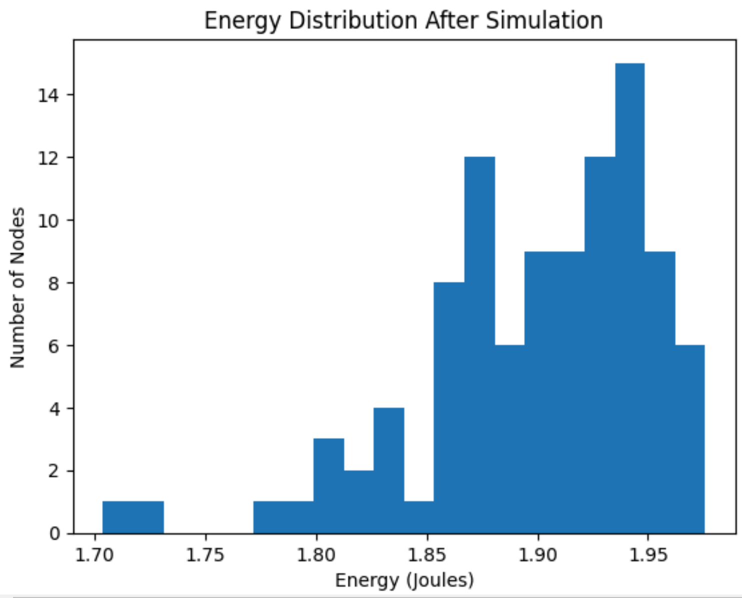
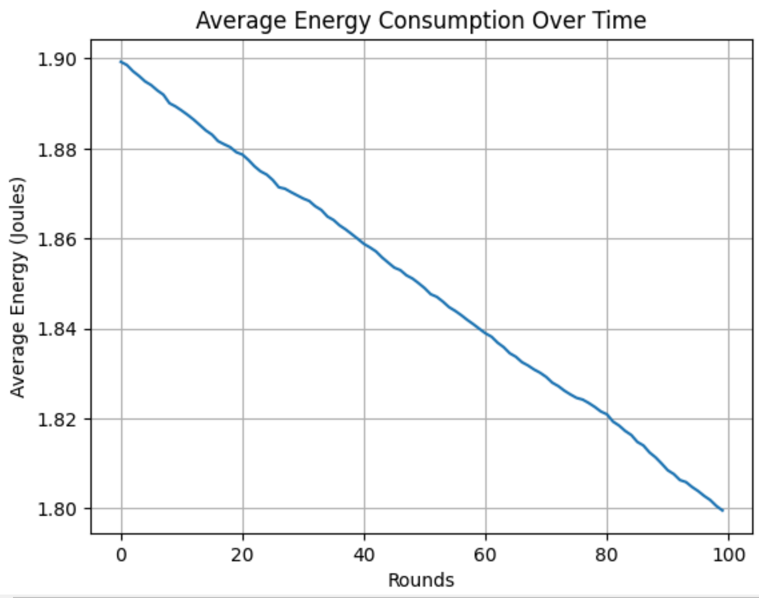
```python
import numpy as np
import matplotlib.pyplot as plt
# Parameters
num_nodes = 100
area_size = 100
initial_energy = 2  # in Joules
packet_size = 500  # in bytes
control_packet_size = 25  # in bytes
base_station = (50, 50)
rounds = 100
prob_ch = 0.1  # Probability of a node becoming a cluster head

# Energy model (in Joules)
E_elec = 50 * 10**-9
E_amp = 100 * 10**-12
E_da = 5 * 10**-9

# Node class
class Node:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.energy = initial_energy
        self.is_cluster_head = False

    def distance(self, other):
        return np.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

# Initialize nodes
nodes = [Node(np.random.uniform(0, area_size), np.random.uniform(0, area_size)) for _ in range(num_nodes)]

# Function to select cluster heads
def select_cluster_heads(nodes):
    for node in nodes:
        node.is_cluster_head = np.random.rand() < prob_ch

# Function to simulate a round
def simulate_round():
    global nodes
    select_cluster_heads(nodes)

    for node in nodes:
        if node.is_cluster_head:
            for other_node in nodes:
                if not other_node.is_cluster_head:
                    distance = node.distance(other_node)
                    # Energy consumption for transmission
