

Term work of

Computer Networks Lab (PCS – 604)

Submitted in partial fulfillment of the requirement for the VI semester

**Bachelor of Technology**

By

Aditya

23

**Under the Guidance of**

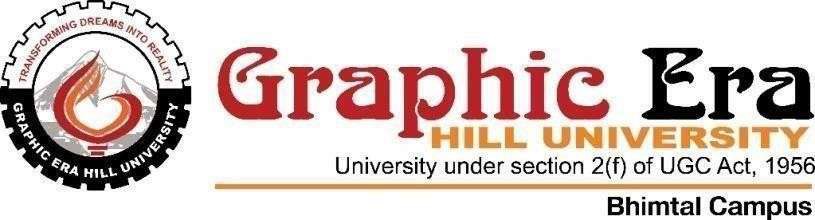
Pandey

**Assistant Professor Department of CSE**

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

**SATTAL ROAD, P.O. BHOWALI DISTRICT-**

**NAINITAL-263132 2024-2025**



CERTIFICATE

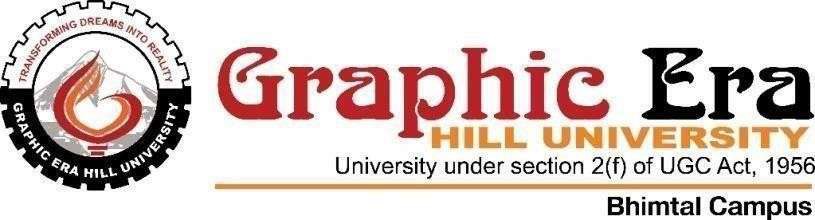
The term work of Computer Networks Lab , being submitted by Aditya

S/O Mr. LD Sharma University Roll Number 23

to Graphic Era Hill University, Bhimtal Campus for the award of bona fide work carried out by him. Aditya has worked under my guidance and supervision and fulfilled the requirement for the submission of this work report.

**(Pandey) (Dr. Ankur Singh Bisht)**

**Assistant Professor HOD, CSE Dept.**



**ACKNOWLEDGEMENT**

I take immense pleasure in thanking Honorable **Mr. Aditya** (Assistant Professor, Dept. of CSE, GEHU Bhimtal Campus) for allowing me to carry out this practical work under his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance and useful suggestions that have helped me in developing my subject concepts as a student.

I want to extend thanks to our President **Prof. (Dr.) Kamal Ghanshala** for providing us all infrastructure and facilities to work in need without which this work would not be possible.

# Aditya

**University Roll Number:** 23



**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S No.** | **Experiment** | **Date** | **Sign** |
| 1. | Familiarization of Network Environment, Understanding and using network utilities: ipconfig, netstat, ping, telnet, ftp, traceroute etc. |  |  |
| 2. | Familiarization with Transmission media and tools: Co-axial cable, UTP cable, Crimping tool, Connectors etc. Preparing the UTP cable for cross and direct connection using crimping tool. |  |  |
| 3. | Installation and introduction of simulation tool. (Packet Tracer) |  |  |
| 4. | To configure a basic network topology consisting of routers, switches, and end devices such as PCs or laptops. Configure IP addresses and establish connectivity between devices. (Using packet Tracer) |  |  |
| 5. | To configure a DHCP server on a router or a dedicated DHCP server device. Assign IP addresses dynamically to devices on the network and verify successful address assignment. (Using packet Tracer) |  |  |
| 6. | To configure a local DNS server to resolve domain names within a network. (Using packet Tracer) |  |  |
| 7. | Network Troubleshooting: Simulate network issues such as connectivity problems, incorrect configurations, or routing failures. Use Packet Tracer's simulation mode to diagnose and troubleshoot the network. |  |  |
| 8. | NAT (Network Address Translation): Set up NAT on a router to translate private IP addresses to public IP addresses for outbound internet connectivity. Test the translation and examine how NAT helps conserve IPv4 address space. (Using packet Tracer) |  |  |
| 9. | TCP Client-Server Communication:  Implement a TCP client program that sends a message to a TCP server program.  Implement the corresponding TCP server program that receives the message and displays it.  Test the communication between the client and server by exchanging messages  (Using ‘C’ Language) |  |  |
| **10.** | UDP Client-Server Communication:  Implement a UDP client program that sends a message to a UDP server program.  Implement the corresponding UDP server program that receives the message and displays it (Using ‘C’ Language). |  |  |

## Familiarization of Network Environment, Understanding and using network utilities: ipconfig, netstat, ping, telnet, ftp, traceroute etc.

* 1. **ipconfig (Internet Protocol Configuration)** ipconfig is a command-line utility available on Windows systems that displays and manages the IP address configuration of network interfaces. It is commonly used to view the current IP address, subnet mask, and default gateway assigned to the system. Advanced options allow users to release and renew IP addresses via DHCP.

### Command Examples:

ipconfig # Shows basic network info

ipconfig /all # Shows detailed info (MAC address, DNS, DHCP status etc.)

* 1. **ifconfig (Interface Configuration)** ifconfig is the Linux/macOS equivalent of ipconfig. It is used to display, configure, and manage network interfaces. It can show IP addresses, MAC addresses, and allow enabling or disabling of interfaces.

### Command Examples:

ifconfig # Show IP and MAC of interfaces sudo ifconfig eth0 down # Disable a network interface

* 1. **ping (Packet Internet Groper)** ping is a diagnostic tool used to test the reachability of a host on an IP network. It sends ICMP echo request packets to the target host and measures the time taken for the responses. It helps determine whether the destination device is online and how fast it responds. **Command Example:**

ping google.com # Sends packets to Google's servers ping 192.168.1.1 # Tests connectivity with local router

### tracert (Windows) / traceroute (Linux/macOS)

This utility traces the route that packets take to reach a destination host. It lists all the intermediate routers the packet passes through, along with the time taken at each hop. This is useful for identifying network bottlenecks or failures.

