 bytes from 151.4.74.2: icmp_seq=8 ttl=59 time=2.12 ms
64 bytes from 151.4.74.2: icmp_seq=9 ttl=59 time=2.33 ms
64 bytes from 151.4.74.2: icmp_seq=10 ttl=59 time=2.06 ms
64 bytes from 151.4.74.2: icmp_seq=11 ttl=59 time=2.11 ms
64 bytes from 151.4.74.2: icmp_seq=12 ttl=59 time=2.60 ms
64 bytes from 151.4.74.2: icmp_seq=13 ttl=59 time=1.99 ms
64 bytes from 151.4.74.2: icmp_seq=14 ttl=59 time=1.06 ms
64 bytes from 151.4.74.2: icmp_seq=15 ttl=59 time=2.38 ms
^C
--- 151.4.74.2 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14177ms
rtt min/avg/max/mdev = 1.066/2.169/3.242/0.457 ms
client4:~#

x - + client1
12:32:03.069602 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 7, length 64
12:32:03.069642 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 7, length 64
12:32:04.089412 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 8, length 64
12:32:04.089444 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 8, length 64
12:32:05.099320 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 9, length 64
12:32:05.099335 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 9, length 64
12:32:06.109020 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 10, length 64
12:32:06.109020 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 10, length 64
12:32:06.600113 IP 151.4.74.1 > 224.0.0.5: OSPFv2, Hello, length: 44
12:32:07.119092 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 11, length 64
12:32:07.119131 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 11, length 64
12:32:08.129494 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 12, length 64
12:32:08.129533 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 12, length 64
12:32:09.138371 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 13, length 64
12:32:09.139010 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 13, length 64
12:32:10.148468 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 14, length 64
12:32:10.148491 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 14, length 64
12:32:11.169216 IP 110.3.74.3 > 151.4.74.2: ICMP echo request, id 5122, seq 15, length 64
12:32:11.169256 IP 151.4.74.2 > 110.3.74.3: ICMP echo reply, id 5122, seq 15, length 64
^C
153 packets captured
153 packets received by filter
0 packets dropped by kernel
client1:~#

```

In this test we ping from 'client4' to 'client1'. We ping between two nodes in order to test the connectivity between the two nodes. We know that we can successfully ping between client 4 and client 1 since client 1 (our destination node) responds with an echo reply. The ping test essentially records the packet's trip and any lost packets. In our case there were no packets being dropped suggesting that my implementation is valid.

Client6 to dnscom: [client6 - r7 - dnscom] [Two nodes from same network]

```

client6:~# ping 210.3.74.4
PING 210.3.74.4 (210.3.74.4) 56(84) bytes of data:
64 bytes from 210.3.74.4: icmp_seq=1 ttl=63 time=0.581 ms
64 bytes from 210.3.74.4: icmp_seq=2 ttl=63 time=0.841 ms
64 bytes from 210.3.74.4: icmp_seq=3 ttl=63 time=0.875 ms
64 bytes from 210.3.74.4: icmp_seq=4 ttl=63 time=0.825 ms
64 bytes from 210.3.74.4: icmp_seq=5 ttl=63 time=0.953 ms
64 bytes from 210.3.74.4: icmp_seq=6 ttl=63 time=0.874 ms
64 bytes from 210.3.74.4: icmp_seq=7 ttl=63 time=0.974 ms
64 bytes from 210.3.74.4: icmp_seq=8 ttl=63 time=0.867 ms
64 bytes from 210.3.74.4: icmp_seq=9 ttl=63 time=1.02 ms
64 bytes from 210.3.74.4: icmp_seq=10 ttl=63 time=1.02 ms
64 bytes from 210.3.74.4: icmp_seq=11 ttl=63 time=0.917 ms
64 bytes from 210.3.74.4: icmp_seq=12 ttl=63 time=0.301 ms
64 bytes from 210.3.74.4: icmp_seq=13 ttl=63 time=0.920 ms
64 bytes from 210.3.74.4: icmp_seq=14 ttl=63 time=0.788 ms
64 bytes from 210.3.74.4: icmp_seq=15 ttl=63 time=0.834 ms
64 bytes from 210.3.74.4: icmp_seq=16 ttl=63 time=0.994 ms
^C
--- 210.3.74.4 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 15154ms
rtt min/avg/max/mdev = 0.301/0.849/1.029/0.180 ms
client6:~#

