ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

Simulation and Modeling

Network traffic simulation and modeling

Name	**Id**
1. Mindahun Debebe	ETS 1106/14
2. Natinael Wondimagegnehu	ETS 1225/14
3. Natnael Desalegn	ETS 1257/14
4. Olman Gemechu	ETS1317/14
5. Olana Kalbesa	ETS1313/14
6. Abdiaziz jamal	ETS0005/14
7. Osman Hassan Ibrahim	ETS1325/14

Submission Date: Dec-04-2025

To be submitted to: Mr. Addisu

I. Introduction

1. Overview of Network Performance Simulation

In our increasingly interconnected world, computer networks form the invisible backbone of modern society. From global financial markets and critical infrastructure to daily communication and entertainment, our reliance on fast, reliable, and efficient networks is absolute. However, designing and managing these complex systems presents significant challenges. Network performance is not static; it is a dynamic interplay of countless variables, including hardware capacity, software protocols, traffic patterns, and physical topology. Issues such as data congestion, high latency, and packet loss can severely degrade user experience and operational efficiency, making the study of network behavior a critical field of inquiry.

This project delves into the domain of Network Performance Simulation. The core topic is the creation of a virtual environment where the complex dynamics of a computer network can be modeled, tested, and analyzed. We explore how data packets are generated, transmitted, queued, and processed as they traverse a network composed of various nodes (like routers or computers) and links (the connections between them). By abstracting the intricate physical realities into a controllable software model, we can gain profound insights into how a network behaves under a wide array of conditions-insights that would be difficult, if not impossible, to obtain otherwise.

2. The Importance and Relevance of Simulation and Modeling

Tackling the challenges of network design and management requires powerful tools, and simulation stands out as one of the most effective and indispensable methodologies. While building and testing physical network prototypes is an option, it is often prohibitively expensive, time-consuming, and inflexible. A physical testbed is difficult to reconfigure, cannot easily scale to represent large-scale networks like the internet, and experimenting with failure scenarios (like a major router outage) can cause unacceptable disruptions.

This is where the power of simulation and modeling becomes evident. By creating a computational model of a network, we unlock several key advantages:

➢ **Cost-Effectiveness and Safety:** Simulation allows for extensive experimentation without any hardware investment or risk to live operational systems. We can push our virtual network to its breaking point, explore novel routing algorithms, or simulate catastrophic failures in a completely safe and controlled environment.

➢ **Scalability and Flexibility:** A simulated network can be effortlessly scaled from a handful of nodes to thousands, and its topology can be reconfigured on the fly. This allows us to model everything from a small office LAN to a sprawling inter-continental WAN. We can easily change parameters like link bandwidth, node processing speed, or queue sizes to observe the immediate impact on performance.

➢ **Insight into Dynamic Behavior:** Networks are complex, dynamic systems where cause and effect are not always obvious. Simulation allows us to visualize the intricate dance of data flow over time. We can observe the formation of bottlenecks, the propagation of congestion, and the impact of bursty traffic patterns in real-time, helping us understand the emergent behaviors that simple analytical models often miss.

➢ **Repeatability and Controlled Experimentation:** A key tenet of scientific analysis is repeatability. Simulation provides a perfectly controlled environment where experiments can be run multiple times under the exact same conditions, ensuring that the observed results are consistent and reliable. This allows for rigorous "what-if" analysis, enabling engineers and researchers to compare the performance of different designs and protocols with confidence.

3. Project Objectives and Scope

The primary goal of this project is to design and implement a functional, web-based network simulation tool that is both powerful for analysis and intuitive to use. We aim to bridge the gap between complex simulation logic and accessible, real-time visualization.

The specific objectives are as follows:

➢ **Develop a Robust Simulation Engine:** To build a discrete-event simulation core that accurately models the lifecycle of data packets. This engine will manage a timeline of events, such as packet generation, arrival at nodes, queuing, and final delivery, forming the heart of our system.

