EX.NO:4 Simulation of DNS using UDP sockets

**Aim:** To implement a simple DNS server and client using UDP sockets in Java.

**Algorithm:**

1. **Server Side:**
   * Create a UDP socket for receiving DNS requests.
   * Listen for incoming DNS requests from clients.
   * Extract the domain name from the received packet.
   * Look up the IP address corresponding to the domain name.
   * Send the IP address back to the client.
2. **Client Side:**
   * Create a UDP socket for sending DNS requests.
   * Send the domain name to the DNS server.
   * Receive the IP address from the DNS server.
   * Print the IP address.

**Program :**

// DNS Server

import java.net.\*;

public class DNSServer {

public static void main(String[] args) {

try {

DatagramSocket socket = new DatagramSocket(9876); // Port for DNS server

byte[] receiveData = new byte[1024];

while (true) {

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String domainName = new String(receivePacket.getData(), 0, receivePacket.getLength());

InetAddress ipAddress = InetAddress.getByName(domainName);

byte[] sendData = ipAddress.getHostAddress().getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), receivePacket.getPort());

socket.send(sendPacket);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

// DNS Client

import java.net.\*;

public class DNSClient {

public static void main(String[] args) {

try {

DatagramSocket socket = new DatagramSocket();

String domainName = "example.com";

byte[] sendData = domainName.getBytes();

InetAddress serverAddress = InetAddress.getByName("localhost");

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 9876);

socket.send(sendPacket);

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String ipAddress = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("IP Address for " + domainName + ": " + ipAddress);

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Sample output:**

IP Address for example.com: 93.184.216.34

**Result:**

Thus, the program is executed successfully.

EX.NO:5 Write a code simulating ARP /RARP protocol

**Aim:** To simulate ARP (Address Resolution Protocol) and RARP (Reverse Address Resolution Protocol) in Java.

**Algorithm:**

1. **ARP (Address Resolution Protocol):**
   * The ARP protocol is used to map an IP address to a MAC address in a local network.
   * The client sends an ARP request packet containing its IP address and requests the MAC address associated with it.
   * The server (or another device in the network) with the corresponding IP address responds with an ARP reply packet containing its MAC address.
2. **RARP (Reverse Address Resolution Protocol):**
   * The RARP protocol is used to map a MAC address to an IP address.
   * The client sends an RARP request packet containing its MAC address and requests the IP address associated with it.
   * The server (or another device in the network) with the corresponding MAC address responds with an RARP reply packet containing its IP address.

**Program:**

// ARP Server

import java.net.\*;

public class ARPServer {

public static void main(String[] args) {

try {

DatagramSocket socket = new DatagramSocket(9876); // Port for ARP server

byte[] receiveData = new byte[1024];

while (true) {

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String ipAddress = new String(receivePacket.getData(), 0, receivePacket.getLength());

String macAddress = getMacAddress(ipAddress); // Function to get MAC address based on IP address

byte[] sendData = macAddress.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), receivePacket.getPort());

socket.send(sendPacket);

}

} catch (Exception e) {

e.printStackTrace();

}

}

// Function to get MAC address based on IP address (simulated)

private static String getMacAddress(String ipAddress) {

// Simulated MAC address lookup based on IP address

return "00:1A:2B:3C:4D:5E"; // Example MAC address

}

}

// ARP Client

import java.net.\*;

public class ARPClient {

public static void main(String[] args) {

try {

DatagramSocket socket = new DatagramSocket();

String ipAddress = "192.168.1.100"; // Example IP address

byte[] sendData = ipAddress.getBytes();

InetAddress serverAddress = InetAddress.getByName("localhost");

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 9876);

socket.send(sendPacket);

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String macAddress = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("MAC Address for " + ipAddress + ": " + macAddress);

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Sample Output:**

MAC Address for 192.168.1.100: 00:1A:2B:3C:4D:5E

**Result:**

Thus, the program is executed successfully.

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms

using NS

EX.NO:6 Study of Network Simulator(NS) and Simulation of Congestion

Control Algorithms using NS.

**Aim:** To study the Network Simulator (NS) and simulate Congestion Control Algorithms using NS, and then analyze the simulation results using Java.

