1. connection between two devices using cisco packet tracer

    1. **Open Cisco Packet Tracer** and place two devices (e.g., two PCs or a PC and a switch) onto the workspace.

    2. **Add appropriate network interfaces** if needed (e.g., Ethernet on both PCs).

    3. **Use the "Copper Straight-Through" cable** to connect the devices directly or through a switch/router.

    4. **Assign IP addresses** to both devices manually via the desktop → IP configuration (make sure they're in the same subnet).

    5. **Test the connection** using the `ping` command from one device to the other to ensure successful communication.


```
Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

2.Develop a client-server application using both TCP and UDP communication.

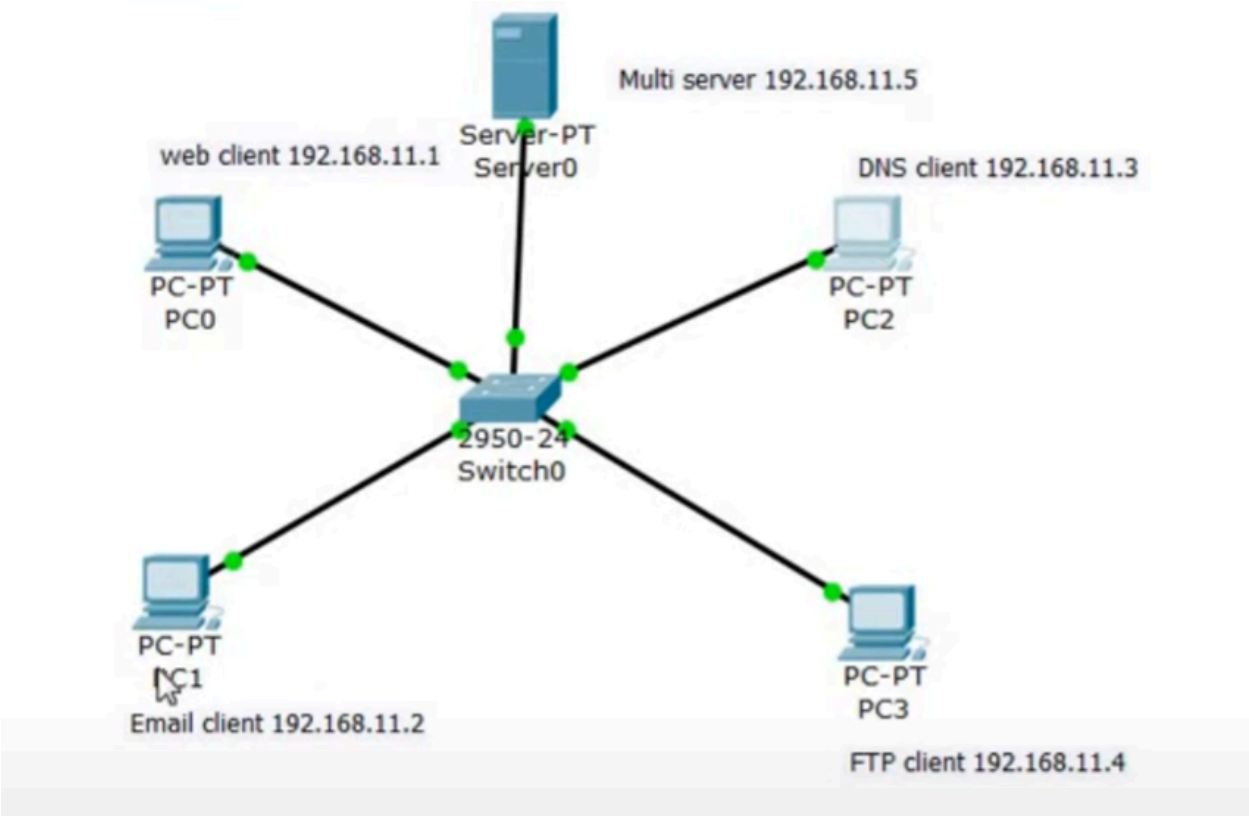
1. **Create the TCP server and client** to establish a reliable connection using `socket()` with `SOCK_STREAM`, and handle data exchange (e.g., messages or files).

2. **Create the UDP server and client** using `socket()` with `SOCK_DGRAM`, allowing connectionless communication for faster but less reliable data transfer.

3. **Bind server sockets** to specific ports and wait for incoming requests from clients (for both TCP and UDP).

4. **Send and receive data** between client and server over both TCP (ensuring reliability) and UDP (for speed or non-critical data).

5. **Close all sockets properly** after communication is complete, ensuring clean termination of both TCP and UDP sessions.



3.Implement the stop and wait protocol using python from reliable data transfer.

1. **Split the message into packets** and assign sequence numbers (usually alternating between 0 and 1).

2. **Send one packet at a time** from sender to receiver and start a timer to wait for an acknowledgment (ACK).

3. **Receiver checks the sequence number**, sends back an ACK if it matches the expected one, and stores the packet.

4. **If the sender receives the correct ACK within timeout**, it sends the next packet; otherwise, it resends the same one.

5. **Repeat the process** until all packets are sent and acknowledged, ensuring complete and reliable transmission.

receiver.py

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host ="localhost"
port =8000
s.connect((host,port))



while 2:
  data=s.recv(1024).decode()
  print("Received --> "+data)
  str="Acknowledgement: Message Received"
  s.send(str.encode())

s.close ()
```