### Command Example:

tracert google.com

* 1. **netstat (Network Statistics)** netstat is a command-line utility that displays active network connections, listening ports, and network protocol statistics. It helps users monitor open connections, detect suspicious activity, and troubleshoot network issues.

### Command Examples:

netstat # Shows active connections

netstat -a # Shows all active ports and listening ports

* 1. **telnet (Teletype Network Protocol)** telnet is a network protocol and command-line tool used to establish a text-based communication session with a remote host using the TCP/IP protocol. It is often used to test connectivity to a specific port (like 80 for HTTP or 25 for SMTP), though it is now mostly replaced by more secure alternatives like SSH.

### Command Example:

telnet google.com 80 # Test connection to port 80 (HTTP)

## ftp (File Transfer Protocol)

ftp is a standard network protocol used to transfer files between a client and a server over a TCP-based network. The ftp command-line tool allows users to upload and download files, authenticate with remote servers, and navigate remote directories.

### Command Example:

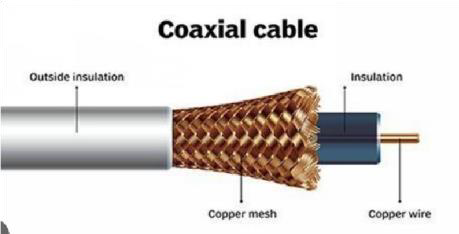
ftp ftp.example.com

## Familiarization with Transmission media and tools: Co-axial cable, UTP cable, Crimping tool, Connectors etc. Preparing the UTP cable for cross and direct connection using crimping tool.

### Coaxial Cable (Co-axial Cable)

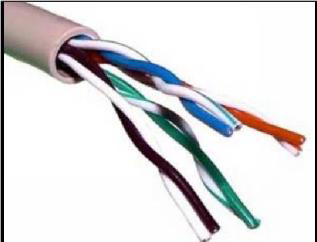
A coaxial cable is a type of electrical cable with an inner conductor surrounded by a concentric conducting shield, separated by an insulating layer. It is commonly used for cable television, internet connections, and long-distance communication due to its excellent shielding from electromagnetic interference (EMI).

Use Case: Broadband internet, CCTV, cable TV Connector Type: BNC, F-type



### UTP Cable (Unshielded Twisted Pair)

A UTP cable consists of pairs of insulated copper wires twisted together. It lacks shielding, making it cheaper and more flexible, but more vulnerable to EMI. It is widely used in Ethernet networks and telephone systems. Types: Cat5, Cat5e, Cat6

Max Range (Ethernet): ~100 meters

### Crimping Tool

A crimping tool is a hand tool used to attach connectors (such as RJ-45) to the ends of UTP cables. It ensures a secure and reliable electrical connection by pressing the connector’s pins into the cable wires. Function: Terminating cables with RJ-45 connectors

### Connectors (RJ-45)

RJ-45 connectors are modular plugs used to connect UTP cables to networking devices. They have 8 pins that correspond to the 8 wires in a UTP cable. These connectors are crucial in forming both straight- through and crossover cables.

Pin Count: 8P8C (8 Positions, 8 Contacts) Use: Ethernet, LAN connections



### Preparing UTP Cable for Cross and Direct Connection

**Straight-through Cable (Direct Connection)** Used to connect different types of devices (e.g., PC → Switch, PC → Router). Wiring Standard:

Both ends use same color coding (usually T568B) T568B Wiring Order:

1. Orange-White
2. Orange
3. Green-White
4. Blue
5. Blue-White
6. Green
7. Brown-White
8. Brown

### Crossover Cable (Cross Connection)

Used to connect similar devices (e.g., PC ↔ PC, Switch ↔ Switch). Wiring Standard:

One end is T568A, the other is T568B T568A End: T568B End:

* 1. Green-White ↔ Orange-White
  2. Green ↔ Orange
  3. Orange-White ↔ Green-White
  4. Blue ↔ Blue
  5. Blue-White ↔ Blue-White
  6. Orange ↔ Green
  7. Brown-White ↔ Brown-White
  8. Brown ↔ Brown

### Steps to Crimp a UTP Cable

1. Strip ~1 inch of the cable jacket using the crimping tool.
2. Untwist and align the wires as per the color code (T568A or B).
3. Trim wires evenly using the cutter.
4. Insert wires into the RJ-45 connector — ensure all reach the end.
5. Insert connector into the crimping slot and press hard.
6. Repeat on the other side as per the required cable type (cross/direct).

## Installation and introduction of simulation tool. (Packet Tracer)

### Introduction to Cisco Packet Tracer

Cisco Packet Tracer is a powerful network simulation and visualization tool developed by Cisco Systems. It is widely used in academic environments, particularly in Cisco Networking Academy programs, to provide students and networking professionals with a realistic and interactive platform for designing, configuring, and troubleshooting network topologies without requiring physical networking equipment. Packet Tracer supports a wide range of networking components, including routers, switches, wireless devices, PCs, and IoT devices. It provides both real-time and simulation modes, enabling users to observe packet flow and protocol behavior in detail.

### Key Features of Packet Tracer Feature Description

Device Simulation Simulates routers, switches, PCs, laptops, servers, firewalls, and IoT devices.

Real-time Mode Emulates live network behavior for active packet flow and protocol execution.

Simulation Mode Allows users to step through packet transmission for deeper analysis.

CLI Support Provides a command-line interface similar to Cisco IOS for configuring devices.

Logical and Physical Views

Users can design topologies logically and visualize physical arrangements.

Multi-User Collaboration Supports network collaboration in a classroom or remote learning environment.

