# Overview

* **Purpose:**  
  The code simulates a wireless network with nodes that communicate using advanced routing protocols (DSR and SFA). It models real-world factors such as energy consumption, sleep cycles, and route maintenance. In addition, the simulation integrates machine learning (ML) components for traffic prediction and reinforcement learning (RL) to help nodes make adaptive routing decisions. Finally, it collects performance metrics and exports the data for further analysis.
* **Main Technologies:**
  + **SimPy:** For discrete-event simulation.
  + **NetworkX:** To represent the network topology as a graph.
  + **TensorFlow/Keras:** For training a neural network traffic prediction model.
  + **Stable-Baselines3 (PPO):** For training an RL agent.
  + **Pandas & Matplotlib:** For data collection and visualization.

# Global Metrics and Logging

* **Global Metrics:**  
  A dictionary (metrics\_summary) is maintained to keep track of several simulation-wide counts, such as the number of route request (RREQ) packets sent, route reply (RREP) packets sent, error messages, forwarded packets, alarm messages, routes established, and nodes deactivated because of energy depletion.
* **Logging:**  
  A logging system is configured to log events both to a file (simulation\_output.txt) and to the console. Different logging levels (DEBUG for detailed tracing and INFO for higher-level events) are used to monitor simulation progress and events.

# Constants and Helper Functions

* **Constants:**  
  Various simulation constants are defined including energy consumption factors, thresholds for alarms and packet loss, maximum sequence numbers, sleep thresholds, and routing-related timeouts. These constants control how nodes manage energy, how long routes remain valid, and how back-off strategies work during route discovery.
* **Helper Functions:**
  + **Distance and Energy Calculation:**  
    Functions like compute\_distance and compute\_energy\_cost calculate the Euclidean distance between nodes and determine the energy cost for transmissions based on that distance. A maximum energy cost cap ensures that no single transmission depletes the node’s energy dramatically.
  + **Energy Consumption:**  
    The consume\_energy function deducts energy from a node when sending or receiving a packet. It also deactivates nodes that cannot sustain the minimum required energy, ensuring that nodes “die” realistically when their energy is too low.

# Packet Class

* **Packet Representation:**  
  The Packet class encapsulates the data sent between nodes. Each packet includes its type (e.g., RREQ, RREP, forwarding, ALARM), source and destination IDs, sequence number, payload, path information, and time-to-live (TTL). This unified representation supports multiple types of messages needed for both routing protocols and alarm/block messages.

# Routing Protocols (DSR and SFA)

* **DSR (Dynamic Source Routing) Protocol:**  
  The DSRProtocol class defines methods for route discovery, data forwarding, and error handling. For example:
  + send\_rreq: Broadcasts a route request to neighbors.
  + send\_forwarding\_packet: Attempts to forward data along an established route, initiating route discovery if necessary.
  + send\_rerr: Sends an error message if a route is broken.
* **SFA (Secure Fault-tolerant Adaptive) Protocol:**  
  Similarly, the SFAProtocol class handles route discovery and error recovery for the SFA protocol. The main difference lies in the packet types (e.g., SFA\_ROUTE\_REQUEST, SFA\_ROUTE\_REPLY) and slight variations in the handling logic.

# Reinforcement Learning Environment

* **Custom Gym Environment:**  
  The RoutingEnv class extends OpenAI Gym’s Env to create an environment in which an RL agent can learn routing decisions. The environment’s state includes parameters like the node’s current energy, sequence number, and the count of available routes. Actions typically correspond to selecting a neighbor for forwarding packets, and rewards are designed to favor energy efficiency.