                    other_node.energy -= (E_elec * control_packet_size + E_amp * control_packet_size * (distance ** 2))
                    node.energy -= (E_elec * control_packet_size)
                    # Data aggregation energy consumption
                    node.energy -= (E_da * packet_size)
                    # Transmission to base station
                    distance_to_bs = node.distance(Node(base_station[0], base_station[1]))
                    node.energy -= (E_elec * packet_size + E_amp * packet_size * (distance_to_bs ** 2))

# Function to run simulation
def run_simulation(rounds):
    global nodes
    for _ in range(rounds):
        simulate_round()

    # Gather statistics
    remaining_energy = [node.energy for node in nodes]
    dead_nodes = sum(1 for energy in remaining_energy if energy <= 0)
    print(f"Remaining Energy: {np.mean(remaining_energy)} J")
    print(f"Dead Nodes: {dead_nodes}")

    # Plot remaining energy
    plt.hist(remaining_energy, bins=20)
    plt.xlabel('Energy (Joules)')
    plt.ylabel('Number of Nodes')
    plt.title('Energy Distribution After Simulation')
    plt.show()

# Run the simulation
run_simulation(rounds)
```

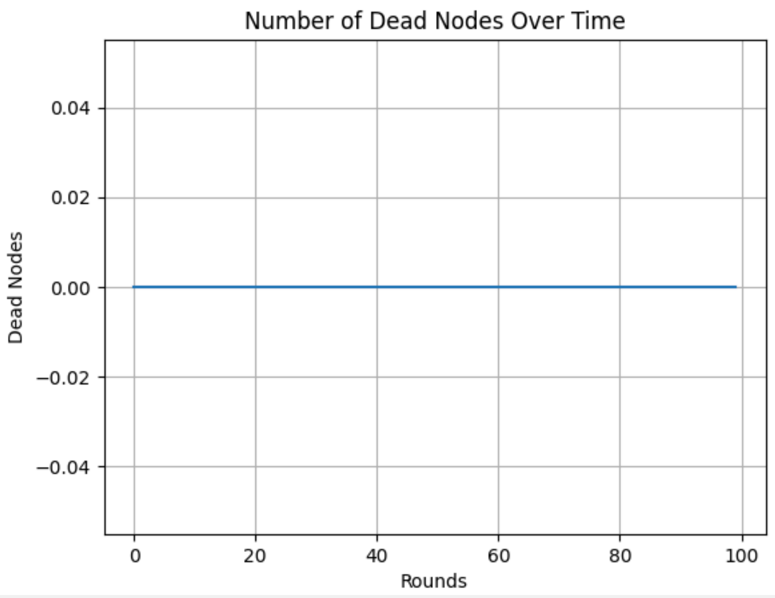
```
Remaining Energy: 1.8998365754218218 J
Dead Nodes: 0
```



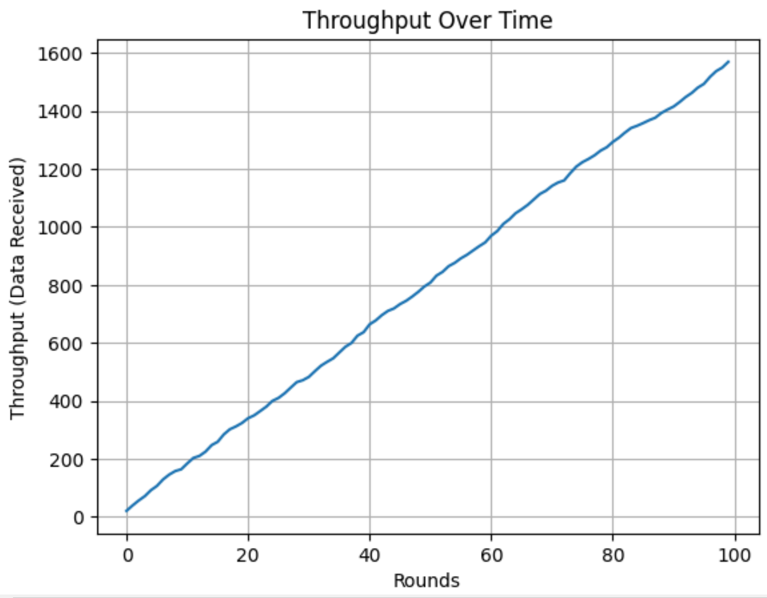