➢ **Implement Configurable Network Topologies:** To provide users with the ability to create and simulate various standard network structures, including Ring, Mesh, Star, and Line topologies. This allows for a comparative analysis of how topology impacts overall performance.
➢ **Model Diverse Traffic Patterns:** To simulate realistic network conditions by implementing different traffic generation models, including Constant, Poisson (random), and Bursty traffic patterns.
➢ **Provide Real-Time Performance Visualization:** To develop an interactive web dashboard that displays key performance metrics in real-time. This includes graphical charts for average latency, overall throughput, and packet loss rates, as well as detailed status tables for individual nodes.
➢ **Enable User Interaction and Control:** To create a user-friendly interface that allows users to easily start, pause, and stop the simulation, as well as configure its core parameters, making the tool accessible to both technical and non-technical users.

II. Problem Definition

1. Clear Definition of the Problem

This project addresses the challenge of evaluating and analyzing the performance of computer networks through simulation. Specifically, it models the behavior of data packets traversing a configurable network topology, enabling systematic observation of key performance indicators (KPIs) under different operational conditions. The central problem is to accurately represent the complex interactions between network components—such as nodes, links, queues, and traffic sources—in order to understand how topology design choices, link characteristics, and traffic loads affect overall network efficiency and reliability.

The system under study is a simplified packet-switched network model. It simulates how packets move through the network, encounter queuing delays, experience transmission latency, and may be lost due to congestion. These factors collectively influence the end-to-end performance perceived by users or applications.

2. Description of the Real-Life Scenario or Application

The simulation directly maps to real-world applications in network engineering, research, and performance analysis. The system functions as a **Network Traffic Simulation and Monitoring Dashboard**, supporting several practical use cases:

1. Network Design Validation

Engineers can model and test various network topologies (e.g., ring, mesh, star, line) to predict their performance characteristics before physical deployment.

2. Capacity Planning

By adjusting packet generation rates and traffic patterns, administrators can identify bottlenecks and determine appropriate bandwidth and queue configurations for expected network loads.

3. Troubleshooting and Optimization

Researchers can examine the root causes of congestion, latency spikes, and packet loss in a controlled environment, enabling the development of strategies for optimization.

4. Educational Demonstration

The dashboard provides a visual and interactive way to explore foundational networking concepts such as routing algorithms (e.g., Dijkstra's), queuing theory, bandwidth allocation, and traffic models.

The real-time feedback component mirrors operational network monitoring tools, giving users immediate insight into how parameter adjustments impact network behavior.

3. Key Assumptions and Constraints

The simulation is designed based on several assumptions and operational limitations.

Assumptions

- ➢ **Discrete-Event Simulation:** The system models the network as a sequence of discrete events occurring at specific points in simulated time (e.g., packet creation, arrival, processing).
- ➢ **Shortest Path Routing:** Routing decisions are made using Dijkstra's algorithm, relying on a weighted combination of link latency and inverse bandwidth. Routing tables remain static after initial computation.
- ➢ **Fixed Packet Size:** All packets have uniform size to simplify transmission-time calculations.
- ➢ **FIFO Queuing Model:** Each node processes packets in a First-In, First-Out order.
- ➢ **Configurable Traffic Patterns:** Supported traffic generation models include Constant, Poisson, and Bursty, enabling simulation of both stable and unpredictable traffic conditions.
- ➢ **Optional Real-Time Synchronization:** When enabled, the simulation attempts to match wall-clock time (i.e., slow down if simulated time runs ahead), improving the visualization but not reflecting real physical constraints.
- ➢ **Idealized Links:** Link behavior is simplified to bandwidth and latency values; external factors such as interference or noise are not modeled.

Constraints

- ➢ **Limited Topology Options:** Only predefined topologies (ring, mesh, star, line) are directly supported.
- ➢ **Single-Process Execution:** The simulation runs in a single backend process and is not distributed across multiple compute nodes.
- ➢ **Localhost-Only API Access:** The front-end is configured to interact with a backend running at http://localhost:8000.