# Node Class – Core of the Simulation

* **Node Initialization:**  
  Each node is an instance of the Node class. It contains attributes such as energy, position (randomly generated within a defined area), routing tables, and counters for packets sent/received. Nodes are instantiated with a specific protocol (DSR or SFA) and can also be enhanced with ML or RL capabilities.
* **Packet Processing:**  
  The run method is the main loop for each node. It continuously processes incoming packets from its queue. Depending on the node type (regular node or base station) and the packet type (RREQ, RREP, FORWARDING, ALARM, etc.), different handler methods are called.
* **Routing Table Management:**  
  Nodes maintain a routing table where multiple routes (up to a defined maximum) can be stored per destination. Methods like add\_route, get\_route, and prune\_routing\_table ensure that the routes are up-to-date and that expired or non-functional routes are removed.
* **Sleep Cycle and Energy Management:**  
  Nodes go to sleep if inactive for a certain period to save energy, and they are reactivated when needed. The simulation carefully manages energy consumption both for sending and processing packets.
* **ML and RL Hooks:**  
  Nodes can use a trained deep learning model to predict traffic and adjust their routing strategies accordingly. Additionally, if an RL model is attached, a node can use it to decide which neighbor to forward packets to, thereby dynamically adapting to network conditions.

# Base Station and Alarm Handling

* **Base Station Role:**  
  The BaseStation class inherits from Node but is designated to handle special alarm messages. When an alarm is received (which indicates potential malicious behavior or abnormal packet loss), the base station processes the alarm and, if necessary, broadcasts block messages to isolate suspicious nodes.
* **Alarm and Block Message Processing:**  
  Nodes generate alarm messages when they detect high packet loss. The base station then verifies these alarms, checks for energy discrepancies, and issues block messages to notify other nodes to remove routes to the malicious node.

# Network Topology and Communication

* **Network Graph Creation:**  
  The Network class sets up the network topology using NetworkX. Nodes are added to a graph, and connections (edges) can be either predefined (through an edge list) or generated randomly. Each node gets a routing table that is initialized based on the total number of nodes in the network.
* **Packet Delivery:**  
  A central method (send\_packet) in the Network class handles delivering packets from one node to another, introducing random transmission delays to simulate network latency.

# Data Collection and Performance Analysis

* **Periodic Data Logging:**  
  A separate process periodically collects metrics such as node energy levels, packet counts, and alarm counts. These data points are stored in a Pandas DataFrame and eventually exported to CSV files for analysis.
* **Performance Metrics Calculation:**  
  After running simulations, the code calculates key performance indicators like Packet Delivery Ratio (PDR), total energy consumed, and the detection rate of malicious behavior. These metrics are also exported and logged.

# Machine Learning and Reinforcement Learning Enhancements

* **Traffic Prediction Model:**  
  The function train\_traffic\_prediction\_model trains a neural network (using Keras) on the collected simulation data to predict traffic patterns. This model, along with a scaler for normalizing input features, is later used by nodes to adapt their routing decisions.
* **RL Agent Training:**  
  An RL agent is trained using the PPO algorithm (from stable-baselines3) in a custom environment (RoutingEnv). The agent learns to make routing decisions (i.e., which neighbor to forward packets to) based on the current network state, and the trained model is then attached to a node for further simulation.

# Simulation Scenarios

* **First Simulation Run:**  
  The simulation is first run in two separate environments (one for DSR and one for SFA) with predefined edge lists. During this run, routing requests, data forwarding, and alarm messages are generated. Data is collected for later analysis and for training the ML model.
* **Second Simulation Run (Enhanced):**  
  Using the trained ML model and RL agent, a second simulation environment is set up. In this run, nodes can use ML predictions to adjust their routing strategies and RL for making more nuanced forwarding decisions. The performance data from this run is again collected, exported, and compared against the first simulation run.
* **Final Reporting:**  
  At the end of the simulation, a summary of overall metrics (such as the total number of various packet types sent, routes established, and nodes deactivated) is written to a text file. Additionally, performance metrics are computed and compared between the two protocols (DSR and SFA), and the results are logged.

**Conclusion**

This simulation code provides a comprehensive framework for:

* Modeling a wireless ad-hoc network with realistic energy and sleep management.
* Implementing and comparing two different routing protocols (DSR and SFA).
* Integrating advanced techniques like ML for traffic prediction and RL for dynamic routing decisions.
* Collecting extensive performance data to evaluate metrics such as packet delivery ratio, energy consumption, and malicious behavior detection.