Includes basic IoT device simulation and allows programming using

IoT and Programming

JavaScript or

Python.

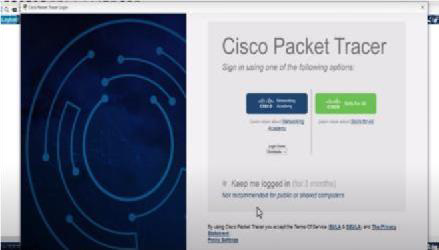
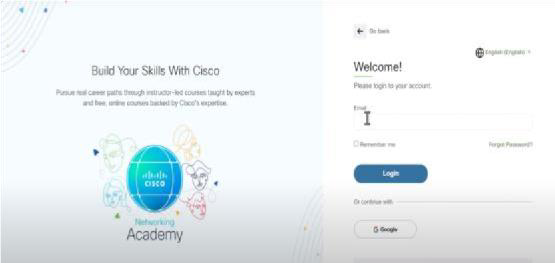
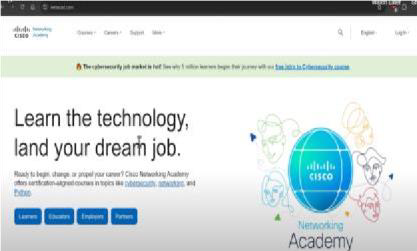
### Importance of Packet Tracer

* Enables hands-on learning of networking concepts.
* Eliminates the cost of purchasing physical routers and switches.
* Ideal for practicing CCNA, CCNP, and other networking certifications.
* Facilitates experimentation and troubleshooting in a risk-free environment.  Supports remote learning, making it accessible to students globally.

### Installation Steps for Cisco Packet Tracer

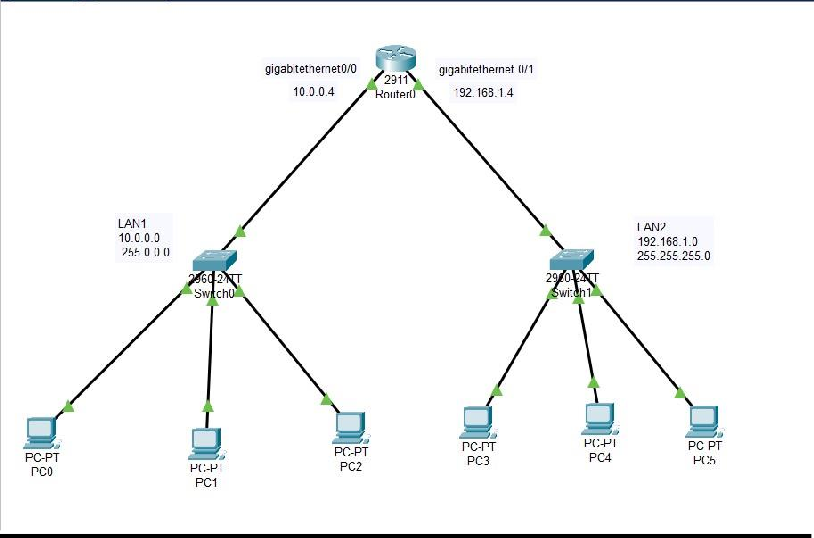
Step-by-Step Installation Process (Windows)

* 1. Register on Cisco Networking Academy: o Visit: https:/[/www.netacad.com](http://www.netacad.com/) o Create a free account and enroll in the *Introduction to Packet Tracer* course.
  2. Download the Software: o After enrollment, navigate to the Packet Tracer download section.
     + Choose the appropriate version for your operating system.
  3. Install Packet Tracer: o Run the downloaded installer file.
     + Accept the license agreement. o Choose the installation directory (default is recommended).
     + Complete the installation process.
  4. Launch and Sign In:
     + Open Packet Tracer.
     + Log in using your Cisco Networking Academy credentials.

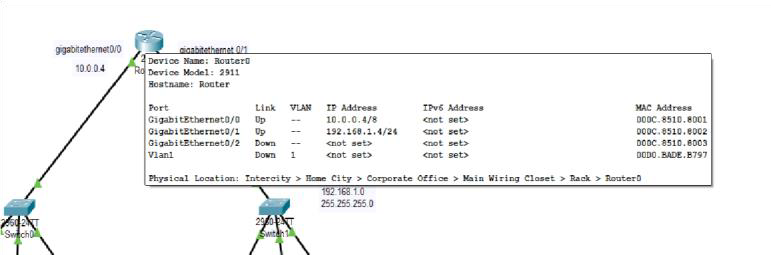
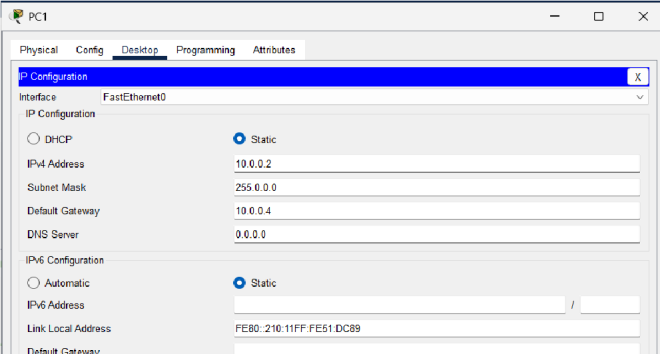


## To configure a basic network topology consisting of routers, switches, and end devices such as PCs or laptops. Configure IP addresses and establish connectivity

**between devices. (Using packet Tracer)**



Step 1: Place Devices



* 1 Router (e.g. 2811)
* 2 Switches (e.g. 2960)
* 4 PCs (2 per network, PC0 & PC1 on SW1, PC2 & PC3 on SW2)

Step 2: Connect with Cables

Use copper straight-through cables:

* PC0 → SW1 (any FastEthernet port)
* PC1 → SW1
* PC2 → SW2
* PC3 → SW2
* SW1 → Router (Router’s FastEthernet0/0)
* SW2 → Router (Router’s FastEthernet0/1)

Use Router interfaces that are FastEthernet (Fa0/0, Fa0/1) or GigabitEthernet depending on router model.