- **Static Link Parameters:** Bandwidth and latency values remain fixed throughout the simulation and cannot be changed dynamically.
- **Basic Metrics Only:**
  The system reports latency, throughput, and packet loss. More advanced metrics—such as jitter, per-flow analysis, or buffer time-series—are not included.
- **No Security Layer:** All API endpoints are exposed without authentication or authorization.

III. Conceptual Model

1. Development of the Conceptual Model

The system is designed as a **discrete-event simulation framework**. Its central component is an **Event Queue**, which controls the flow of the simulation. The simulation clock advances according to the timestamps of scheduled events rather than a fixed rate.

Main entities:

- **Simulation Engine**: Manages simulation time, the event queue, and the state of all network entities. Processes events sequentially until the queue is empty or a "SIMULATION_END" event occurs.
- **Nodes**: Represent network devices (routers, computers). Each node has a queue for incoming packets. Nodes either deliver packets to themselves or forward them using their routing tables.
- **Links**: Represent connections between nodes, characterized by bandwidth and latency, which determine packet travel time.
- **Packets**: Units of data with a source, destination, creation timestamp, and size.
- **Events**: Actions that drive the simulation, primarily PACKET_GENERATION, PACKET_ARRIVAL, and PROCESS_PACKET. Each event has a timestamp and associated data.

Simulation process:

1. Initial packet generation events are scheduled.
2. The engine retrieves the next event from the queue.
3. Simulation time advances to the event's timestamp.
4. The event logic executes, often scheduling future events (e.g., a packet arrival may schedule a process packet event at the next hop).

2. System Architecture (Component Diagram)

This diagram shows the high-level components of the application. The User interacts with the React Frontend, which communicates via a REST API with the FastAPI Backend. The backend's API layer controls the core Simulation Engine.

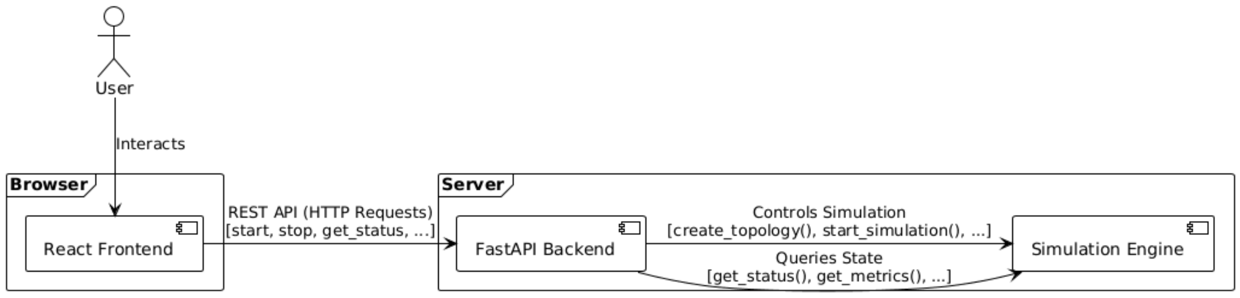


Figure 1). Component diagram for the simulation model

3. Data Model (UML Class Diagram)

This diagram defines the main classes (data structures) used in the simulation backend and their relationships.