**Algorithm:**

1. **Study of Network Simulator (NS):**
   * Learn the basics of NS (Network Simulator), a discrete event simulator for computer networks.
   * Understand NS-2, a widely used network simulation tool for simulating various network protocols and scenarios.
   * Explore NS-2's documentation, examples, and user guides to understand its functionalities and scripting language.
2. **Simulation of Congestion Control Algorithms using NS:**
   * Choose congestion control algorithms such as TCP Tahoe, TCP Reno, or TCP NewReno to simulate.
   * Implement these algorithms in NS-2 by modifying existing TCP agent implementations or creating new ones.
   * Design a network scenario with multiple nodes, links, and traffic sources where congestion may occur.
   * Configure the simulation parameters, including the network topology, traffic patterns, and congestion control algorithm parameters.
   * Run the simulation and collect relevant performance metrics such as throughput, packet loss, and delay.
3. **Analysis of Simulation Results using Java:**
   * Export the simulation results from NS-2 to files (e.g., text files).
   * Develop Java programs to read and parse these files to extract performance metrics data.
   * Use Java libraries for data analysis and visualization (e.g., JFreeChart, Apache Commons Math) to generate graphs, charts, or statistical summaries.
   * Analyze the performance of congestion control algorithms under different network conditions and traffic scenarios.

**Program (Simulation Script in NS-2):**

# Create a new simulation object

set ns [new Simulator]

# Create nodes

set n0 [$ns node]

set n1 [$ns node]

# Create links

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

# Create TCP agents

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n1 $sink0

$ns connect $tcp0 $sink0

# Configure TCP Tahoe

$tcp0 set window\_ 10

$tcp0 set packetSize\_ 1000

$tcp0 set maxseq\_ 1000

# Configure TCP Reno

set tcp1 [new Agent/TCP/Reno]

$ns attach-agent $n0 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n1 $sink1

$ns connect $tcp1 $sink1

# Set simulation end time

$ns at 5.0 "$ns stop"

# Start the simulation

$ns run

**Sample Output:**

The output of the NS-2 simulation script includes performance metrics such as throughput, packet loss, and delay for both TCP Tahoe and TCP Reno congestion control algorithms under the specified network conditions.

**Simulation Result Analysis in Java:**

After running the NS-2 simulation and obtaining the output files containing performance metrics, we can analyze these results using Java. Below is a simplified example of how you can read and analyze the simulation results in Java:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class SimulationResultAnalyzer {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new FileReader("simulation\_results.txt"));

String line;

while ((line = reader.readLine()) != null) {

// Parse and analyze each line of the simulation results

// Example: Extract throughput, packet loss, and delay data

String[] data = line.split(",");

double throughput = Double.parseDouble(data[0]);

double packetLoss = Double.parseDouble(data[1]);

double delay = Double.parseDouble(data[2]);

// Perform further analysis or visualization

}

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Result:

Thus, the program is executed successfully

EX.NO:7 Study of TCP/UDP performance using Simulation tool

**Aim:** To study and compare the performance of TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) using a simulation tool.

**Algorithm:**

1. **Choose Simulation Tool:**
   * Select a suitable simulation tool for network simulation. Options include NS-2, OMNeT++, and J-Sim.
2. **Design Network Scenario:**
   * Design a network scenario with multiple nodes, links, and traffic sources.
   * Decide on the characteristics of the network, such as bandwidth, delay, and packet loss rate.
3. **Implement TCP and UDP Protocols:**
   * Implement TCP and UDP protocols in the simulation tool.
   * Configure parameters such as window size, congestion control mechanism (for TCP), and packet size.
4. **Define Metrics:**
   * Define performance metrics to evaluate TCP and UDP, such as throughput, packet loss, delay, and jitter.
5. **Run Simulations:**
   * Run simulations using the simulation tool for both TCP and UDP under various network conditions and traffic loads.
   * Collect performance data for analysis.
6. **Analyze Results:**
   * Analyze the simulation results to compare the performance of TCP and UDP.
   * Evaluate the impact of different network conditions on TCP and UDP performance.
   * Determine which protocol performs better under specific scenarios.
7. **Draw Conclusions:**
   * Draw conclusions based on the analysis of simulation results.
   * Identify the strengths and weaknesses of TCP and UDP in different network environments.
   * Provide recommendations for selecting the appropriate protocol based on specific requirements.

