CN Lab Mid Notes

CN Lab 01

🔷 Basic Definitions

Term	Definition
Network	Group/system of interconnected devices.
Computer Network	Allows computers (nodes) to share data/resources using cables or wireless links.
Host	Any device (PC, printer, server) on a network with a unique IP.
Topology	Physical/logical arrangement of network devices and links.

🔷 Common Network Topologies

Topology	Advantages	Disadvantages
Fully Connected	High security, alternate paths, simultaneous transfers	Expensive, complex, hard to scale
Bus	Cheap, simple, scalable	Fails easily, poor performance, hard troubleshooting
Mesh	Fault tolerant, alternate paths	Costly, complex maintenance
Star	Easy install/maintenance, scalable	Central failure point, extra cables
Ring	Equal access, simple setup	One failure breaks network, limited scalability

Applications:

- Mesh WiFi, Cloud Providers, ISPs, Corporates.

🔷 RJ45 & Cable Types

Type	**Used For**	**Notes**
Straight Cable	Different devices (PC→Switch, Router→Modem)	Same color order both ends
Crossover Cable	Same devices (PC→PC, Switch→Switch)	TX/RX swapped
Auto MDI/MDI-X	Automatically adjusts; crossover not needed	

🔷 Key Network Devices

Device	**Function**	**Example**
Hub	Broadcasts data to all; no filtering	Old small office networks
Switch	Sends data to specific MAC; reduces collisions	Modern LANs
Router	Forwards packets between networks	Home/ISP router

Switch vs Hub: Switch = intelligent, Hub = broadcast-all.

🔷 Common Terminologies

Term	Meaning
NIC	Hardware that connects host to network
MAC Address	Physical device address (e.g. 00:C0:9F:9B:D5:46)
IP Address	Logical address for host identification
Port	Application identifier (0–65535)
Gateway	Router address to reach external networks
DNS	Converts domain names ↔ IP addresses
DHCP	Automatically assigns IPs dynamically

🔷 OSI Model (7 Layers)

1. **Physical** – Bits, cables
2. **Data Link** – MAC, switches
3. **Network** – IP, routers
4. **Transport** – TCP/UDP, ports
5. **Session** – Connection control
6. **Presentation** – Encryption, compression
7. **Application** – HTTP, FTP, DNS

Common Commands

Linux	Windows	Function
ifconfig	ipconfig	View IP configuration
hostname	hostname	Show system hostname
nslookup	nslookup	DNS info lookup
ping	ping	Test connectivity
traceroute	tracert	Trace path to destination
netstat	netstat	Show active connections/ports

IP Address Classes

Class	Range	Default Mask	Network ID	Host ID	Notes
A	1.0.0.0 – 126.0.0.0	255.0.0.0 (/8)	1st octet	Last 3	Large networks
B	128.0.0.0 – 191.255.0.0	255.255.0.0 (/16)	1st 2	Last 2	Medium networks
C	192.0.0.0 – 223.255.255.0	255.255.255.0 (/24)	1st 3	Last 1	Small networks
D	224–239	–	–	–	Multicasting
E	240–255	–	–	–	Experimental

Private IP Ranges:

- Class A: 10.0.0.0 – 10.255.255.255
- Class B: 172.16.0.0 – 172.31.255.255
- Class C: 192.168.0.0 – 192.168.255.255

🔷 Subnet Masks (Default)

Class	Mask	Example
A	255.0.0.0	10.10.10.10
B	255.255.0.0	150.10.15.0
C	255.255.255.0	192.14.2.0

Example:

192.168.1.10 /24 → Network: 192.168.1.0, Host Range: .1–.254

🔷 Duplex Communication

Type	Description	Example
Half Duplex	Two-way but one at a time	Walkie-talkie, Hub
Full Duplex	Simultaneous two-way	Telephone, Switch

🔷 Sample Windows Commands (Lab Tasks)

Task	Command	Example Output
IP Address	ipconfig	IPv4: 172.16.13.78
Detailed Info	ipconfig /all	Shows DHCP, MAC, Gateway
Hostname	hostname	Lab2-u27
Ping	ping 8.8.8.8 / ping google.com	Connectivity test
Open Ports	netstat -an	Lists listening ports
Path Trace	tracert www.google.com	Shows 11 hops