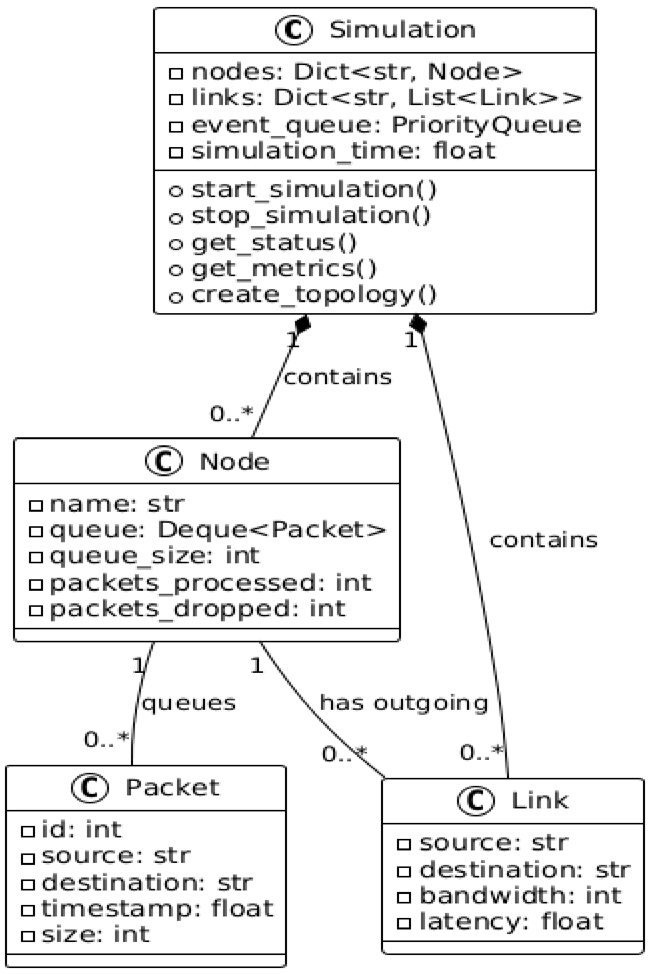
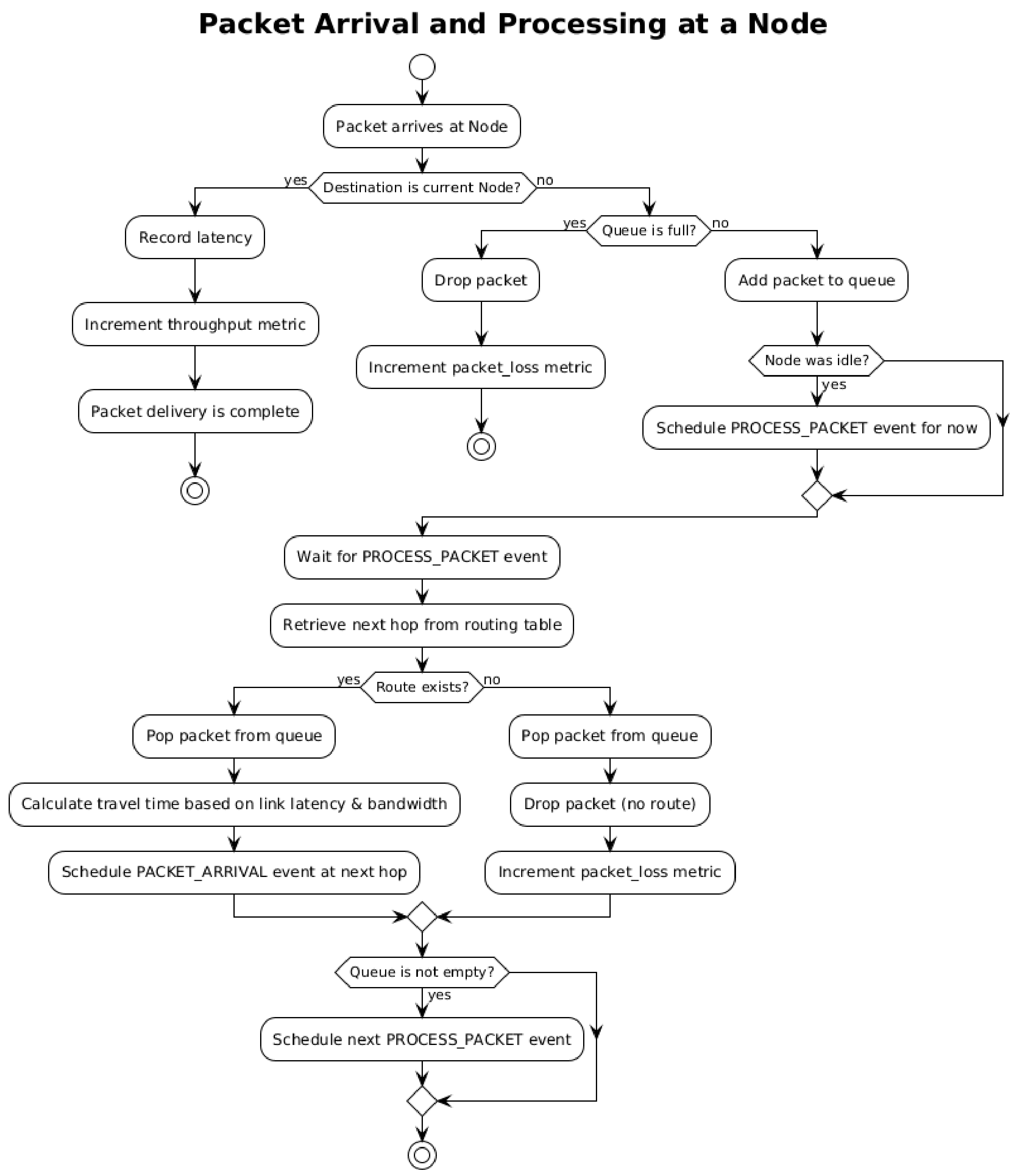