12:49:26.739713 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 8, length 64
12:49:26.739753 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 8, length 64
12:49:27.750137 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 9, length 64
12:49:27.750175 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 9, length 64
12:49:28.760161 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 10, length 64
12:49:28.760193 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 10, length 64
12:49:29.769863 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 11, length 64
12:49:29.769914 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 11, length 64
12:49:30.779206 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 12, length 64
12:49:30.779238 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 12, length 64
12:49:31.780047 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 13, length 64
12:49:31.780091 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 13, length 64
12:49:32.657067 IP 210.3.74.3 > 224.0.0.5: OSPFv2, Hello, length: 44
12:49:32.789640 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 14, length 64
12:49:32.789686 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 14, length 64
12:49:33.799998 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 15, length 64
12:49:33.800036 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 15, length 64
12:49:34.810108 IP 110.3.74.5 > 210.3.74.4: ICMP echo request, id 4098, seq 16, length 64
12:49:34.810147 IP 210.3.74.4 > 110.3.74.5: ICMP echo reply, id 4098, seq 16, length 64
^C
61 packets captured
61 packets received by filter
0 packets dropped by kernel
dnscom:~#

```

In this test we ping from 'client6' to 'dnscom'. We again ping between two nodes in order to test the connectivity between the two nodes. We know that we can successfully ping between client 6 and dnscom since dnscom (our destination node) responds with an echo reply. The ping test essentially records the packet's trip and any lost packets. In our case there were no packets being dropped suggesting that my implementation is valid.

T-E2: Routing Procedure [4 marks]

Client 4 to Client 1: [client4 - r7 - r6 - r2 - r1 - r4 - client1] [Two nodes from the same network]

```

client4:~# traceroute 151.4.74.2
traceroute to 151.4.74.2 (151.4.74.2), 64 hops max, 40 byte packets
 1  110.3.74.1 (110.3.74.1)  1 ms  1 ms  0 ms
 2  100.1.74.1 (100.1.74.1) 11 ms  1 ms  1 ms
 3  110.2.74.2 (110.2.74.2) 12 ms  1 ms  1 ms
 4  151.0.74.1 (151.0.74.1) 11 ms  1 ms  1 ms
 5  151.5.74.2 (151.5.74.2)  6 ms  1 ms  1 ms
 6  151.4.74.2 (151.4.74.2)  9 ms  1 ms  1 ms

```

I used trace route to trace which path the packets took to get to dnscom. Once I found the list of routers, I used the command "show ip ospf route" on all the routers in order to print the 'routing tables' as required in the task.

x - + r7		x - + r6	
N	IA 151.4.74.0/24	[50] area: 1.1.1.1	N IA 151.4.74.0/24
		via 100.1.74.1, eth1	[40] area: 1.1.1.1
			via 110.2.74.2, eth1
x - + r2		x - + r1	
N	151.3.74.0/24	[20] area: 0.0.0.0	N 151.3.74.0/24
		via 151.1.74.2, eth1	[30] area: 0.0.0.0
			via 151.0.74.2, eth0
x - + r4			
N	151.4.74.0/24	[10] area: 0.0.0.0	
		directly attached to eth1	

Here I cropped the image since the other routes in the list were unnecessary. We can see that for each router, there is a routing table that holds a list of networks. From client4 to client1 the process starts from r7 since it's client's 4 default gateway. Looking at r7's routing information we can see that the total cost to get to client 1 (connected to network 151.4.74.0/36) and that we can get there if we go 100.1.74.1 (r6). From r6 to get to client 1 via 110.2.74.2 (r2). From r2 we can get to client 1 via 151.1.74.2 (r1). From r1 we can get to client 1 via 151.0.74.2 (r4). From r4 we

can get to 151.4.72.2 since it is directly attached to r4 through eth1.

Client6 to dnscom: [client6 - r7 - dnscom] [Two nodes from same network]