Step 3: Assign IP Addresses Network 1:

* Router Fa0/0 → 192.168.1.1 /24
* PC0 → 192.168.1.10 /24, Gateway: 192.168.1.1
* PC1 → 192.168.1.11 /24, Gateway: 192.168.1.1 Network 2:
* Router Fa0/1 → 192.168.2.1 /24
* PC2 → 192.168.2.10 /24, Gateway: 192.168.2.1
* PC3 → 192.168.2.11 /24, Gateway: 192.168.2.1

Step 4: Configure Router Interfaces Click the Router → CLI tab → type the following: enable configure terminal interface FastEthernet0/0 ip address 192.168.1.1 255.255.255.0

no shutdown interface FastEthernet0/1 ip address 192.168.2.1 255.255.255.0

no shutdown exit

Step 5: Configure PCs

Click on each PC → Desktop tab → IP Configuration: PC0:

IP: 192.168.1.10

Subnet: 255.255.255.0

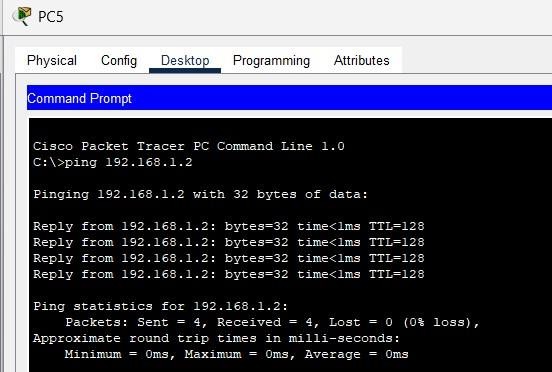
Gateway: 192.168.1.1

Repeat similar for all PCs using IPs we assigned.

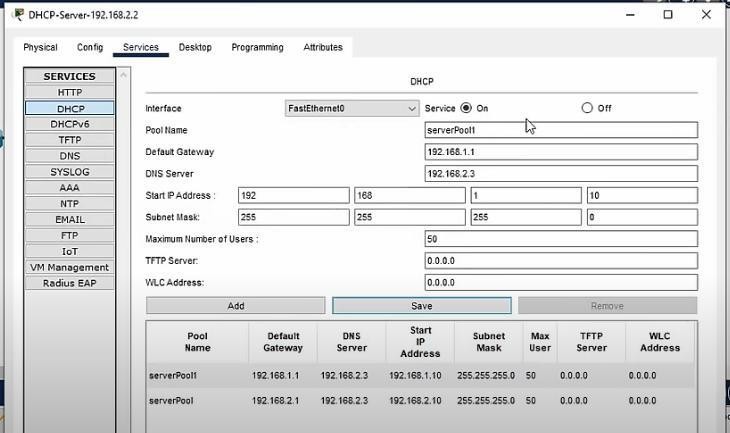
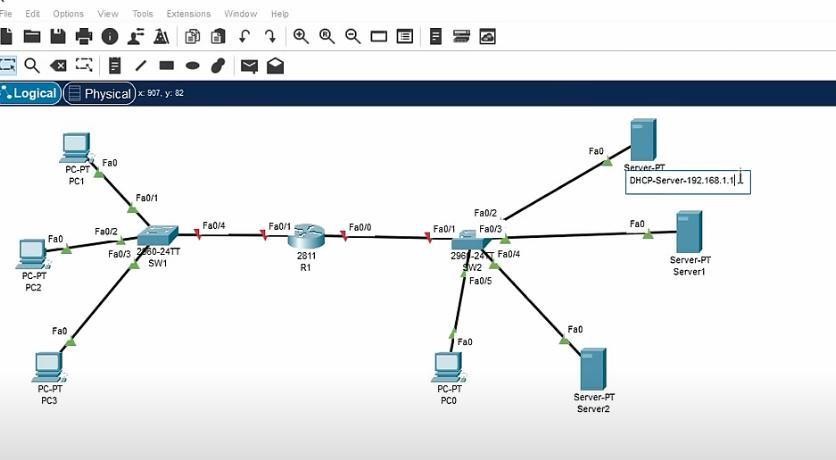
Step 6: Test Connectivity

Open PC0 → Desktop → Command Prompt → type: ping 192.168.1.11 # PC1 (same LAN)

ping 192.168.2.10 # PC2 (other LAN via router)



## To configure a DHCP server on a router or a dedicated DHCP server device. Assign IP addresses dynamically to devices on the network and verify successful address assignment. (Using packet Tracer)



### Step 1: Connect devices

* + Use **straight-through cables**:

o PC0, PC1, PC2 → Switch o

Switch → Router’s FastEthernet0/0

### Step 2: Decide IP Pool

Let’s say we want to assign:

* + **Network**: 192.168.10.0/24

 **Gateway (router)**: 192.168.10.1  **IP Pool Range**: 192.168.10.100 to 192.168.10.200 **Step 3: Configure Router as DHCP Server**