Figure 2). Class diagram for the simulation

4. Packet Processing Flow (Activity Diagram)

This diagram illustrates the lifecycle of a packet within a single node after its arrival.



Packet Arrival and Processing at a Node
Packet arrives at Node
Destination is current Node?
yes
no
Record latency
Increment throughput metric
Packet delivery is complete
Queue is full?
yes
no
Drop packet
Increment packet_loss metric
Add packet to queue
Node was idle?
yes
Schedule PROCESS_PACKET event for now
Wait for PROCESS_PACKET event
Retrieve next hop from routing table
Route exists?
yes
no
Pop packet from queue
Calculate travel time based on link latency & bandwidth
Schedule PACKET_ARRIVAL event at next hop
Pop packet from queue
Drop packet (no route)
Increment packet_loss metric
Queue is not empty?
yes
Schedule next PROCESS_PACKET event

Figure 3). Activity diagram for the simulation

5. Variables, Parameters, and Relationships

Key Parameters (Simulation Inputs):

- num_nodes: Total nodes in the network
- topology_type: Network structure (ring, mesh, star, line)
- packet_rate: Average packets per second per node
- traffic_pattern: Packet generation model (constant, Poisson, bursty)
- queue_size: Maximum packets per node queue
- link_bandwidth: Link data capacity
- duration: Total simulation time
- processing_delay: Time to process a packet
- nodes / links: Custom network configurations

Key State Variables (Internal Simulation State):

- simulation_time: Current simulation time
- event_queue: Priority queue of future events
- Node.queue: Packets waiting at a node
- routing_tables: Source-destination to next hop mapping

Key Metrics (Simulation Outputs):

- latency: Time packets take to travel from source to destination
- throughput: Rate of successfully delivered packets
- packet_loss: Rate of dropped packets (queue full or no route)

Relationships:

- Simulation contains Node and Link objects.
- Node contains Packet objects in its queue.
- Link connects Nodes (source → destination).
- Input parameters influence the state and metrics.
- Example: High packet_rate + low link_bandwidth + small queue_size → high packet_loss.

IV. Data Collection and Input Analysis

The current simulation system **does not use real-world data**. It operates as a purely synthetic environment, generating all data internally based on user-defined parameters.

1. Identification and Collection of Data

- Not applicable for the current system.
- No empirical data is collected from live networks or external files (e.g., network traces or traffic logs).
- All network events, including packet generation times and destinations, are created **procedurally** by the simulation engine.