```
x - + client6
client6:~# traceroute 210.3.74.4
traceroute to 210.3.74.4 (210.3.74.4), 64 hops max, 40 byte packets
 1 110.3.74.1 (110.3.74.1) 1 ms 0 ms 0 ms
 2 210.3.74.4 (210.3.74.4) 1 ms 1 ms 1 ms
client6:~#
```

I used trace route to trace which path the packets took to get to dnscom. Once I found the list of routers, I used the command "show ip ospf route" on all the routers in order to print the 'routing tables' as required in the task.

```
x - + r7
N 210.3.74.0/29 [10] area: 1.1.1.1
    directly attached to eth0
```

Since client's 6 default gateway is r7 we look into r7's routing table. We look up the destination IP 210.3.74.4 and find that it is directly connected to r4 on eth0.

T-E3: Effect of shutting down r1 [4 marks]

Route Client 4 to Client 1, shutting down r1.

```
The system is going down for system halt NOW!
INIT: Switching to runlevel: 0
INIT: Sending processes the TERM signal

--- Starting Netkit phase 2 shutdown script ---
--- Netkit shutdown phase 2 terminated ---

Stopping kernel log daemon....
Stopping system log daemon....

--- Starting Netkit phase 1 shutdown script ---
Unmounting and removing /hostlab...
Unmounting /lib/modules...
Unmounting and removing /hosthome...
--- Netkit phase 1 shutdown terminated ---

Asking all remaining processes to terminate...done.
All processes ended within 1 seconds....done.
Stopping portmap daemon....
Deconfiguring network interfaces...done.
Cleaning up ifupdown....
Deactivating swap...done.
Will now halt.
```

Before shutting down r1: Route from Client 4 to Client 1

```
client4:~# traceroute 151.4.74.2
traceroute to 151.4.74.2 (151.4.74.2), 64 hops max, 40 byte packets
 1 110.3.74.1 (110.3.74.1) 1 ms 1 ms 0 ms
 2 100.1.74.1 (100.1.74.1) 11 ms 1 ms 1 ms
 3 110.2.74.2 (110.2.74.2) 12 ms 1 ms 1 ms
 4 151.0.74.1 (151.0.74.1) 11 ms 1 ms 1 ms
 5 151.5.74.2 (151.5.74.2) 6 ms 1 ms 1 ms
 6 151.4.74.2 (151.4.74.2) 9 ms 1 ms 1 ms
client4:~#
```

```
x - + r7
N IA 151.4.74.0/24 [55] area: 1.1.1.1
    via 100.1.74.1, eth1
```


After shutting down r1: Route from Client 4 to Client 1

```
x - + client4
client4:~# traceroute 151.4.74.2
traceroute to 151.4.74.2 (151.4.74.2), 64 hops max, 40 byte packets
 1 110.3.74.1 (110.3.74.1) 0 ms 1 ms 0 ms
 2 100.1.74.1 (100.1.74.1) 1 ms 1 ms 1 ms
 3 110.2.74.2 (110.2.74.2) 1 ms 1 ms 1 ms
 4 151.3.74.2 (151.3.74.2) 1 ms 1 ms 1 ms
 5 151.5.74.2 (151.5.74.2) 1 ms 1 ms 1 ms
 6 151.4.74.2 (151.4.74.2) 1 ms 1 ms 2 ms

