Table of Contents

1. Global Imports

The simulation begins by importing a variety of Python modules and libraries. These include:

- **Standard Python Modules** such as:
  - **sys**: Handles system-level operations.
  - **logging**: Provides a framework to output diagnostic messages that help in debugging and tracking events during the simulation.
  - **random**: Used for generating random numbers, which is crucial for randomizing node behaviors, positions, timeouts, and selections.
  - **string**: Supplies common string constants and operations, for example to generate node names.
  - **math**: Offers mathematical functions (like calculating square roots) needed for computing distances between nodes.
- **SimPy**: A discrete-event simulation library. It is used to create and manage the simulation's timeline, enabling processes such as packet transmissions, node activities, and sleep cycles to be scheduled over simulated time.
- **NetworkX**: A library for creating, analyzing, and visualizing complex networks. In this simulation, it is used to represent the network topology (the nodes and their connections).
- **Pandas and Matplotlib**:
  - **Pandas** is used for collecting, organizing, and later analyzing the simulation data.
  - **Matplotlib** (specifically its pyplot module) is imported to provide tools for plotting and visualizing results, though its usage in the simulation is optional.
- **TensorFlow and Keras**: These libraries support the creation, training, and evaluation of deep learning models. In the simulation, they are used to build a model that predicts network traffic based on past simulation data.
- **Stable Baselines3 and Gym**:
  - **Stable Baselines3** provides advanced reinforcement learning (RL) algorithms such as PPO (Proximal Policy Optimization).
  - **Gym** supplies a standardized interface to build and test RL environments. The simulation includes a custom RL environment that integrates with Gym's API.
- **NumPy**: A numerical library that offers efficient array operations and is used for various calculations, particularly in the RL environment and when processing data for machine learning models.

---

2. Global Metrics

A global dictionary is defined to keep track of various metrics during the simulation. This dictionary contains counters for:

- **Route Request (RREQ) messages sent**
- **Route Reply (RREP) messages sent**
- **Route Error (RERR) messages sent**
- **Data packets forwarded**
- **Alarm messages sent**
- **Successful routes established**
- **Nodes that become deactivated due to energy loss**
- **Acknowledgement (ACK) packets sent and received**
- **Denial-of-Service (DoS) packets sent by malicious nodes**

These metrics are updated throughout the simulation and later used to calculate performance indicators, such as packet delivery ratios, energy consumption, and malicious detection rates.

---

3. Logging Configuration

The logging system is configured early in the simulation to capture detailed events. The logger is set up with:

- A **file handler** that writes all messages (even debug-level messages) to a text file, ensuring that a complete record is saved for later review.
- A **console handler** that prints less verbose (INFO-level and above) messages to the console so that the simulation's progress is visible without overwhelming the user with details.
- A **formatter** that timestamps each log entry and includes the log level and the message. This helps in tracking the simulation's flow and diagnosing issues when they arise.

---

4. Constants

Several constants are defined to standardize values used throughout the simulation. These include:

- **Energy-Related Constants** such as the base energy consumption per operation, the initial energy available to each node, and thresholds below which nodes are considered inactive.
- **Timing and Back-Off Parameters** for managing how quickly nodes attempt to send packets or discover new routes. A dynamic back-off mechanism helps to prevent excessive route discovery in low-energy situations.
- **Routing Table Size**: Determines how many potential routes can be stored for each destination.

- **Packet Type Identifiers**: String constants are defined to represent different types of packets (e.g., RREQ, RREP, FORWARDING, ALARM, ACK, DOS) to avoid errors and ensure consistency.
- **Protocol Names**: Constants are used to identify the two different routing protocols (DSR and SFA) simulated.
- **Default TTL (Time-To-Live)** for packets, ensuring that they are discarded if they travel too far.
- **Sleep Threshold**: The duration of inactivity after which a node will automatically enter sleep mode to conserve energy.
- **Minimum Send Interval**: Ensures that nodes wait a specified time between transmissions to avoid energy-draining burst transmissions.

---

5. Helper Functions

Three helper functions are provided:

- **Distance Calculation**: Computes the Euclidean distance between two nodes based on their randomly assigned positions. This is important for determining energy costs.
- **Energy Cost Calculation**: Determines how much energy is required to transmit a packet over a given distance. It scales the energy consumption based on distance but also ensures a minimum cost and a maximum cap.
- **Energy Consumption Function**: Decreases a node's energy when an operation (like sending or receiving a packet) occurs. If a node lacks sufficient energy, this function deactivates the node and clears its packet queue, ensuring that the simulation accurately reflects energy constraints.