sender.py

```
import socket
from threading import *

serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "localhost"
port = 8000

serversocket.bind((host, port))

class client(Thread):
    def __init__(self, socket, address):
        Thread.__init__(self)
        self.sock = socket
        self.addr = address
        self.start()

    def run(self):
        while 1:
            r=input("Send data -->")
            clientsocket.send(r.encode())
            print(clientsocket.recv(1024).decode())


serversocket.listen(5)
print ('Sender ready and is listening')
while (True):

    #to accept all incoming connections
    clientsocket, address = serversocket.accept()
    print("Receiver "+str(address)+" connected")
    #create a different thread for every
    #incoming connection
    client(clientsocket, address)
```

4 Develop a setup to analyze subnets, including network, broadcast addresses, and host ranges

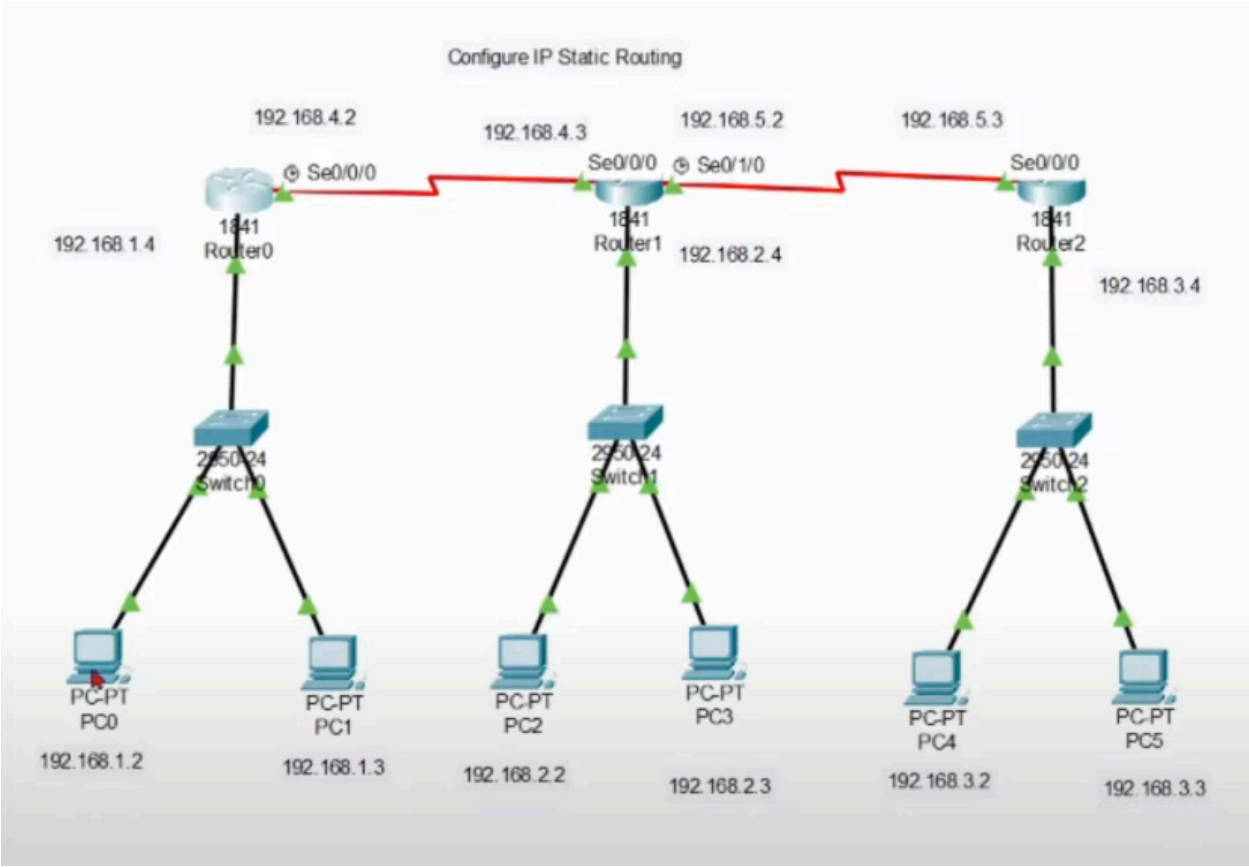
1. **Input the IP address and subnet mask** (e.g., `192.168.1.10/24`) to begin subnet analysis.