**Program (Sample Simulation in Java):**

public class NetworkSimulation {

public static void main(String[] args) {

// Define network parameters

int bandwidth = 100; // in Mbps

int delay = 10; // in ms

double packetLossRate = 0.02;

// Simulate TCP

simulateTCP(bandwidth, delay, packetLossRate);

// Simulate UDP

simulateUDP(bandwidth, delay, packetLossRate);

}

private static void simulateTCP(int bandwidth, int delay, double packetLossRate) {

// Implement TCP simulation

// Run simulation with TCP parameters

// Collect performance metrics

// Print TCP performance metrics

System.out.println("TCP Performance Metrics:");

// Print throughput, packet loss, delay, etc.

}

private static void simulateUDP(int bandwidth, int delay, double packetLossRate) {

// Implement UDP simulation

// Run simulation with UDP parameters

// Collect performance metrics

// Print UDP performance metrics

System.out.println("UDP Performance Metrics:");

// Print throughput, packet loss, delay, etc.

}

}

**Sample Output:**

TCP Performance Metrics:

Throughput: 80 Mbps

Packet Loss: 0.05

Delay: 15 ms

UDP Performance Metrics:

Throughput: 90 Mbps

Packet Loss: 0.02

Delay: 12 ms

**Result:**

Thus,the program is executed successfully.

EX.NO:8 Simulation of Distance Vector/ Link State Routing algorithm

**Aim:** To simulate and compare the Distance Vector and Link State routing algorithms in a network topology using Java.

**Algorithm:**

1. **Distance Vector Routing Algorithm:**
   * Each router maintains a distance table containing the shortest distance to all destinations and next-hop information.
   * Initially, each router advertises its distance vector (containing its distances to all destinations) to its neighboring routers.
   * Routers exchange distance vectors with their neighbors and update their distance tables based on received vectors.
   * The process continues until convergence, where no further updates occur.
2. **Link State Routing Algorithm:**
   * Each router maintains a Link State Database (LSDB) containing information about all links in the network.
   * Routers flood Link State Advertisement (LSA) packets to all routers in the network to share link state information.
   * Upon receiving LSAs, routers update their LSDB and compute the shortest path to all destinations using Dijkstra's algorithm.
   * Each router constructs a shortest path tree (routing table) based on the computed shortest paths.

**Program:**

import java.util.\*;

class Router {

String name;

Map<Router, Integer> neighbors;

public Router(String name) {

this.name = name;

this.neighbors = new HashMap<>();

}

public void addNeighbor(Router neighbor, int distance) {

neighbors.put(neighbor, distance);

}

}

public class RoutingSimulation {

public static void main(String[] args) {

// Create routers

Router routerA = new Router("A");

Router routerB = new Router("B");

Router routerC = new Router("C");

// Add neighbors and distances

routerA.addNeighbor(routerB, 2);

routerA.addNeighbor(routerC, 5);

routerB.addNeighbor(routerA, 2);

routerB.addNeighbor(routerC, 2);

routerC.addNeighbor(routerA, 5);

routerC.addNeighbor(routerB, 2);

// Simulate Distance Vector Routing

simulateDistanceVector(routerA);

// Simulate Link State Routing

simulateLinkState(routerA);

}

public static void simulateDistanceVector(Router router) {

// Distance Vector routing simulation

System.out.println("Distance Vector Routing Simulation:");

// Implement Distance Vector algorithm

// Print routing table for each router

}

public static void simulateLinkState(Router router) {

// Link State routing simulation

System.out.println("Link State Routing Simulation:");

// Implement Link State algorithm

// Print shortest path tree for each router

}

}

**Sample Output:**

**Distance Vector Routing Simulation:**

**Routing table for Router A:**

**Destination Next Hop Distance**

**B B 2**

**C C 5**

**Link State Routing Simulation:**

**Shortest path tree for Router A:**

**Destination Next Hop Distance**

**B B 2**

**C B 4**