```
NetSim/
    src/
        core/
            Packet.cpp
            Simulator.cpp
            SimulationClock.cpp
        packet_transmission/
            PacketTransmission.cpp
        bandwidth_utilization/
            BandwidthUtilization.cpp
        error_rate/
            ErrorRateModel.cpp
        load_balancing/
            LoadBalancer.cpp
        network_congestion/
            NetworkCongestion.cpp
        network_delay/
            NetworkDelay.cpp
        topology_change/
            TopologyManager.cpp
        routing_efficiency/
            RoutingEfficiency.cpp
        packet_prioritization/
            PacketPrioritizer.cpp
        main.cpp
    include/
        SimulationClock.h
        PacketTransmission.h
        BandwidthUtilization.h
        ErrorRateModel.h
        LoadBalancer.h
        NetworkCongestion.h
        NetworkDelay.h
        TopologyManager.h
        RoutingEfficiency.h
        PacketPrioritizer.h
        Simulator.h
        Packet.h
    obj/
    bin/
    Makefile
    README.md
```

1 Overview

NetSim is a comprehensive C++ network simulation project designed to model and analyse various aspects of network behavior. It provides a modular and extensible framework for simulating complex network scenarios, including packet transmission, bandwidth utilisation, error rates, load balancing, congestion, delays, topology changes, and routing efficiency.

2 Key Features

1. **Packet Transmission Simulation**: Uses a Poisson process to model realistic packet arrivals.
2. **Bandwidth Utilization**: Simulates network bandwidth usage over time.
3. **Error Rate Modelling**: Implements complex error models considering environmental factors.
4. **Load Balancing**: Simulates distribution of network traffic across multiple paths.
5. **Network Congestion**: Models congestion levels under varying network conditions.
6. **Network Delay**: Uses exponential distribution to simulate realistic network delays.
7. **Topology Management**: Simulates dynamic changes in network structure.
8. **Routing Efficiency**: Calculates and optimises routing paths in the network.
9. **Packet Prioritization**: Implements dynamic packet prioritization based on network conditions.
10. **Network State Snapshots**: Captures and logs the network state at regular intervals during simulation.

3 Building the Project

3.1 Prerequisites

- C++17 compatible compiler (e.g., GCC 7+ or Clang 5+)
- Make build system
- POSIX-compliant operating system (Linux, macOS, etc.)

3.2 Compilation

To build the project, navigate to the project root directory and run:

```
make
```

This command will:

1. Create `obj/` and `bin/` directories if they don't exist.
2. Compile all `.cpp` files in `src/` and its subdirectories into object files in `obj/`.
3. Link all object files to create the executable `netsim` in the `bin/` directory.

3.3 Cleaning the Build

To remove all compiled files and start fresh:

```
make clean
```

This will delete the `obj/` and `bin/` directories.

4 Running the Simulation

After building, you can run the simulation using:

```
./bin/netsim [initial_node_count] [simulation_duration] [snapshot_interval]
```

4.1 Parameters

- `initial_node_count` (optional): The number of nodes to start the simulation with. Default is 10.
- `simulation_duration` (optional): The duration of the simulation in seconds. Default is 3600 (1 hour).
- `snapshot_interval` (optional): The interval between network state snapshots in seconds. Default is 300 (5 minutes).

Examples:

```
# Run with default parameters
./bin/netsim

# Run with 20 initial nodes
./bin/netsim 20

# Run with 20 initial nodes for 2 hours
./bin/netsim 20 7200

# Run with 20 initial nodes for 2 hours, taking snapshots every 2 minutes
./bin/netsim 20 7200 120
```

5 Simulation Components

5.1 SimulationClock

The SimulationClock is a central component that manages the simulation time. It is implemented as a singleton class to ensure a single, globally accessible clock for all components. Key features include:

- Precise timekeeping using `std::chrono` for high-resolution time points.
- Methods to start, stop, and advance the simulation time.
- Thread-safe implementation using atomic variables for concurrent access.
- Ability to simulate time passage at different rates compared to real-time.

5.2 PacketTransmission

The PacketTransmission component simulates the generation and transmission of network packets. It uses a Poisson process to model realistic packet arrivals. Implementation details:

- Utilizes `std::poisson_distribution` for generating inter-arrival times.
- Implements a packet generator that creates packets with varying sizes and priorities.
- Supports customizable packet size distributions (e.g., bimodal for typical internet traffic).
- Provides methods to simulate burst traffic and periodic transmissions.

5.3 BandwidthUtilization

This component tracks and analyses the usage of network bandwidth over time. Key features:

- Calculates instantaneous and average bandwidth utilisation.
- Implements sliding window algorithms for real-time bandwidth monitoring.
- Supports multiple bandwidth calculation methods (e.g., token bucket, leaky bucket).
- Provides statistics on peak usage, utilisation patterns, and bottlenecks.