---

6. Packet Class

A Packet class is defined to encapsulate all the information that travels between nodes. Each packet includes:

- **Type**: Identifies whether it is a route request, route reply, route error, forwarding data, alarm, ACK, or DOS packet.
- **Source and Destination IDs**: Indicates where the packet is coming from and where it should be delivered.
- **Sequence Number**: Helps to avoid processing duplicate packets and keeps track of packet order.
- **Payload**: Contains the actual data or additional information (such as sensor readings or alarm details).

- **Path**: A list that tracks which nodes the packet has passed through, which is especially important for route discovery.
- **Previous Node**: Records the last node that handled the packet.
- **TTL (Time-To-Live)**: Ensures that packets are eventually discarded if they circulate too long.
- **Final Destination**: In some protocols, distinguishes the ultimate recipient of a data packet from the immediate next hop.

---

7. DSR Protocol Class

The simulation implements the Dynamic Source Routing (DSR) protocol. The DSR protocol class includes methods to:

- **Broadcast Route Requests (RREQs)**: When a node needs a route, it increases its sequence number, updates the global counter, and sends out RREQ packets to all its neighbors. The packet's path is initialized with the node's own identifier.
- **Forward Data Packets**: Before sending a data packet, the node checks its routing table for an existing route to the final destination. If found, the packet is sent to the next hop; if not, the node initiates a new route discovery.
- **Send Route Error (RERR) Packets**: If a route fails, a route error message is generated and sent to notify other nodes, prompting them to remove the invalid route.

---

8. SFA Protocol Class

Similarly, the SFA protocol class (which represents an alternative or enhanced routing protocol) provides analogous methods to DSR but with differences in the packet types and possibly slight variations in the route discovery and data forwarding mechanisms. This class handles:

- **Sending SFA-Specific Route Requests**: Initiates a route request using an SFA-specific packet type.
- **Data Packet Forwarding**: Forwards data packets based on the SFA protocol's rules.
- **Sending SFA Route Errors**: Notifies nodes of route failures with SFA-specific error messages.

---

9. Reinforcement Learning Environment (RoutingEnv Class)

A custom environment for reinforcement learning is built using Gym's API. Key aspects include:

- **Action Space**: Defined as a discrete set of possible actions corresponding to the number of neighboring nodes. Essentially, the RL agent's decision is to choose which neighbor to forward a packet to.
- **Observation Space**: Represents the current state of the node, including its remaining energy, its current sequence number, and the number of routes available in its routing table.
- **Reset Method**: Initializes or resets the state when a new episode begins.
- **Step Method**: Simulates taking an action (forwarding a packet), calculates a reward (which is influenced by energy consumption), and updates the state. The step function returns the new state, reward, and an indicator of whether the episode is finished.
- **Render Method**: Although not implemented, it provides a placeholder for future visualization if needed.

---

10. Node Class with Advanced Features

The Node class is the central representation of an individual network node. It contains numerous properties and methods:

- **Properties**:
  - **Identification and State**: Each node has a unique identifier, an active state flag, and starts with a set amount of energy.
  - **Position**: Each node is assigned a random (x, y) position, which is used to calculate transmission distances.
  - **Routing Table**: Initially empty, this structure is later populated with potential routes to different destinations.
  - **Sequence Number**: Used to tag outgoing packets uniquely.
  - **Packet Queue**: A SimPy store where incoming packets are held until processed.
  - **Protocol Handler**: Depending on whether the node uses DSR or SFA, an appropriate protocol handler object is attached.
  - **Machine Learning and RL Models**: Optionally, a deep learning model for traffic prediction and an RL model for routing decisions can be assigned to a node.
- **Processes Initiated on Creation**:
  - **Main Run Process**: Continuously processes packets from the node's queue. Depending on the type of packet (e.g., DoS, ACK, or normal routing packets), the node takes appropriate actions.
  - **Sleep Cycle Management**: Monitors activity and puts the node into sleep mode after a period of inactivity to conserve energy.
  - **Sensor Data Transmission**: For non-base station nodes, simulates periodic sensor readings and transmits these readings to the base station.
- **Routing and Energy Management Methods**:
  - **Rate-Limiting**: A mechanism ensures that the node waits for a minimum interval between packet transmissions to avoid excessive energy drain.