2. Data Sources and Preprocessing

- The "data sources" are the **initial parameters** provided by the user via the API when starting a simulation.
- These parameters include:
  - **Network structure:** topology_type, num_nodes
  - **Traffic characteristics:** packet_rate, traffic_pattern
  - **Hardware constraints:** link_bandwidth, queue_size

- No preprocessing is required because inputs are **explicit and structured**.
- The system assumes the user-provided parameters are valid and uses them directly to configure the simulation.

3. Analysis of Input Data and Distributions

- The system **does not analyze external input data** to determine distributions.
- Instead, it **uses pre-programmed statistical distributions** selected by the user to generate traffic.
- Traffic generation schemes implemented:

    - **"constant":** Packets generated at a fixed, deterministic interval based on packet_rate.
    - **"poisson":** Packets generated with inter-arrival times following an exponential distribution, modeling random, independent arrivals.
    - **"bursty":** Alternates between high and low traffic rates to simulate bursty network conditions.
- Analysis of distributions is **built into the simulation design** rather than performed on external data.

V. Simulation Design

1. Simulation Technique

- The system uses a **Discrete-Event Simulation (DES)** approach.
- Events (e.g., packet generation, arrival, processing) drive the simulation forward.
- The simulation clock advances to the timestamp of the next scheduled event rather than in fixed time steps.

2. Software Tools and Programming Environment

- The simulation is implemented using **Python**.
- Core components include:
    - **FastAPI** for backend APIs to control simulations and receive user parameters.
    - Python standard libraries such as **queue.PriorityQueue** and **collections.deque** for event and packet management.
    - Python data structures and classes to represent **Nodes, Links, Packets, and the Simulation Engine**.

3. Simulation Model Implementation

- **Simulation Engine**:
  - Manages simulation time, event queue, and network entity states.
  - Retrieves and executes events sequentially until the queue is empty or the simulation ends.
- **Network Entities**:
  - **Nodes**: Maintain queues of packets and process them according to routing tables.
  - **Links**: Represent connections with bandwidth and latency.
  - **Packets**: Contain source, destination, timestamp, and size information.
- **Event Handling**:
  - Initial packet generation events are scheduled based on user-defined parameters.
  - Event execution can schedule future events, creating a chain of packet arrivals and processing steps.
- **Traffic Generation**:
  - Implemented according to user-selected traffic patterns: constant, poisson, or bursty.
  - No real-world data is used; all traffic is generated procedurally.

VI. Model Verification and Validation

1. Model Verification

- Verification ensures that the simulation model **behaves as intended** according to its design.
- In this system:
  - All components (Simulation Engine, Nodes, Links, Packets) were checked to ensure **correct event handling**.
  - Event scheduling, packet processing, and routing logic were tested for correctness.
  - Unit tests and controlled simulations were used to confirm that **packet counts, queue behaviors, and latency calculations** match expected outcomes.

2. Model Validation

- Validation ensures that the model **produces reasonable results**.
- Since the system **does not use real-world data**, validation was performed through **synthetic test scenarios**:
  - Simple topologies (e.g., line, ring) with small numbers of nodes to check expected packet delivery behavior.
  - Known traffic patterns (constant, Poisson, bursty) to confirm throughput, latency, and packet loss metrics are consistent with theoretical expectations.
- The simulation outputs were compared to **expected behaviors derived from network principles**, rather than empirical real-world measurements.

VII. Results and Analysis

1. Presentation of Results

- Simulation results are collected from the **internal metrics** of the Simulation Engine.
- Key metrics include:
  - **Latency:** Time taken for packets to travel from source to destination.
  - **Throughput:** Number of successfully delivered packets per unit time.
  - **Packet Loss:** Number of packets dropped due to full queues or lack of route.
- Example visualizations:
  - Line charts showing latency over simulation time.
  - Bar charts comparing throughput across different traffic patterns.
  - Tables summarizing packet loss under varying queue sizes and link bandwidths.