```
def plot_energy_over_time(rounds):
    global nodes
    avg_energy = []
    for _ in range(rounds):
        simulate_round()
        remaining_energy = [node.energy for node in nodes]
        avg_energy.append(np.mean(remaining_energy))

    plt.plot(range(rounds), avg_energy)
    plt.xlabel('Rounds')
    plt.ylabel('Average Energy (Joules)')
    plt.title('Average Energy Consumption Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_energy_over_time(rounds)
```



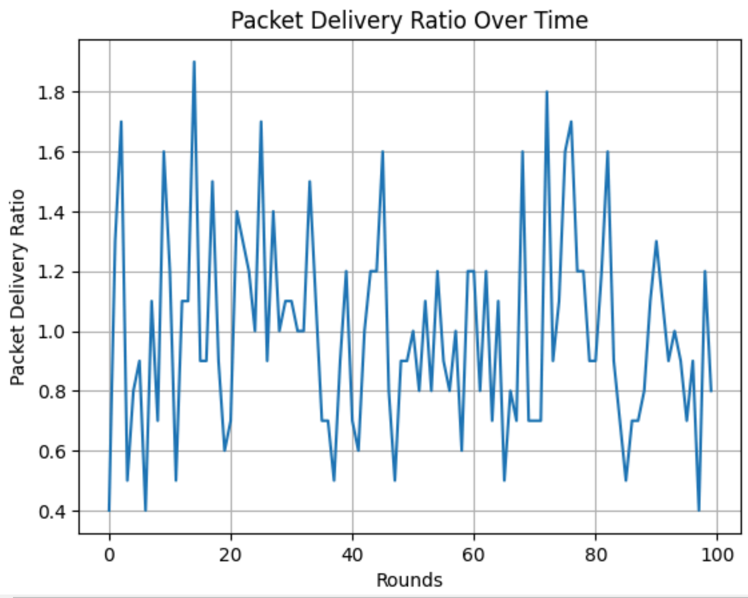
```
def plot_dead_nodes_over_time(rounds):
    global nodes
    dead_nodes = []
    for _ in range(rounds):
        simulate_round()
        dead_nodes.append(sum(1 for node in nodes if node.energy <= 0))

    plt.plot(range(rounds), dead_nodes)
    plt.xlabel('Rounds')
    plt.ylabel('Dead Nodes')
    plt.title('Number of Dead Nodes Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_dead_nodes_over_time(rounds)
```



```
def plot_throughput_over_time(rounds):
    global nodes
    throughput = []
    total_data_received = 0
    for _ in range(rounds):
        simulate_round()
        total_data_received += sum(node.energy for node in nodes if node.is_cluster_head and node.energy > 0)
        throughput.append(total_data_received)

    plt.plot(range(rounds), throughput)
    plt.xlabel('Rounds')