🔷 Common Troubleshooting

Problem	Likely Cause
Ping fails by name but works by IP	DNS resolution issue
No connectivity	Bad cable, disabled adapter, wrong gateway
DHCP fails	Server off, static IP set, range exhausted

🔷 MAC vs IP

Aspect	MAC	IP
Level	Data Link	Network
Type	Physical (permanent)	Logical (assignable)
Example	00:C0:9F:9B:D5:46	192.168.0.1
Scope	Local Network	Global Routing

CN **LAB 02 -** Cisco Packet Tracer, Switch vs Hub, DHCP, and Basic LAN Setup

---

🔷 1. Cisco Packet Tracer Overview

Developed by: Cisco Systems (Dennis Frezzo team)
Modes:

- **Realtime Mode:** Normal operation
- **Simulation Mode:** Visualize packet movement

Protocols Supported:

Layer	**Example Protocols**
2	Ethernet, PPP
3	IP, ARP, ICMP
4	TCP, UDP
Routing	RIP, OSPF

🔷 2. Creating a Basic Topology

Step-by-Step

1. **Launch** Packet Tracer → *New File*
2. **Add Devices:**
   - PCs, Switches, Routers, or Hubs from bottom toolbar
3. **Connect Devices:**
   - **PC ↔ Switch:** Straight-through
   - **Switch ↔ Switch:** Crossover
   - **PC ↔ PC:** Crossover
4. **Assign IPs:**
   - PC → Config → FastEthernet
   - Example:
   - IP: 192.168.10.2
   - Mask: 255.255.255.0
   - Gateway: 192.168.10.1
5. **Test Connectivity:**
   - Desktop → Command Prompt → ping <other IP>
   - Or use **Add Simple PDU Tool (Ping)**

🔷 3. Understanding Devices (Functionality)

Device	Function
PC0–PC3	End devices, each with unique IP in 192.168.10.x subnet
Server0	Centralized services; Static IP 192.168.10.100
Switch0	Connects PC0, PC1, Server0; forwards frames using MAC; uplink to Switch1
Switch1	Connects PC2 & PC3; extends LAN by linking with Switch0 (same broadcast domain)

🔷 4. Hub vs Switch

Hub

- Works at **Layer 1 (Physical)**
- Broadcasts all traffic to every port (no filtering)
- Causes **collisions** and unnecessary traffic
- One **collision + broadcast domain** for all devices
- **Example:**
- A sends file to B via Hub
- Hub sends it to B, C, D, E
- Only B uses it; others discard it.

Switch

- Works at **Layer 2 (Data Link)**
- Forwards frames using **MAC table** (learned dynamically)
- Each port = **separate collision domain**
- Reduces congestion and improves efficiency
- **Example:**
- A sends file to B via Switch
- Switch checks MAC table → sends only to B

🔷 5. ARP (Address Resolution Protocol)

Used to map **IP** → **MAC**.

Process Example

With a Switch:

1. A wants to send file to B.
2. A knows B's IP but not MAC → broadcasts ARP: *"Who has IP B?"*
3. B replies with its MAC.
4. Switch learns B's MAC and saves it in its **MAC table**.
5. Next time, Switch forwards directly to B → Efficient delivery.

With a Hub:

1. A sends ARP request → Hub broadcasts to all.
2. B replies → Hub sends response to all ports again.
3. Every device receives unnecessary traffic → Inefficient.

✅ **In short:**

- ARP resolves IP–MAC.
- Switch = direct delivery.
- Hub = broadcasts to all.

🔷 6. DHCP Configuration (Dynamic Host Configuration Protocol)

Purpose: Automatically assign IPs to clients.

Setup Example

1. Add **1 Server + 3 PCs + 1 Switch**
2. Server → Config → FastEthernet
3. IP: 192.168.20.1
4. Mask: 255.255.255.0
5. Server → Services → DHCP
6. Pool Name: LabPool
7. Default Gateway: 192.168.20.1
8. DNS Server: 8.8.8.8
9. Start IP: 192.168.20.10
10. End IP: 192.168.20.50
11. Each PC → Desktop → IP Configuration → Select **DHCP**
12. PC receives automatic IP → verify with ping.