Click Router → CLI tab : enable configure terminal

! Set up the DHCP pool ip dhcp pool AyushNet

network 192.168.10.0 255.255.255.0

default-router 192.168.10.1 dns-

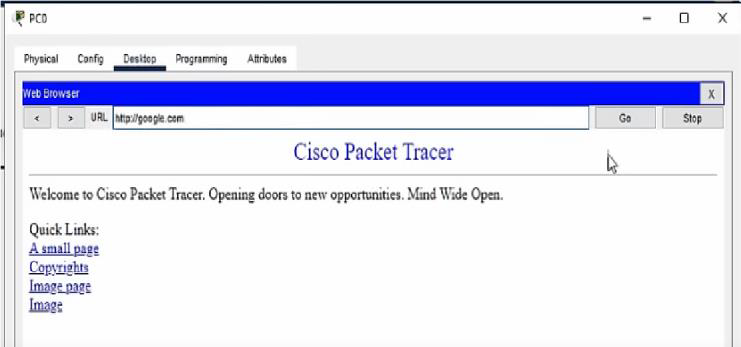
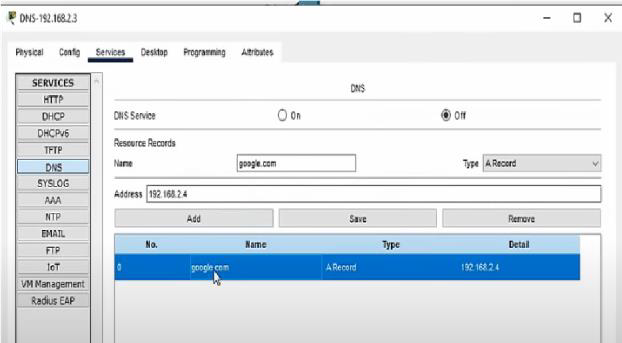
server 8.8.8.8

! Exclude some addresses (like gateway, servers, etc.) ip dhcp excluded-address 192.168.10.1 192.168.10.99

! Assign IP to router interface (gateway) interface FastEthernet0/0

ip address 192.168.10.1 255.255.255.0

no shutdown exit



### Step 4: Configure PCs for DHCP

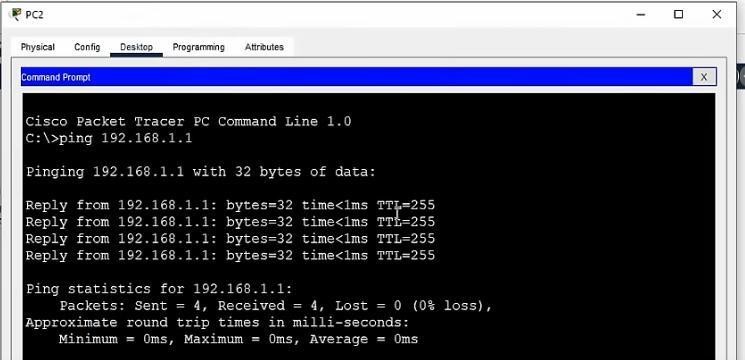
Click PC0 → Desktop → **IP Configuration** → choose

**DHCP** Do the same for PC1, PC2, etc.

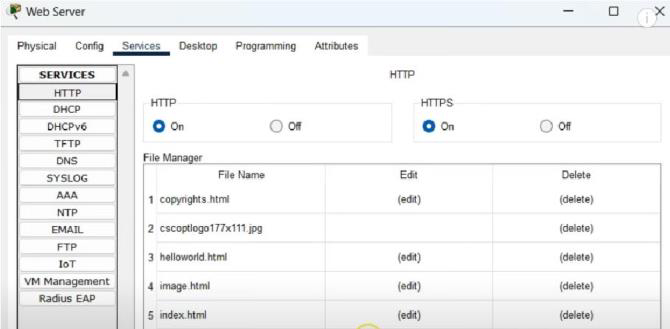
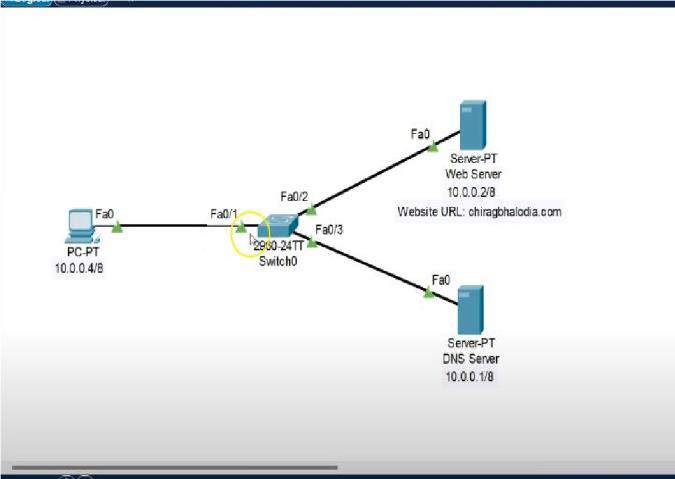
### Step 5: Verify

On each PC, after clicking DHCP:

* It should auto-fill with an IP like 192.168.10.100+
* Subnet Mask: 255.255.255.0
* Gateway: 192.168.10.1 Now test it: ping 192.168.10.1 # router ping 192.168.10.101 # other PC



## To configure a local DNS server to resolve domain names within a network. (Using packet Tracer)



### Step 1: Connections

Use **straight-through cables**:

* + PC0, PC1 → Switch
  + DNS Server → Switch
  + Switch → Router (Fa0/0)

### Step 2: Assign Static IPs

Use the **192.168.10.0/24** network:

### Device IP Address Notes

Router (Fa0/0) 192.168.10.1 Default Gateway DNS Server 192.168.10.2 DNS service runs here PC0 DHCP or Static Gets DNS from config PC1 Same as PC0



