Experiment 01 – Basic Network Commands

AIM: To study and execute basic network commands (ping, tracert, nslookup, pathping, router config).

THEORY: Basic network commands help in diagnosing connectivity, routing paths, DNS lookup and network configuration. Ping uses ICMP echo requests to test host reachability. Tracert traces route hops. Nslookup queries DNS servers. Pathping combines ping and tracert to measure loss and latency. Router configuration commands allow interface setup, mode switching, and saving configurations.

APPARATUS / TOOLS REQUIRED: Command Prompt, Cisco Packet Tracer, Router Simulator.

STEPS TO PERFORM: 1. Open Command Prompt. 2. Run commands: ping, tracert, nslookup, pathping. Note outputs. 3. Enter router simulation mode (Packet Tracer). 4. Use: enable ’n

configure terminal ’n interface setup. 5. Show running-config and startup-config. 6. Save config safely.

ADDITIONAL NOTES: All basic diagnostic and configuration commands were understood and executed successfully.

CONCLUSION: Router commands require privileged mode. 'show ip route' displays routing table. '?' shows available commands

Experiment 02 – Packet Capture Using Wireshark

AIM: To capture ICMP packets generated by ping and analyze header details.

THEORY: Wireshark is a packet sniffer that displays packet headers at each OSI layer. ICMP is

used for diagnostic messages. Echo Request (Type 8) and Echo Reply (Type 0) help verify network connectivity. The experiment involves capturing packets in real-time.

APPARATUS / TOOLS REQUIRED: Wireshark, Internet Connection, PC

STEPS TO PERFORM: 1. Start Wireshark. 2. Select active interface. 3. Start capture. 4. Run 'ping google.com'. 5. Apply filter: icmp. 6. Examine Ethernet, IP and ICMP headers. 7. Stop capture and save file.

ADDITIONAL NOTES: ICMP traffic was captured, filtered and analyzed successfully with protocol-level insights.

CONCLUSION: Important fields: Source/Destination IP, ICMP Type/Code, Identifier, Sequence number.

Experiment 03 – TCP Three-Way Handshake

AIM: To observe SYN, SYN-ACK, ACK packets during connection establishment.

THEORY: TCP is a reliable connection-oriented protocol that uses a 3-step handshake. The client sends SYN, server replies with SYN_ACK, and client finalizes with ACK. This ensures synchronization of sequence numbers.

APPARATUS / TOOLS REQUIRED: Wireshark, Browser, Internet

STEPS TO PERFORM: 1. Start Wireshark and begin capture. 2. Open browser ‘n http://example.com. 3. Stop capture. 4. Filter using 'tcp'. 5. Identify SYN, SYN_ACK, ACK packets. 6. Inspect sequence/ack numbers. 7. Calculate RTT by time reference.

ADDITIONAL NOTES: The TCP handshake was recorded and analyzed successfully.

CONCLUSION: Handshake diagrams normally show arrows Client’n Server with SYN bits set.

Experiment 04 – Network Topologies

AIM: To demonstrate different topologies (Bus, Star, Ring, Mesh, Tree) in Packet Tracer.

THEORY: Network topology defines how devices are interconnected. Bus uses a single backbone, Star has central switch, Ring forms a closed loop, Mesh connects all devices pairwise, and Tree combines star and bus. Transmission media include copper, fiber, and wireless.

APPARATUS / TOOLS REQUIRED: Packet Tracer, PCs, Switches, Cables

STEPS TO PERFORM: 1. Open Packet Tracer. 2. Create Bus (line), Star (switch center), Ring (loop), Mesh (fully connected), Tree (hierarchical). 3. Use appropriate cables—straight-through, cross-over, fiber. 4. Add wireless AP for WLAN. 5. Test using ping. 6. Observe simulation mode.
ADDITIONAL NOTES: Topologies were created, connected and tested successfully.

CONCLUSION: Topology diagrams include linear bus, star with hub, closed ring loops, and full mesh node interconnects.

Experiment 05 – Error Detection and Correction

(CRC & Hamming Code)

AIM: To implement error detection and correction techniques using CRC and Hamming Code.

APPARATUS / TOOLS REQUIRED: JDK Compiler, Wireshark, Linux/Windows PC.

THEORY: CRC detects errors by polynomial division of the received frame; if remainder is zero, frame is correct. Hamming Code performs single bit error correction using parity bits placed at power of two positions. It detects 2-bit errors and corrects 1-bit errors. The experiment demonstrates encoding, syndrome generation, and error correction as shown in the PDF

STEPS TO PERFORM: 1. Input data bits. 2. For CRC ‘n append zeros, divide using generator polynomial, compute FCS. 3. For Hamming ‘n place parity bits at positions 1,2,4,8… 4. Compute P1, P2, P4 from parity coverage. 5. Transmit encoded bits. 6. Receiver recomputes parity to form a syndrome. 7. If syndrome ≠ 0, bit at that position is flipped. 8. Verify corrected output.

CONCLUSION: CRC detected errors using remainder method. Hamming Code detected and corrected single bit errors successfully

Experiment 06 – Sliding Window Protocols (GBN & SR)

AIM: To simulate Go-Back-N and Selective Repeat ARQ protocols.

THEORY: Sliding window improves channel efficiency by allowing multiple frames in transit. GBN retransmits all frames after an error; Selective Repeat retransmits only the lost/damaged frame. Windows move as ACKs are received.

APPARATUS / TOOLS REQUIRED: PC, GCC/JDK, Wireshark.

STEPS TO PERFORM: 1. Run provided program. 2. Enter number of frames. 3. Simulate random loss. 4. Observe GBN: retransmission from error frame onward. 5. Observe SR: only missing frame resent. 6. Compare bandwidth efficiency.

ADDITIONAL NOTES: Both ARQ mechanisms were simulated and behavior compared successfully.

CONCLUSION: GBN uses cumulative ACKs, SR requires buffering and individual ACKs.

Experiment 07 – Distance Vector Routing

AIM: To compute shortest paths using Bellman-Ford distance vector algorithm.

THEORY: Routers exchange distance vectors periodically. Each router updates routes using the Bellman-Ford equation: $D(x,y)=\min\{C(x,v)+D(v,y)\}$. This continues until convergence forming routing tables.

APPARATUS / TOOLS REQUIRED: GCC Compiler, Wireshark, Linux/Windows.

STEPS TO PERFORM: 1. Run routing program. 2. Enter number of nodes. 3. Enter link costs (999 for no link). 4. View initial routing table. 5. Allow iterative DV updates. 6. Query shortest paths. 7. Note routing tables for each node.

ADDITIONAL NOTES: Shortest paths and routing tables were generated successfully.

CONCLUSION: Routing table diagrams typically include Destination, Cost, Next Hop.

Experiment 08 – DNS Client-Server

AIM: To implement DNS lookup using socket programming.

THEORY: DNS resolves domain names to IPs using a distributed hierarchical system. UDP port 53 is used for standard queries. The experiment sets up a custom DNS server with static hostname_IP mappings.

APPARATUS / TOOLS REQUIRED: Python, Terminal, PC.

STEPS TO PERFORM: 1. Run dns_server.py. 2. Run dns_client.py. 3. Enter hostname/IP. 4. Server checks dictionary. 5. Client prints resolved address. 6. Exit using 'exit'.

ADDITIONAL NOTES: A working DNS client-server demonstration was achieved.

CONCLUSION: DNS resolution involves query ’n lookup ’n response cycle.