✅ Task Verification:

All PCs get IPs dynamically and can communicate with Server.

🔷 7. Bridge Operation

- Works at **Layer 2 (Data Link)**
- Connects multiple network segments (e.g., Hub0 ↔ Hub1)
- Each hub = single broadcast domain → Bridge separates them
- Learns MACs of connected devices → forwards only where needed
- **Result:** Reduced collisions and better performance

Task Example:

- 8 PCs connected to 2 Hubs via 1 Bridge
- Bridge filters traffic between domains efficiently.

🔷 9. STP (Spanning Tree Protocol)

- Used in **Switches** to avoid loops.
- Ports show **amber (listening/learning)** → then **green (forwarding)** after ~30 sec.

🔷 10. Simulation Mode

Use to visualize packet flow

1. Change to *Simulation Mode*
2. Select **ICMP only** filter
3. Use *Add Simple PDU Tool* → Ping PC0 → PC3
4. Press *Capture/Forward* to observe frames (e.g., ARP → Echo → Reply)

🖧 Computer Networks – Lab 03 – Socket Programming

🧩 Socket Programming Overview

Sockets enable communication between processes over a network — locally or across continents. They are endpoints of a bidirectional communication channel.

🔷 Common Socket Types

Type	Description
TCP (SOCK_STREAM)	Reliable, connection-oriented (used for most client-server apps).
UDP (SOCK_DGRAM)	Unreliable, connectionless, used for faster data transfer.

🧠 Steps to Establish a TCP Connection

1. **Server** creates socket and binds to an address/port.
2. **Server** listens for incoming client connections.
3. **Client** creates socket and connects to the server.
4. **Connection established** → both sides can send/receive data.
5. **Communication** continues until one closes the socket.

📃 Socket Function Summary

Category	Function	Description
Server	s.bind()	Binds address (host, port) to socket.
	s.listen()	Starts TCP listener.
	s.accept()	Waits for and accepts client connection.
Client	s.connect()	Connects to server socket.
General	s.send()	Sends data to socket.
	s.recv()	Receives data from socket.
	s.close()	Closes the connection.
	socket.gethostname()	Returns local host name.

🧮 Socket Creation Syntax

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

🔍 Useful Python Socket Functions

Function	Description
socket.gethostbyname(hostname)	Get IP from hostname.
socket.gethostbyaddr(ip)	Get hostname from IP.
socket.getservbyport(port, 'tcp')	Get service name for port.

🧩 Task 01 – Two-Way Chat Application

Goal: Client and Server exchange messages until one types exit.

🖥️ Server

```
import socket  # Import socket module to enable network communication

# Create a TCP socket (default: IPv4, TCP)
s = socket.socket()
# Bind the socket to localhost (same machine) and port 12345
s.bind(("localhost", 12345))
# Start listening for incoming connections (1 = max queued clients)
s.listen(1)
print("Waiting for connections...")

# Accept a client connection → returns new socket (c) and client
address (addr)
c, addr = s.accept()
print("Got connection from ", addr)
# Continuous chat loop
while True:
    # Receive message from client (max 1024 bytes) and decode from
bytes to string
    clientMessage = c.recv(1024).decode()
    # If client disconnects or sends 'exit', end the chat
    if not clientMessage or clientMessage.lower() == "exit":
        print("Client ended the chat")
        break

    # Display client's message
    print("Client: ", clientMessage)

    # Take input from server user
    serverMessage = input("You (server): ")

    # Send server's message to client (encode to bytes)
    c.send(serverMessage.encode("utf-8"))

    # If server types 'exit', close chat from server side
    if serverMessage.lower() == "exit":
        print("You (server) closed the chat")
        break
# Close the client connection and main server socket
c.close()
s.close()
print("Chat closed")
```

💻 Client

```python
import socket  # Import socket module for network communication

# Create a TCP socket (IPv4, TCP by default)
c = socket.socket()

# Connect to the server running on localhost (same system) at port 12345
c.connect(("localhost", 12345))

print("Connected to server. Type 'exit' to quit")

# Continuous chat loop
while True:
    # Take message input from the client user
    message = input("You (client): ")

    # Send the message to the server (encode string to bytes)
    c.send(message.encode("utf-8"))

    # If client types 'exit', close the chat from client side
    if message.lower() == "exit":
        print("You (client) ended the chat")
        break

    # Receive reply from server (max 1024 bytes) and decode it
    reply = c.recv(1024).decode()

    # If server closes the chat or sends 'exit', end loop
    if not reply or reply.lower() == "exit":
        print("Server ended the chat")
        break

    # Display server's reply
    print("Server reply:", reply)

# Close client socket connection
c.close()
print("Chat closed")
```

🧮 Task 02 – Factorial Service

Goal: Server calculates factorial of number sent by client.