### Step 3: Configure the DNS Server

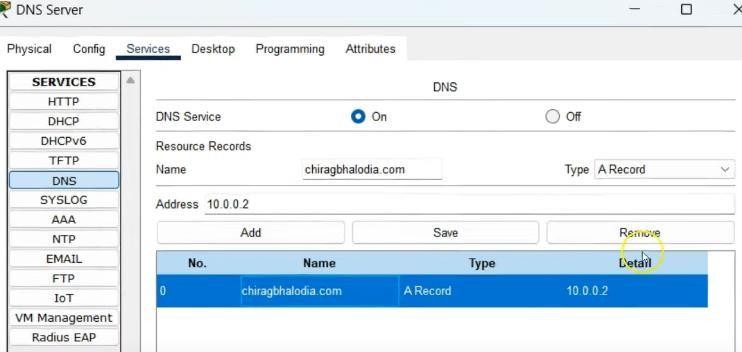
Click the Server → **Config tab** → Set static IP (as above) Then:

* + Go to **Services tab** → Select **DNS**
  + Turn DNS service **ON**
  + Add DNS records (name-to-IP mappings):

### Name Address

ayushgod.local 192.168.10.100

packettracer.local 192.168.10.101



### Step 4: Configure PC to Use the DNS Server

Click PC0 → Desktop → IP Configuration: IP Address: 192.168.10.10

Subnet Mask: 255.255.255.0

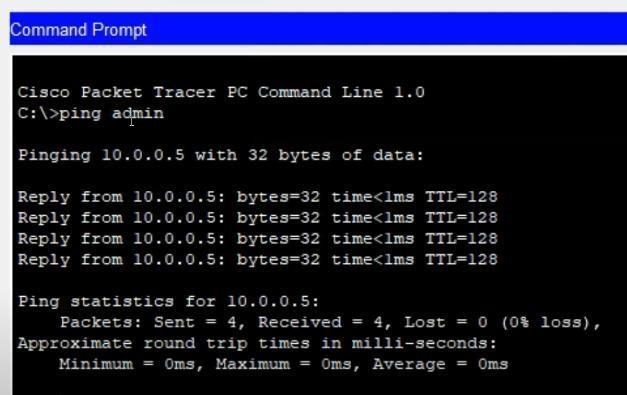
Gateway: 192.168.10.1

DNS Server: 192.168.10.2

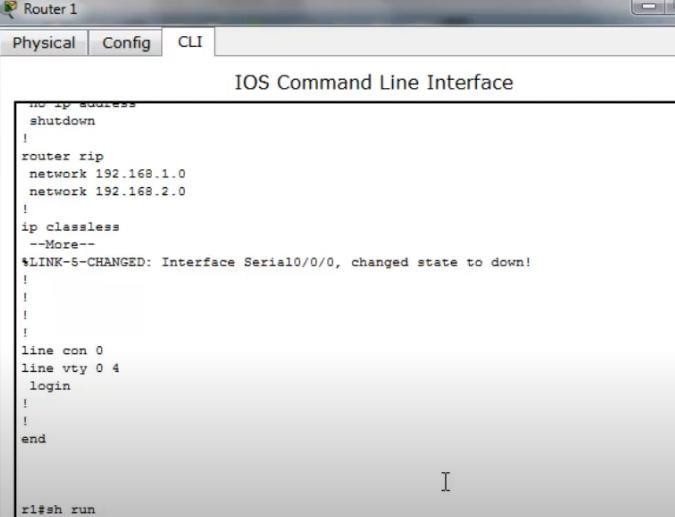
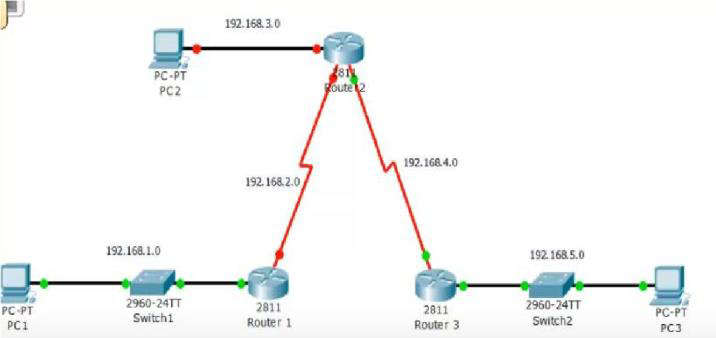
(Repeat for PC1 or use DHCP)

Step 5: Test DNS Resolution

Open PC0 → Desktop → Command Prompt: ping ayushgod.local



## Network Troubleshooting: Simulate network issues such as connectivity problems, incorrect configurations, or routing failures. Use Packet Tracer's simulation mode to diagnose and troubleshoot the network.



**Step 1: Inject Some Mistakes** Let's deliberately break some stuff:

### Issue Type Problem Introduced

IP Misconfig Set wrong IP on PC1 (192.168.30.10) Wrong Gateway Set PC0's gateway to 192.168.10.254 Cable Cut Disconnect Fa0/1 to LAN2

Interface Shutdown shutdown Fa0/0 on router

Routing Failure No routes configured on multi-router setup

### Step 2: Simulation Mode to Sniff Packets

How to Use:

1. Click Simulation Mode (bottom right in Packet Tracer).
2. Click Add Simple PDU (lightning bolt with a +).
3. Click PC0 → PC1.
4. See the packet move hop-by-hop.
5. When it fails — click the red X → analyze the problem in the Event List.