2. **Calculate the network address** by performing a bitwise AND between the IP address and subnet mask.

3. **Determine the broadcast address** by setting all host bits to 1 in the subnet.

4. **Calculate the valid host range** by identifying the first and last usable IP addresses in the subnet.

5. **Display the results** showing the network address, broadcast address, and the range of valid host IPs.



5 Create a Wi-Fi network, evaluate throughput, latency, and packet loss.

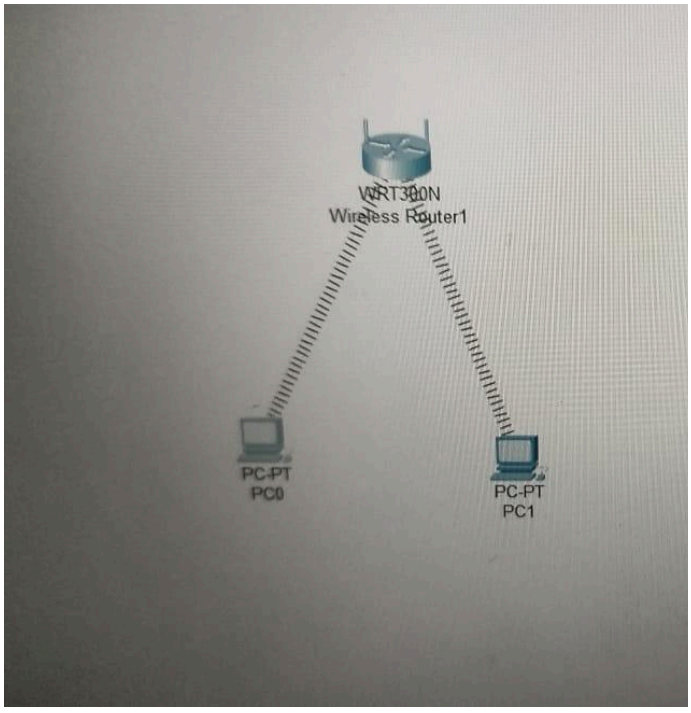
1. **Set up a Wi-Fi network** using a wireless router and connect client devices (laptops, phones) via Wi-Fi.

2. **Assign IP addresses** automatically via DHCP or manually configure static IPs for each connected device.

3. **Use the `ping` command** to measure **latency** and check for **packet loss** by sending ICMP requests to another device or router.

4. **Measure throughput** using tools like **iPerf** or **Speedtest.net** to determine how much data can be transferred per second.

5. **Analyze the results**: compare latency (in ms), throughput (in Mbps), and packet loss (as %) to evaluate network performance.



6.Diagnose the network issues or test the performance of your internet connection with two fundamental tools come into play — Ping and Traceroute

1. **Use the `ping` command** to check if a remote host is reachable and how long it takes for packets to travel.

2. **Examine the ping output** for response time, packet loss, and any fluctuations indicating network instability.

3. **Run `traceroute` (or `tracert`)** to map out the path that packets take to the destination through different routers.

4. **Check each hop in the traceroute** for delays or timeouts that may indicate congestion or a failing node.

5. **Compare ping and traceroute results** to determine where in the network path issues like latency or loss are occurring.