🖥 Server

```python
import socket
import math

s = socket.socket()
print("Socket created")

s.bind(('localhost', 9000))
s.listen(5)
print("Waiting for connection...")

while True:
    c, addr = s.accept()
    print("Got connection from", addr)

    data = c.recv(1024).decode()
    print("Number received from the client: ", data)

    try:
        num = int(data)
        result = math.factorial(num)
        response = "Factorial of " + str(num) + " is " + str(result)
    except ValueError:
        response = "Input is invalid. Please enter an integer"

    c.send(response.encode('utf-8'))
    c.close()
    break
```

💻 Client

```python
import socket

s = socket.socket()
s.connect(('localhost', 9000))

number = input("Enter number whose factorial is to be found: ")
s.send(number.encode('utf-8'))

print(s.recv(1024).decode())
s.close()
```

📁 Task 03 – File Transmission

Goal: Send and receive files (no chat).

🖥 Server

```python
import socket
import os

s = socket.socket()
s.bind(('localhost', 9000))
s.listen(5)
print("Waiting for connection...")

while True:
    c, addr = s.accept()
    print("Got connection from", addr)

    filename = c.recv(1024).decode()
    print("Filename received from client:", filename)
    c.send("OK".encode())

    with open("received_" + filename, "wb") as f:
        while True:
            data = c.recv(1024)
            if data == b"EOF":
                break
            f.write(data)

    print("File received and saved as received_" + filename)

    if os.path.exists("received_" + filename):
        c.send("READY".encode())
        with open("received_" + filename, "rb") as f:
            while True:
                data = f.read(1024)
                if not data:
                    break
                c.send(data)
        print("File sent back to client")
    else:
        c.send("NOFILE".encode())

    c.close()
    break
```

💻 Client

```python
import socket
import os

s = socket.socket()
s.connect(('localhost', 9000))

filename = input("Enter file name to send: ")
if not os.path.exists(filename):
    print("File does not exist")
else:
    s.send(filename.encode())
    if s.recv(1024).decode() == "OK":
        with open(filename, "rb") as f:
            while True:
                data = f.read(1024)
                if not data:
                    break
                s.send(data)
        s.send(b"EOF")
        print("File sent to server")

        status = s.recv(1024).decode()
        if status == "READY":
            with open("downloaded_" + filename, "wb") as f:
                while True:
                    data = s.recv(1024)
                    if not data:
                        break
                    f.write(data)
            print("File received back as downloaded_" + filename)
        elif status == "NOFILE":
            print("No file sent back")

s.close()
```

🔡 Task 04 – Vowel Finder

Goal: Server returns vowels found in a word sent by client.

🖥️ Server

```python
import socket

s = socket.socket()
s.bind(('localhost', 9000))
s.listen(1)

print("Waiting for connection...")

c, addr = s.accept()
print("Got connection from", addr)

word = c.recv(1024).decode()
print("Word received from client:", word)

vowels = "aeiouAEIOU"
found = [ch for ch in word if ch in vowels]

if found:
    response = "Vowels found: " + ", ".join(found)
else:
    response = "No vowels found in the given word"

c.send(response.encode('utf-8'))

c.close()
s.close()
```

💻 Client

```python
import socket

s = socket.socket()
s.connect(('localhost', 9000))

word = input("Enter a word: ")
s.send(word.encode('utf-8'))

print("Server response:", s.recv(1024).decode())

s.close()
```

💬 Task 05 – Chat + File Transfer

Goal: Chat and send files in same session.