x - + r7
N IA 151.4.74.0/24      via 100.1.74.1, eth1
                        [90] area: 1.1.1.1
N IA 151.5.74.0/24      via 100.1.74.1, eth1
                        [90] area: 1.1.1.1
```

After shutting down r1, from traceroute we can see that the route taken to the destination has changed. The optimal route had r1 in the set, with r1 shut down the OSPF is forced to recalculate the shortest route taking in account of cost as well. Therefore, the cost to our destination has changed from 55 to 90. The loss of a route can be significant as it can negatively or positively impact the network, depending on the network topology and costs. However, in our case the loss of r1 did not shut down or corrupt our network by any means.

T-E4: Effect of changing interface cost [4 marks]

Route Client 4 to Client 1, changing the cost of r2 eth1 from 50 to default cost 10.

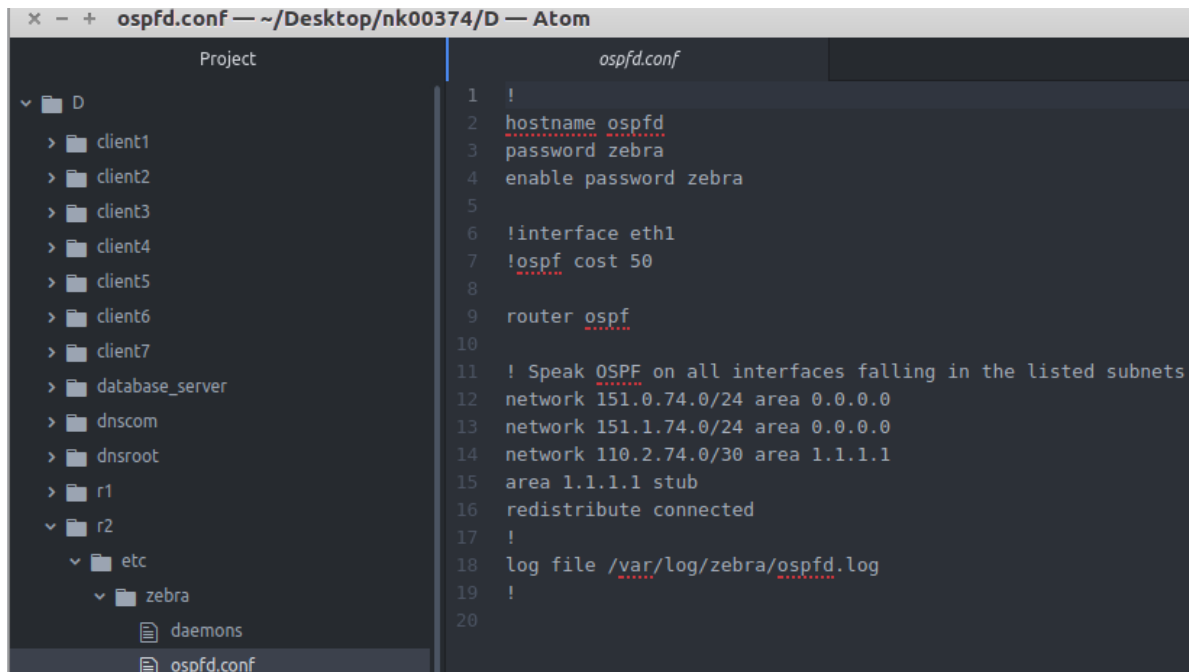
```
x - + r7
N IA 151.4.74.0/24      [55] area: 1.1.1.1
                        via 100.1.74.1, eth1
```

Normal Route before change: Cost is 55 [client4 – r7 – r6 – r2 – r1 – r4 – client1] The screenshot above shows the cost of the route. I used the command 'show ip ospf route' to get the routing information to find the cost to a route