    - **Energy Consumption and Deactivation**: Methods ensure that every transmission and reception consumes energy. If the node runs out of energy, it is deactivated and its packet queue is cleared.
    - **Routing Table Management**: The node can add routes, retrieve the best available route (considering factors like energy levels and route expiry), remove outdated or invalid routes, and periodically prune the routing table.
- **Handling of Special Packet Types**:
    - **Route Discovery Packets (RREQ, RREP, RERR)**: The node processes these packets to establish and maintain routes.
    - **Forwarding Packets**: Based on the packet's destination and the node's current routing table, data packets are forwarded toward their final destination.
    - **Alarm and Block Messages**: When alarms are received (e.g., indicating malicious behavior or high packet loss), the node processes these messages and may remove routes to suspicious nodes.
    - **Acknowledgements (ACKs)**: The node can send and process ACK packets to confirm the successful delivery of data.
- **Integration with ML and RL**:
    - **Traffic Prediction**: If a deep learning model is available, the node can predict future traffic based on current simulation time and protocol type.
    - **Routing Decisions**: The node can use an RL model to help decide when to initiate route discovery or forward packets, thereby optimizing performance based on learned behavior.

---

11. Malicious Node (DoS)

A specialized subclass of Node is created to simulate a malicious node. This node:

- Inherits all the properties of a normal node but has an additional flag indicating its malicious intent.
- Runs a process that repeatedly sends out Denial-of-Service (DoS) packets to all its neighbors. This flooding behavior is designed to disrupt normal network operations, drain energy from nearby nodes, and test the network's ability to detect and respond to malicious behavior.
- Updates the global counter for DoS packets each time it sends a malicious packet.

---

12. Base Station

The Base Station is another specialized type of node that acts as the central hub of the network. Key features include:

- It is marked specifically as the base station so that other nodes recognize it.
- It maintains a record of the last alarm sequence numbers from various nodes to avoid acting on outdated alarm messages.
- Its primary role is to receive sensor data and alarms, evaluate the information (such as identifying suspicious nodes), and, if necessary, broadcast block messages to other nodes.

---

13. Network Class

The Network class represents the entire network topology and is responsible for:

- **Creating Nodes**: The network is populated with a set number of normal nodes (typically 20, named using letters) along with one Base Station and one Malicious Node.
- **Graph Construction**: Using NetworkX, the nodes are added as vertices in a graph. The edges between nodes can either be defined by a provided edge list (for specific topologies) or generated randomly.
- **Edge Creation**: If an edge list is provided, the network creates the connections accordingly; if not, it randomly connects nodes while ensuring reasonable network connectivity.
- **Packet Delivery**: The network manages the actual sending of packets. When a node sends a packet, the network schedules its delivery after a random delay, simulating real network latencies. The network then places the packet in the destination node's queue.
- **Neighbor Discovery and Routing Updates**: The network provides utility methods to get neighbors for a given node and to remove routes associated with a node that has become inactive.

---

14. Setup Simulation

A helper function is provided to set up the simulation. This function:

- Creates a new network using the provided simulation environment, edge list, and protocol choice (either DSR or SFA).
- Iterates over all nodes in the network to assign the deep learning model, scaler, and RL model (if available) so that every node has access to these enhancements.
- Returns the fully configured network, making it easy to reuse the setup process for multiple simulation runs.

---

15. Data Collection

A dedicated process is set up for collecting simulation data at regular intervals. This process:

- Runs periodically (every few simulated time units) and records key information for every node in the network, such as current energy levels, active/inactive state, and counts of packets sent and received.
- Aggregates these data points into a structured format (using a data table) so that later analysis can compute performance metrics and generate visualizations.
- Stores the collected data in a data structure (typically a list of data frames) that is later concatenated and exported to a CSV file for further examination.

---

16. Machine Learning (ML) Model Training

After data has been collected, a function is used to train a deep learning model that predicts network traffic. The process involves:

- Concatenating all the collected data and converting categorical variables (such as protocol type) into numerical codes.
- Selecting relevant features (for example, simulation time and protocol) and targets (like the number of packets sent).
- Scaling the features so that the model can train effectively.
- Splitting the data into training and testing subsets.
- Building a neural network with multiple hidden layers, using an optimizer (Adam) and a loss function (mean squared error) to train on the data.
- Evaluating the model on a test set and logging the performance (mean absolute error) so that the effectiveness of the model can be gauged.
- Finally, returning both the trained model and the scaler for future predictions during subsequent simulation runs.