🖥 Server

```
import socket
import os
import threading

exit_flag = False

def handle_receive(conn):
    global exit_flag
    try:
        while True:
            data = conn.recv(1024).decode()
            if not data:
                break
            if data.lower() == "exit":
                print("Client ended chat")
                exit_flag = True
                break
            if data.startswith("FILE:"):
                parts = data.split(":")
                filename = parts[1]
                filesize = int(parts[2])
                print("Receiving file: " + filename + " (" + str(filesize) + " bytes)")
                received = 0
                with open("received_" + filename, "wb") as f:
                    while received < filesize:
                        chunk = conn.recv(1024)
                        if not chunk:
                            break
                        f.write(chunk)
                        received += len(chunk)
                print("File '" + filename + "' received successfully")
            else:
                print("Client:", data)
    except:
        pass

def handle_send(conn):
    global exit_flag
    try:
        while not exit_flag:
            msg = input("You: ")
            if exit_flag:
                break
            if msg.lower() == "exit":
                conn.send(msg.encode())
                print("Closing connection...")
                exit_flag = True
                break
            if msg.startswith("FILE:"):
                filename = msg.split(":")[1]
                if os.path.exists(filename):
                    filesize = os.path.getsize(filename)
                    header = "FILE:" + filename + ":" + str(filesize)
                    conn.send(header.encode())
                    with open(filename, "rb") as f:
                        chunk = f.read(1024)
                        while chunk:
                            conn.send(chunk)
                            chunk = f.read(1024)
                    print("File '" + filename + "' sent
successfully")
                else:
                    print("File not found:", filename)
            else:
                conn.send(msg.encode())
    except:
        pass
    conn.close()

s = socket.socket()
s.bind(('localhost', 9000))
s.listen(1)

print("Waiting for connection...")

conn, addr = s.accept()
print("Got connection from", addr)
print("Type messages to chat or 'FILE:filename' to send files")

recv_thread = threading.Thread(target=handle_receive, args=(conn,))
send_thread = threading.Thread(target=handle_send, args=(conn,))
recv_thread.start()
send_thread.start()
recv_thread.join()
send_thread.join()
s.close()
```

💻 Client

```
import socket
import os
import threading

exit_flag = False

def handle_receive(s):
    global exit_flag
    try:
        while True:
            data = s.recv(1024).decode()
            if not data:
                break
            if data.lower() == "exit":
                print("Server ended chat")
                exit_flag = True
                break
            if data.startswith("FILE:"):
                parts = data.split(":")
                filename = parts[1]
                filesize = int(parts[2])
                print("Receiving file: " + filename + " (" +
str(filesize) + " bytes)")
                received = 0
                with open("received_" + filename, "wb") as f:
                    while received < filesize:
                        chunk = s.recv(1024)
                        if not chunk:
                            break
                        f.write(chunk)
                        received += len(chunk)
                print("File '" + filename + "' received
successfully")
            else:
                print("Server:", data)
    except:
        pass

def handle_send(s):
    global exit_flag
    try:
        while not exit_flag:
            msg = input("You: ")
            if exit_flag:
                break
            if msg.lower() == "exit":
                s.send(msg.encode())
                print("Closing connection...")
                exit_flag = True
                break
            if msg.startswith("FILE:"):
                filename = msg.split(":")[1]
                if os.path.exists(filename):
                    filesize = os.path.getsize(filename)
                    header = "FILE:" + filename + ":" + str(filesize)
                    s.send(header.encode())
                    with open(filename, "rb") as f:
                        chunk = f.read(1024)
                        while chunk:
                            s.send(chunk)
                            chunk = f.read(1024)
                    print("File '" + filename + "' sent
successfully")
                else:
                    print("File not found:", filename)
            else:
                s.send(msg.encode())
    except:
        pass
    s.close()

s = socket.socket()
s.connect(('localhost', 9000))

print("Type messages to chat or 'FILE:filename' to send files")

recv_thread = threading.Thread(target=handle_receive, args=(s,))
send_thread = threading.Thread(target=handle_send, args=(s,))
recv_thread.start()
send_thread.start()
recv_thread.join()
send_thread.join()
```

Computer Networks CL3001 – Lab 04 - HTTP/HTTPS, DNS, Routers

🔷 1. HTTP vs HTTPS

Feature	**HTTP**	**HTTPS**
URL	http://	https://
Port	80	443
Security	Unsecured	Secured (SSL/TLS)
OSI Layer	Application	Transport
Encryption	None	Present
Certificates	Not required	Required (signed or self-signed)

Key Points:

- HTTP: Client requests → server responds → page rendered.
- HTTPS: Secure communication; encrypts data; uses certificates.
- Used in banking, payments, emails, corporate websites.