```
client4:~# traceroute 151.4.74.2
traceroute to 151.4.74.2 (151.4.74.2), 64 hops max, 40 byte packets
 1 110.3.74.1 (110.3.74.1) 1 ms 1 ms 0 ms
 2 100.1.74.1 (100.1.74.1) 11 ms 1 ms 1 ms
 3 110.2.74.2 (110.2.74.2) 12 ms 1 ms 1 ms
 4 151.0.74.1 (151.0.74.1) 11 ms 1 ms 1 ms
 5 151.5.74.2 (151.5.74.2) 6 ms 1 ms 1 ms
 6 151.4.74.2 (151.4.74.2) 9 ms 1 ms 1 ms
```

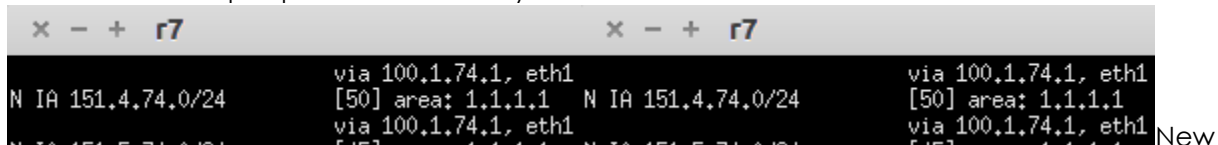
This screenshot shows the route from client 4 to client 1. Comparing this and to the network topology I double checked the cost and found that the cost is correct, and the path is the shortest.

Below I edited the ospf.conf file in r2, I changed the cost from 50 to 10 by commenting out the code. By commenting out the code the default cost will be set to 10.



```
1 !
2 hostname ospfd
3 password zebra
4 enable password zebra
5
6 !interface eth1
7 !ospf cost 50
8
9 router ospf
10
11 ! Speak OSPF on all interfaces falling in the listed subnets
12 network 151.0.74.0/24 area 0.0.0.0
13 network 151.1.74.0/24 area 0.0.0.0
14 network 110.2.74.0/30 area 1.1.1.1
15 area 1.1.1.1 stub
16 redistribute connected
17 !
18 log file /var/log/zebra/ospfd.log
19 !
20
```

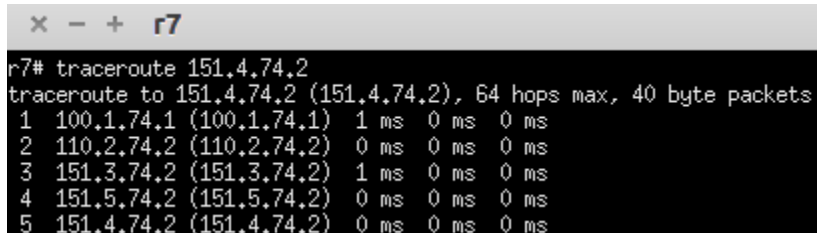
Here I ran 'show ip ospf route' to identify the cost of the new route to client 1.



```
via 100.1.74.1, eth1
[50] area: 1.1.1.1
via 100.1.74.1, eth1

via 100.1.74.1, eth1
[10] area: 1.1.1.1
via 100.1.74.1, eth1
```

Calculated Route Cost: 50, then I used 'traceroute 151.4.74.2' to find the route path.