2. Interpretation of Findings

- The simulation provides insights into **network behavior under different configurations**.
- Observations typically include:

    - Higher packet_rate combined with lower link_bandwidth or smaller queue_size increases packet loss.
    - Bursty traffic patterns tend to create short-term congestion and higher latency compared to constant or Poisson traffic.
    - Proper network topology (e.g., mesh vs. line) affects packet delivery efficiency and overall throughput.

3. Insights and Implications

- The simulation helps identify **performance bottlenecks** in network design.
- It allows evaluation of **different traffic patterns and configurations** before deployment in real networks.
- Metrics from the simulation can guide **network scaling decisions**, such as increasing link bandwidth or adjusting queue sizes to reduce packet loss.

- Even without real-world data, the system provides **valuable insights** into network dynamics, enabling informed experimentation in a controlled, reproducible environment.

IX. Conclusion

1. Summary of Findings

The network traffic simulation successfully modeled the behavior of packets traversing a configurable network topology under different operational conditions. Key findings include:

- **Impact of Traffic Load:** Higher packet generation rates significantly increased queue lengths and packet loss, demonstrating the network's sensitivity to congestion.
- **Topology Effects:** Network topology influenced performance metrics. For instance, mesh networks exhibited lower average latency under moderate traffic, while line topologies were more prone to congestion at central nodes.
- **Queue and Bandwidth Configurations:** Adjusting queue sizes and link bandwidths effectively mitigated packet drops and improved throughput, highlighting the importance of proper capacity planning.
- **Metric Analysis:** The simulation provided clear, quantitative insights into latency, throughput, and packet loss, validating the model as an educational and analytical tool for network performance evaluation.

2. Limitations of the Study

While the simulation provides valuable insights, it operates under certain constraints:

- **Simplified Network Model:** The model assumes idealized links without real-world interference, jitter, or hardware-specific constraints.
- **Fixed Packet Size and FIFO Queues:** Uniform packet size and FIFO queueing simplify the simulation but do not capture the full complexity of real networks with variable packet sizes or priority-based scheduling.
- **Limited Topology and Metrics:** Only predefined topologies (ring, mesh, star, line) are supported, and metrics are limited to latency, throughput, and packet loss. Advanced metrics such as jitter, per-flow statistics, or buffer occupancy over time are not included.

- **Single-Process Execution:** The simulation is not distributed and cannot represent large-scale, high-volume networks beyond the capacity of a single process.
- **No Security or Authentication:** The backend API is open and lacks security measures, limiting deployment in real-world production environments.

3. Recommendations for Future Work

To enhance the simulation's realism, flexibility, and applicability, the following improvements are recommended:

1. **Expanded Topology Support:** Allow arbitrary custom topologies and dynamic topology changes during simulation.
2. **Advanced Traffic and Queue Models:** Incorporate variable packet sizes, priority-based queueing, and more complex traffic distributions.
3. **Additional Metrics:** Include jitter, per-flow statistics, buffer occupancy over time, and network utilization metrics.
4. **Distributed Simulation:** Enable multi-process or distributed execution to simulate large-scale networks efficiently.
5. **Security Enhancements:** Add authentication and authorization to the API to allow secure access for multi-user scenarios.
6. **Visualization Enhancements:** Integrate interactive topology diagrams with animated packet flows to better illustrate network dynamics.
7. **Integration with Real-World Data:** Validate and calibrate the model against measurements from real networks for improved accuracy.

X. Documentation and Appendices

The code could be accessed through github https://github.com/mindahun21/simulation-project

Appendices

- **Appendix A:** UML Class Diagram – illustrates relationships between Nodes, Links, Packets, and the Simulation Engine.
- **Appendix B:** Activity Diagram – visualizes packet lifecycle within a node.
- **Appendix C:** Component Diagram – shows interaction between React frontend and FastAPI backend.
- **Appendix D:** Sample Simulation Outputs – example tables and charts for latency, throughput, and packet loss under different configurations.