🔷 2. HTTP Status Codes

Client Errors (4xx)

Code	Meaning
400	Bad Request – malformed syntax/invalid request
401	Unauthorized – authentication required/failed
403	Forbidden – server refuses valid request
404	Not Found – resource unavailable
408	Request Timeout – client did not send request in time

Server Errors (5xx)

Code	Meaning
500	Internal Server Error – generic error
501	Not Implemented – unsupported request method
502	Bad Gateway – invalid response from upstream server
503	Service Unavailable – temporary server overload/down

🔷 3. DNS (Domain Name System)

Definition:
Hierarchical naming system mapping human-readable domains → IP addresses. Essential for internet functionality.

Common Record Types:

Record	**Purpose**
A Record	Domain → IPv4 address
CNAME Record	Alias → canonical name (multiple systems share IP)
NS Record	Authoritative DNS server for a domain
SOA Record	Start of Authority; zone server/admin/version info

DNS Functions:

- Resolves domain names to IPs
- Delegates authority for subdomains
- Provides fault tolerance

🔷 4. HTTP/HTTPS Lab Implementation (Packet Tracer)

HTTP Steps:

1. Assign IPs to PCs and server.
2. Enable HTTP on server (port 80).
3. PC → Web Browser → Server IP/domain → capture HTTP in simulation mode.
4. Outbound PDU shows HTTP details.

HTTPS Steps:

1. Enable HTTPS on server (port 443).
2. PC → Web Browser → Server IP → capture HTTPS packets.
3. Outbound PDU shows encrypted HTTPS details.

Tip: HTTPS encrypts data; HTTP is plain text.

🔷 6. Routers

Definition: Layer 3 devices connecting multiple networks and forwarding packets based on routing tables.

Features:

- Operates at **Network Layer**
- Contains CPU, memory, I/O interfaces, OS (Cisco IOS/DD-WRT)
- Routing table → Destination, Next hop, Interface

Basic CLI IP Assignment Example:

```
Router> enable

Router# configure terminal

Router(config)# interface fastEthernet 0/0

Router(config-if)# ip address 192.168.1.1 255.255.255.0

Router(config-if)# no shutdown
```

🔷 8. HTTP Headers & Caching

- **Header Groups:** General, Request, Response, Entity
- **Caching Headers:**
  - Age → time since response generated
  - Expires → when resource becomes stale

CN Lab 05 – SMTP & FTP

---

1. SMTP (Simple Mail Transfer Protocol)

- **Purpose:** Sending emails between servers & clients.
- **Ports:**
  - Server-to-server: **25**
  - Client-to-server submission: **587** (465 deprecated, sometimes still used)
- **Transport:** TCP
- **Retrieval protocols:** POP3 or IMAP
- **Cisco Packet Tracer Steps:**

1. Enable **SMTP & POP3** on Mail Server.
2. Set domain (e.g., fast.com).
3. Add users:

Username	Password
CS	123
EE	456
BBA	789

4. Configure email on PCs (Desktop → Email) → Save → Send/Receive.
5. Use **Simulation Mode** → Filters → SMTP & POP3 → Capture/Forward to observe headers.

---

2. FTP (File Transfer Protocol)

- **Purpose:** Transfer files between client and server.
- **Port:** 21
- **Transport:** TCP
- **Modes:**
  - **Active:** Server connects to client on client's specified port (PORT M)
  - **Passive:** Client connects to server's provided port (PASV) (useful behind firewall)
- **Security:** FTPS (SSL/TLS), SFTP (SSH)
- **Packet Tracer Steps:**

1. Enable FTP service on server → Add user accounts:

Username	**Password**	**Permissions**
Fast	1234	Read, Write, List

2. From PC command prompt:
   - Connect: ftp <server_ip> → Login
   - Upload: put test.bin
   - List files: dir
3. Simulation Mode → Filters → FTP → Capture/Forward to view FTP headers.