---

17. Reinforcement Learning (RL) Agent Training

A separate function is provided for training an RL agent using the PPO algorithm. The steps include:

- Creating an instance of the custom RL environment tailored for routing decisions.
- Instantiating the PPO algorithm with a multi-layer perceptron policy.
- Training the agent for a specified number of timesteps, during which the agent learns to optimize routing decisions (such as selecting the best neighbor to forward a packet) based on rewards that are influenced by energy consumption and successful packet delivery.

- Once training is complete, the RL model is returned and can be attached to nodes to assist in making dynamic routing decisions.

---

18. Processes for Data and Route Operations

Multiple processes are defined to simulate ongoing network operations:

- **Route Initiation**:
  Two separate processes exist for initiating routes:
  - One for nodes using the DSR protocol.
  - Another for nodes using the SFA protocol.

  Each process periodically selects a random active node (excluding the Base Station) and a random destination. The chosen node then begins route discovery by broadcasting a route request.

- **Data Forwarding**:
  There are processes that periodically simulate the forwarding of data:
  - For DSR, the process checks which nodes have valid routes (by examining their routing tables) and then sends a data packet along the best route.
  - For SFA, a similar process is executed, but it uses the SFA protocol's rules to forward packets.
- **RL-Driven Route Initiation**:
  An additional process is dedicated to invoking the RL-based routing decision method on a selected node. This allows the RL agent to make decisions at regular intervals, testing the integration of machine learning into routing.

---

19. Alarm Trigger Process

A dedicated process periodically checks for nodes that show signs of problematic behavior, such as a high packet loss rate (where the difference between packets sent and received exceeds a predefined threshold). When such nodes are found:

- The process generates an alarm packet containing details about the suspicious node and the packet loss observed.
- This alarm is sent to the Base Station, and a global alarm counter is incremented.
- The Base Station, upon receiving such alarms, can further investigate and, if necessary, broadcast block messages to isolate the suspected node from the network.

---

20. Main Simulation

The main simulation sequence is organized into two phases:

1. **Initial Simulation Run**:
    - Two separate networks are created using predefined edge lists: one for DSR and one for SFA.
    - In this phase, processes for data collection, route initiation, data forwarding, and alarm generation are started.
    - The simulation runs for a fixed duration (a defined number of simulated time units).
    - After this run, the collected data is exported to CSV files, and a sample of the data is logged.
2. **Enhanced Simulation with ML and RL**:
    - After the initial simulation, the collected data is used to train a deep learning model for traffic prediction.
    - A new simulation environment is set up, and the previously trained ML model (and its scaler) is injected into each node.
    - An RL agent is also trained for one of the nodes using the PPO algorithm.
    - In addition to the standard processes, a process for RL-driven route decisions is also launched.
    - This enhanced simulation is run for a similar duration.
    - At the end of the simulation, performance metrics such as total packets sent/received, energy consumption, packet delivery ratio, and detection rates are computed.
    - These metrics are compared between the DSR and SFA protocols, logged, and exported to CSV files.
    - Finally, a summary of the simulation, including all key metrics (like the total number of route requests, route replies, error messages, forwarded packets, alarms, acknowledgments, DoS packets, successful routes, and nodes deactivated due to energy loss), is compiled and saved as a text file.

---

21. Simulation Summary

After both simulation phases have completed, a final summary is generated that includes:

- A detailed report of the total counts of each type of packet sent.
- The number of routes successfully established.
- Energy metrics that indicate how much energy was consumed by each protocol.
- A comparison of the performance (for example, packet delivery ratio and energy consumption) between the DSR and SFA protocols.
- A section on the detection of malicious nodes (using alarm messages) and the overall effectiveness of the network in handling DoS attacks.

- This summary is saved in a text file and logged to provide a comprehensive overview of the simulation's functionality and results.

Overall Summary

This simulation is a comprehensive framework for modeling a wireless network that uses two different routing protocols (DSR and SFA). It encompasses:

- **Node Behavior**: Simulating individual nodes with energy constraints, random positions, and sleep cycles.
- **Routing Protocols**: Implementing the specific mechanisms of route discovery, data forwarding, and error handling for both DSR and SFA.
- **Security Considerations**: Including a malicious node to simulate DoS attacks and processes to detect and mitigate such behavior.
- **Data Collection and Analysis**: Continuously collecting network performance data for later analysis.
- **Advanced Enhancements**: Integrating machine learning for traffic prediction and reinforcement learning to optimize routing decisions.
- **Performance Evaluation**: Comparing the efficiency, energy consumption, and robustness of the two protocols and exporting the results for further study.