```
r7# traceroute 151.4.74.2
traceroute to 151.4.74.2 (151.4.74.2), 64 hops max, 40 byte packets
 1 100.1.74.1 (100.1.74.1) 1 ms 0 ms 0 ms
 2 110.2.74.2 (110.2.74.2) 0 ms 0 ms 0 ms
 3 151.3.74.2 (151.3.74.2) 1 ms 0 ms 0 ms
 4 151.5.74.2 (151.5.74.2) 0 ms 0 ms 0 ms
 5 151.4.74.2 (151.4.74.2) 0 ms 0 ms 0 ms
```

New route path: [client4 – r7 – r6 – r2 – r5 – r4 – client1], Cost: 50

Old route path: [client4 – r7 – r6 – r2 – r1 – r4 – client1], Cost: 55

We can see that by changing r2's eth1's cost from 50 to 10, we found that we could save a cost of 5. If r2's eth1's cost was not changed and remained 50, the cost of the route would have been 90.

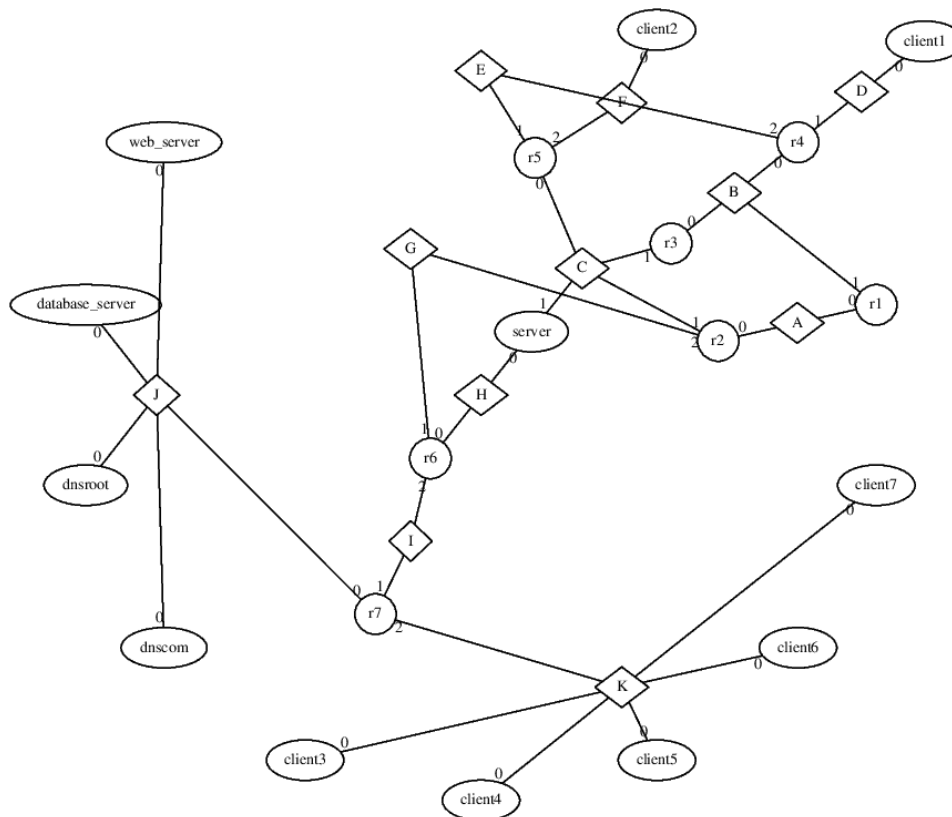
T-E5: Connecting Server to network C [4 marks]

For this task in order to understand the implications I decided to implement the change by adding the server to network C. A series of screenshot's below proves that I have done this. The image below shows

the changes I made to my files for the task.