5.4 ErrorRateModel

The ErrorRateModel simulates network errors based on various factors. Implementation details:

- Models bit errors using Binary Symmetric Channel (BSC) and Gilbert-Elliott models.
- Incorporates environmental factors like signal-to-noise ratio, interference, and distance.
- Simulates burst errors using Markov chain models.
- Provides methods for Forward Error Correction (FEC) and retransmission protocols.

5.5 LoadBalancer

This component implements various load balancing algorithms to distribute network traffic across multiple paths. Features include:

- Implementation of popular algorithms: Round Robin, Least Connections, Weighted Round Robin.
- Dynamic load balancing based on real-time network conditions.
- Support for heterogeneous server capabilities and health monitoring.
- Extensible design allowing easy addition of new load balancing strategies.

5.6 NetworkCongestion

The NetworkCongestion component models congestion levels under varying network conditions. It includes:

- Implementation of congestion control algorithms (e.g., TCP congestion control variants).
- Simulation of buffer overflow and packet drops in routers and switches.
- Modelling of congestion windows and slow start mechanisms.
- Analysis tools for identifying congestion hotspots and bottlenecks.

5.7 NetworkDelay

This module simulates network delays using exponential distribution and considers various factors. Key features:

- Modelling of propagation, transmission, processing, and queuing delays.
- Implementation of jitter using normal distribution.
- Simulation of delay variations based on network load and congestion.
- Support for delay-based routing and Quality of Service (QoS) implementations.

5.8 TopologyManager

The TopologyManager handles the network topology, including dynamic changes. It includes:

- Graph-based representation of network topology using adjacency lists or matrices.
- Support for various network topologies (e.g., mesh, star, tree, hybrid).
- Dynamic node and link addition/removal during simulation.
- Topology analysis tools for connectivity, centrality, and clustering coefficient calculations.
- Simulation of network failures and recovery.

5.9 RoutingEfficiency

This component calculates and optimises routing paths in the network. Implementation details:

- Implementation of routing algorithms: Dijkstra's, Bellman-Ford, A*.
- Support for dynamic routing table updates based on network changes.
- Calculation of routing efficiency metrics (e.g., path length, hop count, latency).
- Simulation of routing protocols like OSPF, BGP for realistic scenarios.

5.10 PacketPrioritizer

The PacketPrioritizer implements dynamic packet prioritization based on network conditions. Features include:

- Implementation of priority queues for packet scheduling.
- Support for multiple Quality of Service (QoS) classes.
- Dynamic priority adjustment based on packet age, size, and network congestion.
- Integration with congestion control and load balancing components for holistic traffic management.

6 Network State Snapshots

The simulation now includes a feature to capture and log the network state at regular intervals:

- Configurable snapshot interval via command-line argument.
- Comprehensive snapshot of all network components' states.
- Real-time display of snapshots during simulation.
- Logging of snapshots to a separate file (`simulation_snapshots.log`) for later analysis.

7 Customization and Extension

The modular design of NetSim allows for easy customization and extension:

1. To add a new component, create corresponding `.h` and `.cpp` files in `include/` and `src/` directories.
2. Update the `Simulator` class in `Simulator.h` and `Simulator.cpp` to integrate the new component.
3. Modify `main.cpp` if you need to add new command-line parameters for your component.

8 Output and Logging

The simulation results are logged to two files in the project root directory:

- `simulation_results.log`: Contains detailed statistics about the final network state.
- `simulation_snapshots.log`: Contains periodic snapshots of the network state throughout the simulation.

These files include information such as:

- Simulation duration
- Congestion levels
- Bandwidth utilisation
- Error rates
- Routing efficiency
- Network topology changes
- Packet prioritization statistics

9 Performance Considerations

For large-scale simulations:

1. The project uses C++17 features for improved performance.
2. Consider using parallel processing techniques for computationally intensive components.
3. Optimize data structures and algorithms in critical paths of the simulation.
4. Adjust snapshot interval for balance between detail and performance in long simulations.

10 Future Enhancements

1. Implement visualization tools for real-time simulation monitoring and snapshot analysis.
2. Develop a comprehensive test suite using Google Test or a similar framework.
3. Add support for more complex network protocols and behaviors.
4. Implement a configuration system for easy parameter adjustments without recompilation.
5. Enhance logging capabilities for more detailed analysis of simulation results.
6. Implement parallel processing for improved performance in large-scale simulations.

11 Contributing

Contributions to NetSim are welcome. Please follow these steps:

1. Fork the repository.
2. Create a new branch for your feature or bug fix.
3. Commit your changes with clear, descriptive messages.
4. Push the branch to your fork.
5. Submit a pull request with a detailed description of your changes.

12 License