DNS

- **Purpose:** Translate hostnames ↔ IP addresses
- **Tools:**
  - nslookup <hostname> → resolve IP
  - nslookup -type=NS <domain> → find authoritative DNS servers
  - ipconfig /displaydns → show cached records
  - ipconfig /flushdns → clear cache

TCP

- **Key concepts:**
  - **Three-way handshake:** SYN → SYN/ACK → ACK
  - **Sequence & Ack numbers** → reliability
  - **Congestion control:** Slow start, congestion avoidance
  - **Flow control:** Receiver-advertised window

Quick Commands

- **FTP on PC:** ftp <server_ip> → user <username> → put <file> → dir
- **nslookup:** nslookup www.example.com
- **ipconfig:** ipconfig /all, ipconfig /displaydns, ipconfig /flushdns

CN Lab 06(A) – Telnet & SSH

---

1. Telnet Configuration

Switch Configuration Example

```
Switch> enable

Switch# configure terminal


# Assign IP to VLAN1

Switch(config)# interface vlan 1

Switch(config-if)# ip address 06.66.1.1 255.0.0.0

Switch(config-if)# no shutdown

Switch(config-if)# exit


# Set hostname

Switch(config)# hostname Switch0-Telnet


# Set enable password

Switch0-Telnet(config)# enable password cisco
```

```
# Set username/password for login
Switch0-Telnet(config)# username admin password cisco

# Configure VTY lines for Telnet
Switch0-Telnet(config)# line vty 0 4
Switch0-Telnet(config-line)# login local
Switch0-Telnet(config-line)# transport input telnet
Switch0-Telnet(config-line)# exit

# Save configuration
Switch0-Telnet# write
```

PC Telnet Test

```
C:\> telnet 06.66.3.1
Username: admin
Password: cisco
Switch2-Telnet> enable
Password: cisco
Switch2-Telnet#
```

---

2. SSH Configuration

Router Configuration Example

```
Router> enable
Router# configure terminal

# Set hostname
Router(config)# hostname Router1-SSH

# Configure interfaces
Router1-SSH(config)# interface fastEthernet0/0
Router1-SSH(config-if)# ip address 06.66.1.1 255.255.255.0
Router1-SSH(config-if)# no shutdown
Router1-SSH(config-if)# exit

Router1-SSH(config)# interface fastEthernet0/1
Router1-SSH(config-if)# ip address 06.67.2.1 255.255.255.0
Router1-SSH(config-if)# no shutdown
Router1-SSH(config-if)# exit
```

```
# Set domain & generate RSA keys
Router1-SSH(config)# ip domain-name mynet.com
Router1-SSH(config)# crypto key generate rsa
# choose 1024 bits

# Set SSH username & secret
Router1-SSH(config)# username admin secret admin123

# Enable SSH on VTY lines
Router1-SSH(config)# line vty 0 4
Router1-SSH(config-line)# login local
Router1-SSH(config-line)# transport input ssh
Router1-SSH(config-line)# exit

# Set SSH version
Router1-SSH(config)# ip ssh version 2

# Set enable password
Router1-SSH(config)# enable password 123

# Save configuration
Router1-SSH# write
```

PC SSH Test

```
C:\> ssh -l admin 06.67.2.1
```

```
Password: admin123

Router1-SSH> enable

Password: 123

Router1-SSH#
```

---

3. Switch IP Update via Telnet

```
Switch2-Telnet> enable

Password: cisco

Switch2-Telnet# configure terminal

Switch2-Telnet(config)# interface vlan 1

Switch2-Telnet(config-if)# ip address 06.66.3.5 255.0.0.0

Switch2-Telnet(config-if)# exit

Switch2-Telnet(config)# write
```

---

4. Useful Show Commands

```
# Show running configuration

Switch# show running-config


# Show IP addresses

Switch# show ip interface brief

Router# show ip route
```

---

5. Key Notes

- **Telnet:** Port 23, unencrypted, login using username/password.

- **SSH:** Port 22, encrypted, login using username/password or key-based authentication.
- **VTY Lines:** Determines max simultaneous remote sessions (0-4 or 0-15).
- Always write or do wr to save configuration.
- Use no shutdown on all interfaces for connectivity.
- Ping between devices to verify IP configuration.