```
Project      server.startup      lab.conf
> client5    1 ifconfig eth0 210.2.74.1 netmask 255.255.255.252 broadcast 210.2.74.3 up 1 web_server[0]=J
> client6    2 ifconfig eth1 151.1.74.4 netmask 255.255.255.0 broadcast 151.1.74.255 up 2 database_server[0]=J
> client7    3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
> database_server 4 route add default gw 210.2.74.2 dev eth0 5 server[0]=H
> dnscom     5 route add default gw 151.1.74.1 dev eth1 6 server[1]=C
> dnsroot   6 route add default gw 151.1.74.2 dev eth1
> dnsroot   7 route add default gw 151.1.74.3 dev eth1
> r1         8
> r2         9 a2enmod userdir
> r3         10 /etc/init.d/apache2 start
> r4         11
> r5         12
> r6         13
> r7         14
> etc        15
> zebra      16
> daemons    17
> ospfd.conf 18
> server     19
> web_server 20
> client1.disk 21
> client1.ready 22
> client1.startup 23
> client2.disk 24
> client2.ready 25
> client2.startup 26
> client3.disk 27
> client3.ready 28
> client3.startup 29
> client4.disk 30
> client4.ready 31
> client4.startup 32
> client5.disk 33
> client5.ready 34
> client5.startup 35
> client6.disk 36
> client6.ready 37
> client6.startup 38
> client7.disk 39
> client7.ready 40
> client7.startup 41
> client8.disk 42
> client8.ready 43
> client8.startup 44
```

I recreated the graph using the command "linfo -m net.ps" to see if the changes were applied. From the results of the graph we can see that the changes seem to be ok.



The implications of connecting the server to network C is quite significant. My initial thoughts were that trying to ping from server to client 2 we would see the server send its packets through

the new added route since r2 eth1 interface cost is 50, because of this the path taken to get to client 2 through r2 eth1 would less likely be the optimal path.

Again, If we connected server to network C I had thought (with my implementation) that the server would be able to directly go from server to r5 to client 2. I added 4 default static gateways for the server, thinking that the server will pick the best gateway holding the optimal path to the destination. However, after testing using trace route, I found that my theory was incorrect, this is because I believe you cannot have multiple default gateways. As a solution I tried to add static routes in the server startup files. These static routes would automatically load in the router's routing table. I tried to add the following line to the server file;
route add -net 151.2.74.2 netmask 255.255.255.0 gw 151.1.74.1 dev eth1, however the route was not added. Therefore, I removed the other gateways from the server and left the gateway to r6.

If the server could use multiple gateways, the addition of connecting the server to network C will allow packets to reach destinations through quicker routes. Though the disadvantages would be that you would need to add a static route to which ever destination the packets need to be set too. This job could be tedious and lengthy especially if the network was large. Take for example server to client 2.

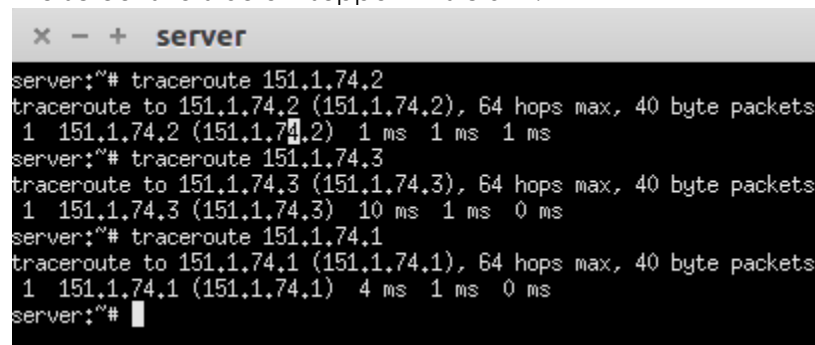
Here are all the non-cyclic paths when exchanging packets from server to client 2 along with their associated cost.

Route 1: [Server – r6 – r2 (eth1) – r5 – client2] [Cost: 70]
Route 2: [Server – r6 – r2 (eth0)– r1 – r3 – r5 – client2] [Cost:65] <- Smallest route
Route 3: [Server – r6 – r2 (eth0)– r1 – r4 – r5 – client2] [Cost:95]
Route 4: [Server – r6 – r2 (eth1)– r3 – r4 – r5 – client2] [Cost:194]

We can see that Route 2 is the optimal route since if we exit r2 through eth1 then the hefty cost of 50 will make the route less optimal. This effect is visualized by looking at route 1 and route 4 we can see how leaving through eth1 had a high overall cost even though route 1 had passed through a few nodes.

Comparing this with the new changes, the server would be able to get to client 2 through r5 with only a cost of 10. This can be done if r5 was set as a gateway (but currently r6 is, and when I tried, I ran into a lot of problems). This is a big implication as this means that packets can get across to another network area at an optimal rate. This cannot be done with my implementation, instead the server can directly send packets to the adjacent routers: r2, r3 and r5.

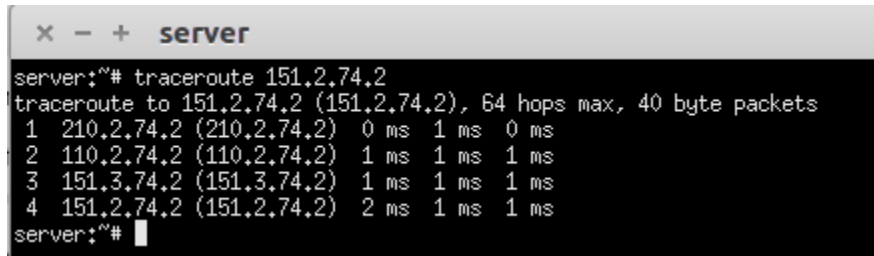
The screenshots below support this claim.



