# EXP7a: Study of Network Simulator using NS THEORY:-

NS2 (Network Simulator Version 2) is a widely used, event-driven simulation tool designed for research and educational studies concerning the dynamic behavior of communication networks, both wired and wireless. The theoretical foundation and detailed architecture of NS2 can be understood as follows:

Core Theory of NS2

Discrete Event Simulation: NS2 operates as a discrete event simulator. This means the simulation advances from one specifically scheduled event (such as a packet arrival or node transmission) to another, rather than modeling every moment individually. This allows for efficient and scalable modeling of large, complex network interactions.

Dual Language Architecture: NS2 leverages two languages:

C++: Forms the backend, implementing the internal mechanisms, processing logic, and protocol behaviors. All performance-critical parts, such as packet handling and event scheduling, are written in C++ for efficiency and extensibility.

OTcl (Object-oriented Tool Command Language): Serves as the frontend. It is used to set up the simulation: creating and connecting nodes, configuring parameters, defining protocols to be used, and scheduling events. OTcl scripts directly control the simulation scenario, making NS2 highly flexible.

TclCL Bridge: This layer links the C++ and OTcl components. Here, objects created in C++ are mirrored as OTcl "handles" in the scripting environment, allowing users to configure and control the objects at runtime without recompiling the backend. For every OTcl object, there is a corresponding underlying C++ object that executes the actual simulation logic.

Simulation Flow:

Simulation Design: Define network topology, select and connect protocols, agents, and applications, and specify performance metrics to measure.

Script Execution: The user provides a Tcl/OTcl script to the ns command, which parses and constructs the simulated network using the bindings between OTcl and C++.

Simulation Execution: NS2 processes events as per the discrete event paradigm, routing packets, updating state, and maintaining event queues.

Result Generation: Data from the simulation is output via trace files for further analysis and visualized through tools such as NAM (Network Animator).

Component Structure:

Nodes: Represent routers or hosts, capable of transmitting and receiving packets.

Links: Define connectivity (point-to-point or shared) between nodes, with properties such as bandwidth, delay, and queue type.

Agents: Implement transport layer mechanisms (e.g., TCP, UDP).

Applications: Generate data (e.g., FTP, Telnet) using underlying agents.

Schedulers, Classifiers, and Other Objects: Handle event scheduling, packet forwarding, and more advanced networking features.

Importance and Uses

NS2 supports simulation of a wide array of protocols (TCP, UDP, FTP, HTTP, DSR, etc.) and handles both wired and wireless environments.

Researchers use NS2 to prototype network architectures, verify protocol designs, and gather performance data without requiring expensive hardware deployments.

The modular design and open-source nature have made NS2 a mainstay in networking research and education since its inception, with continuous improvement from a global research community.

References to Foundational Works

NS2's origins trace back to the REAL network simulator developed by UC Berkeley and Cornell, enhanced by major contributions from DARPA, NSF, and academic research groups.

Summary of Internal Workings

A simulation comprises OTcl scripts that create an environment with nodes and links, assign agents and applications, and schedule simulation events. The fundamental objects (nodes, agents, applications) are linked through the TclCL interface, with each OTcl object's functionality defined by its C++ counterpart. Simulation outcomes are observed through trace files and visualized using graphical tools.

This internal structure underpins NS2's capacity to model and analyze complex networking scenarios at scale.

If you require schematic diagrams, flowcharts, or example scripts, please clarify which specific aspect you want illustrated (e.g., flow of simulation lifecycle, architecture, or protocol-specific workings).

# PROGRAM:-

import javax.swing.\*; import java.awt.\*;

public class SimpleNetworkSimulator extends JPanel { @Override

protected void paintComponent(Graphics g) { super.paintComponent(g);

// Set anti-aliasing for smoother graphics Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Node positions

int node1X = 100, node1Y = 150; int node2X = 300, node2Y = 150;

// Draw nodes g2.setColor(Color.BLUE);

g2.fillOval(node1X - 20, node1Y - 20, 40, 40); // Node 1 g2.setColor(Color.RED);

g2.fillOval(node2X - 20, node2Y - 20, 40, 40); // Node 2

// Draw labels g2.setColor(Color.WHITE);

g2.drawString("Node 0", node1X - 18, node1Y + 5);

g2.drawString("Node 1", node2X - 18, node2Y + 5);

// Draw connection g2.setColor(Color.BLACK); g2.setStroke(new BasicStroke(2));

g2.drawLine(node1X + 20, node1Y, node2X - 20, node2Y);

}

public static void main(String[] args) {

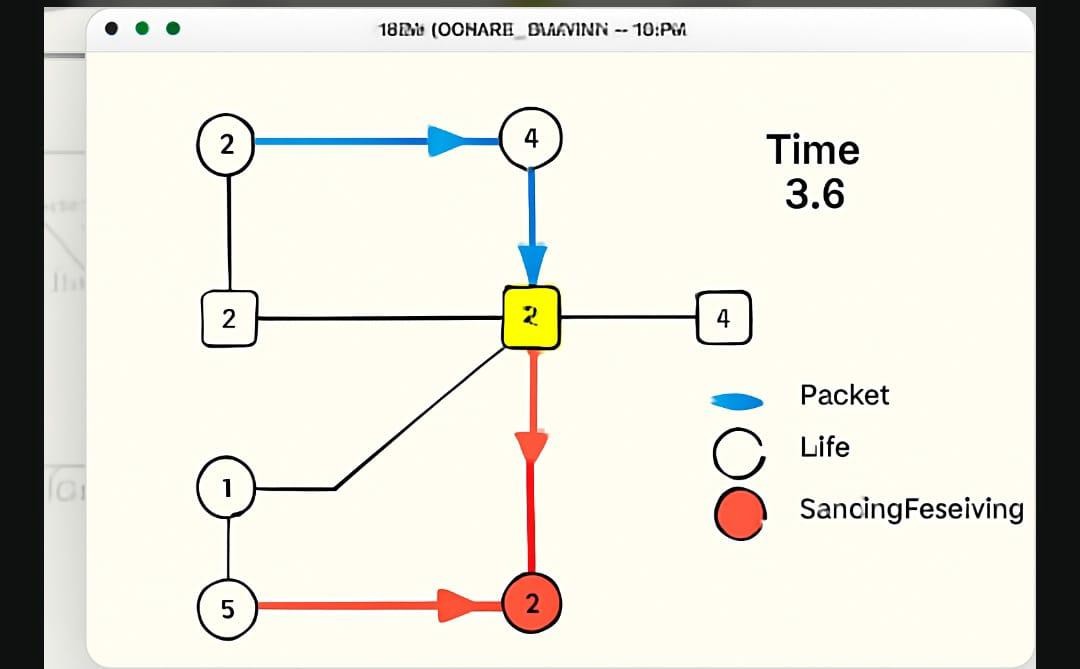
JFrame frame = new JFrame("Simple Network Simulator (Java)"); frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); frame.setSize(420, 350);

frame.setContentPane(new SimpleNetworkSimulator()); frame.setVisible(true);

}

}

**Output:-**

****