CN Computer Networks – Lab 06(B)/07 ACL

---

1. ACL Basics

- **ACL (Access Control List):** Set of filtering rules applied to interfaces to control traffic.
- **Types:**
  - **Standard ACL:** Filters by **source IP only**
    - Numbers **1–99**, **1300–1999**
    - Usually applied **near destination**
- **Implicit Deny:** Unmatched packets are automatically dropped.
- **Sequence Order:** First matching rule is applied.
- **Numbered vs Named ACLs:**
  - *Numbered* → cannot remove a single entry
  - *Named* → allows editing/deleting specific entries

---

2. Standard Numbered ACL – Syntax & Example

Syntax:

```
access-list <1-99> permit|deny <source-ip> <wildcard-mask>
```

Wildcard Masks:

Purpose	Wildcard
Exact match	0.0.0.0
Ignore all bits	255.255.255.255

Example – Permit Two Hosts, Deny All Others

```
Router> enable

Router# configure terminal

Router(config)# access-list 10 permit host 192.168.10.10

Router(config)# access-list 10 permit host 192.168.10.20

Router(config)# access-list 10 deny any

Router(config)# interface fa0/0

Router(config-if)# ip access-group 10 in

Router(config-if)# do write
```

Verification:

```
Router# show access-lists

Standard IP access list 10

10 permit host 192.168.10.10

20 permit host 192.168.10.20

30 deny any
```

---

3. Standard Named ACL – Syntax & Examples

Purpose: Same filtering as Standard ACL but allows editing specific entries.

Syntax:

```
Router(config)# ip access-list standard <ACL-NAME>

Router(config-std-nacl)# permit|deny <source-ip> <wildcard-mask>

Router(config-std-nacl)# exit

Router(config)# interface <intf>

Router(config-if)# ip access-group <ACL-NAME> in|out
```

Example 1 – BLOCK_PC1 (Deny one host)

```
Router> enable

Router# configure terminal

Router(config)# ip access-list standard BLOCK_PC1

Router(config-std-nacl)# deny 192.168.10.10 0.0.0.0

Router(config-std-nacl)# permit any

Router(config-std-nacl)# exit

Router(config)# interface fastEthernet0/1

Router(config-if)# ip access-group BLOCK_PC1 out

Router(config-if)# exit

Router(config)# do write
```

Example 2 – BLOCK_RANGE (Deny range of hosts)

```
Router> enable

Router# configure terminal

Router(config)# ip access-list standard BLOCK_RANGE

Router(config-std-nacl)# deny 192.168.10.0 0.0.0.255

Router(config-std-nacl)# permit any

Router(config-std-nacl)# exit

Router(config)# interface fastEthernet0/1

Router(config-if)# ip access-group BLOCK_RANGE out

Router(config-if)# exit

Router(config)# do write
```

To remove named ACL:

```
Router(config)# no ip access-list standard BLOCK_PC1
```

To edit an existing named ACL:

```
Router(config)# ip access-list standard BLOCK_PC1

Router(config-std-nacl)# no deny 192.168.10.10 0.0.0.0

Router(config-std-nacl)# deny 192.168.10.20 0.0.0.0

Router(config-std-nacl)# exit
```

4. Applying ACLs on Interfaces

```
Router(config)# interface <type/number>

Router(config-if)# ip access-group <ACL-number/name> in

Router(config-if)# ip access-group <ACL-number/name> out
```

- **Standard ACL → near destination**

---

5. Common Lab Tasks

Task	**ACL Type**	**Example**
Block single host	Standard	access-list 10 deny host 192.168.1.10
Permit only selected hosts	Standard	access-list 10 permit host 192.168.1.11
Block single host (Named)	Named	ip access-list standard BLOCK_PC1deny 192.168.10.10 0.0.0.0permit any
Block subnet (Named)	Named	ip access-list standard BLOCK_RANGEdeny 192.168.10.0 0.0.0.255permit any

---

6. Verification Commands

```
# Show ACL rules

Router# show access-lists



# Show ACLs applied on interfaces

Router# show running-config

# Test ACLs

PC> ping <target-ip>

PC> ssh <server-ip>

PC> http://<server-ip>
```