### Common Diagnoses & Fixes:

Issue #1: PC IP Misconfig

Symptom: PC1 not pingable, can't reach network. Fix:

* + Click PC1 → Desktop → IP Config → Fix IP: IP Address: 192.168.20.10

Subnet: 255.255.255.0

Gateway: 192.168.20.1

Issue #2: Wrong Gateway on PC0

Symptom: PC0 can't reach PC1 even though IP is correct. Fix:

* + Set correct gateway: Gateway: 192.168.10.1

Issue #3: Cable unplugged / Interface Down

Symptom: Link light is red or off, packet dies at router. Fix:

* + Check physical connections.
  + On Router CLI: enable configure terminal interface FastEthernet0/1 no shutdown Issue #4: No Routing Between

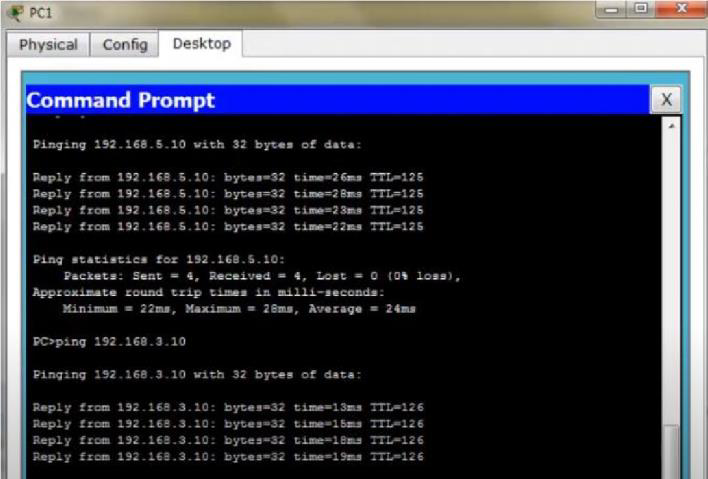
Networks If you’re using 2 routers, you need routing. Static Routing Fix:

RouterA(config)# ip route 192.168.20.0 255.255.255.0 [next-hop] RouterB(config)# ip route 192.168.10.0 255.255.255.0 [next-hop]

### Step 3: Verify Fix

After each fix:

* + Go back to Simulation Mode
  + Re-send PDU
  + Make sure the packet reaches the destination
  + Green ✔ means success



## NAT (Network Address Translation): Set up NAT on a router to translate private IP addresses to public IP addresses for outbound internet connectivity. Test the translation and examine how NAT helps conserve IPv4 address space. (Using packet Tracer)

### Objective:

* Set up NAT on a router to translate private IPs to public IPs.
* Test the translation using ping.
* Understand how NAT helps conserve IPv4 addresses.

### Step 1: Create Topology

1. **Place devices**: 2 PCs, 1 Router (NAT), 1 Server (simulating public server), 1 Cloud (optional).

### Connect using copper cables:

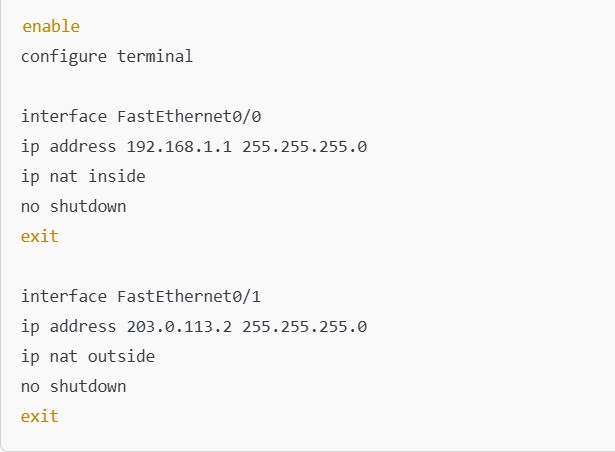
* + PC0 & PC1 → Router (FastEthernet0/0)
  + Router (Serial or FastEthernet0/1) → Server0 or ISP Router

### Step 2: Assign IP Addresses Step 3: Configure IPs

**PCs:**

* On each PC > Desktop > IP Configuration Router

Enter CLI and run:



Step 4: Configure NAT

access-list 1 permit 192.168.1.0 0.0.0.255

ip nat inside source list 1 interface FastEthernet0/1 overload Step 5: Configure Routing

ip route 0.0.0.0 0.0.0.0 203.0.113.1 ! Assuming 203.0.113.1 is the ISP or next hop

### Step 6: Test NAT

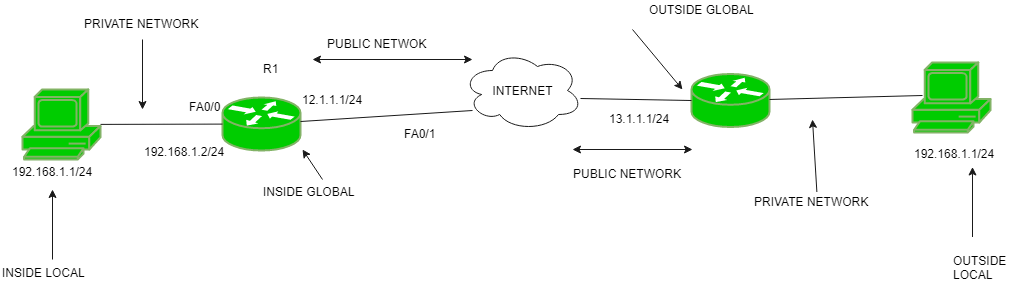
1. From PC0/PC1, go to **Command Prompt** > ping 203.0.113.3 (Server).
2. You should receive replies.