```
x - + server
server:~# traceroute 151.1.74.2
traceroute to 151.1.74.2 (151.1.74.2), 64 hops max, 40 byte packets
 1 151.1.74.2 (151.1.74.2) 1 ms 1 ms 1 ms
server:~# traceroute 151.1.74.3
traceroute to 151.1.74.3 (151.1.74.3), 64 hops max, 40 byte packets
 1 151.1.74.3 (151.1.74.3) 10 ms 1 ms 0 ms
server:~# traceroute 151.1.74.1
traceroute to 151.1.74.1 (151.1.74.1), 64 hops max, 40 byte packets
 1 151.1.74.1 (151.1.74.1) 4 ms 1 ms 0 ms
server:~#
```

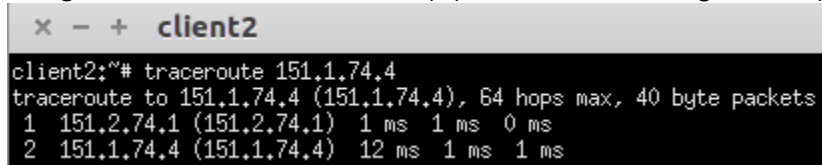
In this screen shot, the server only takes one hop to get to r5. This is much quicker than hoping to r2 then to r5 (which is achieved when the server is not connected to network C). We can do the

same with r2 and r3.



```
server:~# traceroute 151.2.74.2
traceroute to 151.2.74.2 (151.2.74.2), 64 hops max, 40 byte packets
 1 210.2.74.2 (210.2.74.2) 0 ms 1 ms 0 ms
 2 110.2.74.2 (110.2.74.2) 1 ms 1 ms 1 ms
 3 151.3.74.2 (151.3.74.2) 1 ms 1 ms 1 ms
 4 151.2.74.2 (151.2.74.2) 2 ms 1 ms 1 ms
server:~#
```

In this screenshot, when we send packets from the server to the client, we can see the server using 151.2.74.2 (r6) as a default gateway which is an expected behavior though since it is connected to network C there should be a way to simply send the packets through to r5 then straight to client 2. This would simply cost 10 which is significantly less than 64.



```
client2:~# traceroute 151.1.74.4
traceroute to 151.1.74.4 (151.1.74.4), 64 hops max, 40 byte packets
 1 151.2.74.1 (151.2.74.1) 1 ms 1 ms 0 ms
 2 151.1.74.4 (151.1.74.4) 12 ms 1 ms 1 ms
```

In this screenshot we can see that we can simply ping from a node to the server on network c as expected.

References:

[1] Joddies.de "IP Calculator" [Online] . Available at <http://jodies.de/ipcalc>

[Accessed: 23/04/2019]

[2] Cisco.com "Load balancing with Cisco Express Forwarding" [Online]. Available at https://www.cisco.com/en/US/products/hw/modules/ps2033/prod_technical_reference09186a00800afeb7.html [Accessed: 24/04/2019]

[3] Cisco.com "OSPF Design Guide" [Online]. Available at <https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html> [Accessed: 24/04/2019]