### Step 7: View NAT Translations

On Router CLI:

show ip nat translations You’ll see mappings like:

Inside global: 203.0.113.2 Inside local: 192.168.1.10



1. **TCP Client-Server Communication:**

**Implement a TCP client program that sends a message to a TCP server program.**

**Implement the corresponding TCP server program that receives the message and displays it.**

**Test the communication between the client and server by exchanging messages**

**(Using ‘C’ Language)**

**Tools Required:**

A C compiler (e.g., GCC)

Run on Linux/Unix or Windows (using MinGW or Code::Blocks)

## TCP Server Code (C Language)

// File: tcp\_server.c #include <stdio.h> #include <stdlib.h> #include <string.h>

#include <unistd.h> // for close() #include <netinet/in.h> // for sockaddr\_in

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

char buffer[BUFFER\_SIZE] = {0}; struct sockaddr\_in address;

int addrlen = sizeof(address);

// 1. Create socket

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0); if (server\_fd == 0) {

perror("Socket failed"); exit(EXIT\_FAILURE);

}

// 2. Bind socket to IP/Port address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY; // Accept connections from any IP address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr\*)&address, sizeof(address)) < 0) { perror("Bind failed");

exit(EXIT\_FAILURE);

}

// 3. Listen

if (listen(server\_fd, 3) < 0) { perror("Listen failed"); exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d...\n", PORT);

// 4. Accept connection

new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen); if (new\_socket < 0) {

perror("Accept failed"); exit(EXIT\_FAILURE);

}

// 5. Read message from client read(new\_socket, buffer, BUFFER\_SIZE); printf("Message from client: %s\n", buffer);

// Optional: Send a reply

char \*reply = "Hello from server!"; send(new\_socket, reply, strlen(reply), 0);

// 6. Close sockets close(new\_socket); close(server\_fd);

return 0;

}

## TCP Client Code (C Language)

// File: tcp\_client.c #include <stdio.h> #include <stdlib.h> #include <string.h>

#include <unistd.h> // for close()

#include <arpa/inet.h> // for sockaddr\_in, inet\_addr #define PORT 8080

#define BUFFER\_SIZE 1024 int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char \*message = "Hello from client!"; char buffer[BUFFER\_SIZE] = {0};

// 1. Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0); if (sock < 0) {

printf("\n Socket creation error \n"); return -1;

}

// 2. Set server address serv\_addr.sin\_family = AF\_INET; serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 addresses from text to binary

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) { printf("\nInvalid address/Address not supported \n");

return -1;

}

// 3. Connect to server

if (connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) { printf("\nConnection Failed \n");

return -1;

}

// 4. Send message to server

send(sock, message, strlen(message), 0); printf("Message sent to server: %s\n", message);

// 5. Read server reply

read(sock, buffer, BUFFER\_SIZE); printf("Message from server: %s\n", buffer);

// 6. Close socket close(sock); return 0;

}

## How to Compile & Run

On Linux Terminal:

gcc tcp\_server.c -o server gcc tcp\_client.c -o client

## Run in two separate terminals: Terminal 1 (Start Server):

./server

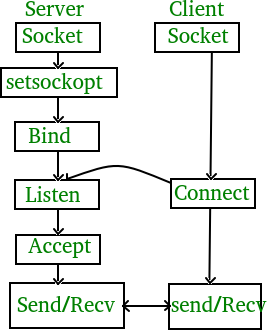
## Terminal 2 (Start Client):

./client

## Output Example Server:

Server is listening on port 8080... Message from client: Hello from client! **Client:**

Message sent to server: Hello from client! Message from server: Hello from server!



1. **UDP Client-Server Communication:**

**Implement a UDP client program that sends a message to a UDP server program.**

**Implement the corresponding UDP server program that receives the message and displays it**

**(Using ‘C’ Language)**

**Objective:**

Implement a **UDP client** that sends a message. Implement a **UDP server** that receives and displays it. Use **sockets** and **Datagram communication**.

## Step 1: UDP Server Code (udp\_server.c)

// File: udp\_server.c #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h> #include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() { int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in servaddr, cliaddr;

// 1. Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0); if (sockfd < 0) {

perror("Socket creation failed"); exit(EXIT\_FAILURE);

}

// 2. Define server address memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

servaddr.sin\_family = AF\_INET; // IPv4 servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Any IP servaddr.sin\_port = htons(PORT); // Port

// 3. Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) { perror("Bind failed");

close(sockfd); exit(EXIT\_FAILURE);

}

printf("UDP Server is running on port %d...\n", PORT);

// 4. Receive message int len, n;

len = sizeof(cliaddr);

n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*) &cliaddr, &len); buffer[n] = '\0';

// 5. Display message

printf("Message from client: %s\n", buffer);

// Optional: Send reply

char \*reply = "Message received!";

sendto(sockfd, reply, strlen(reply), 0, (struct sockaddr \*) &cliaddr, len);

// 6. Close socket close(sockfd);

return 0;

}

## Step 2: UDP Client Code (udp\_client.c)

// File: udp\_client.c #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h> #include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() { int sockfd;

char buffer[BUFFER\_SIZE];

char \*message = "Hello from client!"; struct sockaddr\_in servaddr;

// 1. Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0); if (sockfd < 0) {

perror("Socket creation failed"); exit(EXIT\_FAILURE);

}

// 2. Define server address memset(&servaddr, 0, sizeof(servaddr)); servaddr.sin\_family = AF\_INET; servaddr.sin\_port = htons(PORT);

servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// 3. Send message to server

sendto(sockfd, message, strlen(message), 0, (const struct sockaddr \*) &servaddr, sizeof(servaddr));

printf("Message sent to server: %s\n", message);

// 4. Receive reply from server int len, n;

len = sizeof(servaddr);

n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*) &servaddr, &len);

buffer[n] = '\0';

printf("Message from server: %s\n", buffer);

// 5. Close socket close(sockfd);

return 0;

}

## Step 3: Compile and Run

gcc udp\_server.c -o udp\_server gcc udp\_client.c -o udp\_client

## Run in two terminal windows:

### Terminal 1:

./udp\_server

### Terminal 2:

./udp\_client

### Output Example

**Server Output:**

UDP Server is running on port 8080... Message from client: Hello from client! **Client Output:**

Message sent to server: Hello from client! Message from server: Message received!