    plt.ylabel('Throughput (Data Received)')
    plt.title('Throughput Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_throughput_over_time(rounds)
```

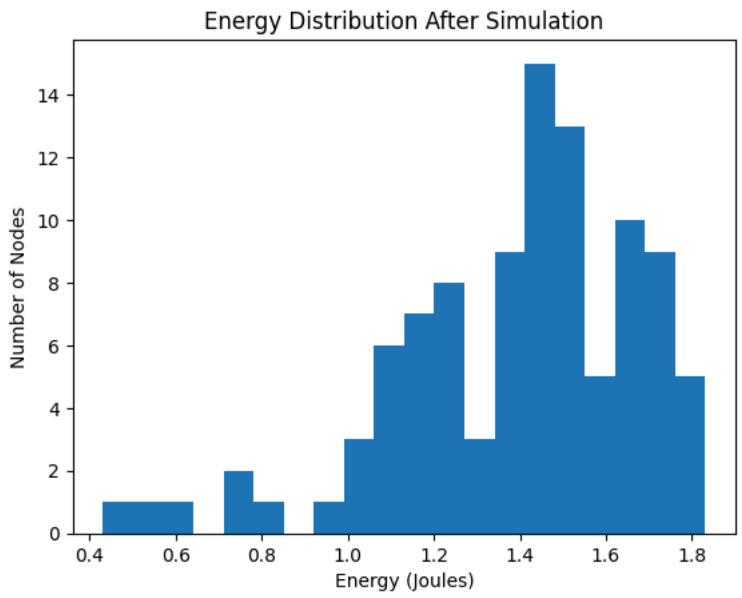


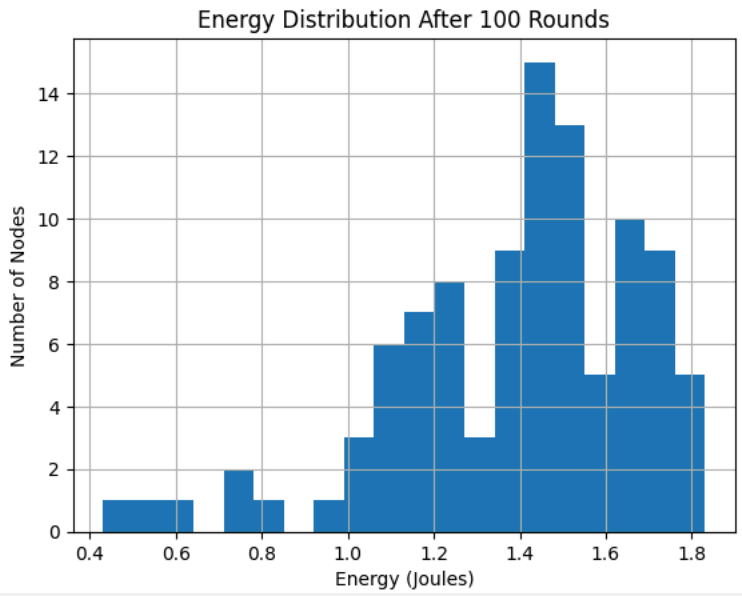
```
def plot_packet_delivery_ratio_over_time(rounds):
    global nodes
    pdr = []
    for _ in range(rounds):
        simulate_round()
        successful_transmissions = sum(1 for node in nodes if node.is_cluster_head and node.energy > 0)
        total_transmissions = len(nodes) * prob_ch
        pdr.append(successful_transmissions / total_transmissions)

    plt.plot(range(rounds), pdr)
    plt.xlabel('Rounds')
    plt.ylabel('Packet Delivery Ratio')
    plt.title('Packet Delivery Ratio Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_packet_delivery_ratio_over_time(rounds)
```



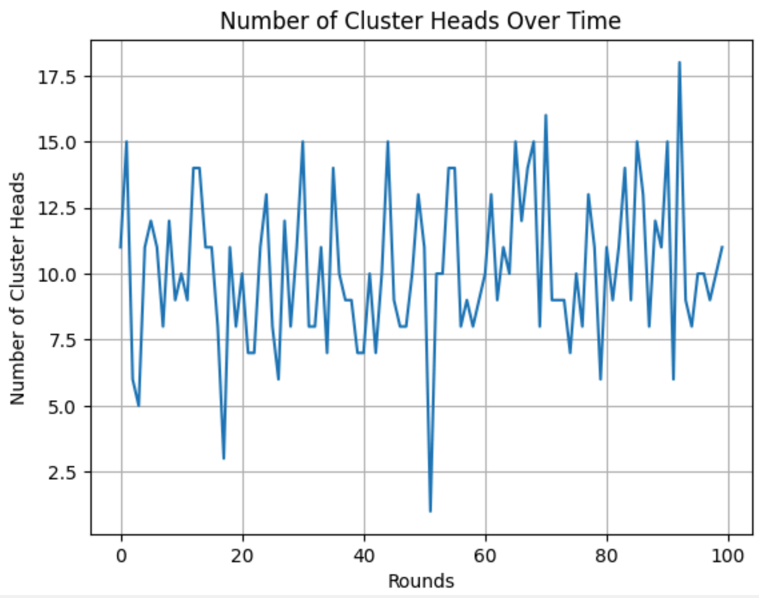
```
def plot_energy_distribution(rounds):
    global nodes
    run_simulation(rounds)
    remaining_energy = [node.energy for node in nodes]
    plt.hist(remaining_energy, bins=20)
    plt.xlabel('Energy (Joules)')
    plt.ylabel('Number of Nodes')
    plt.title(f'Energy Distribution After {rounds} Rounds')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_energy_distribution(rounds)
```

```
Remaining Energy: 1.3942000496253713 J
Dead Nodes: 0
```

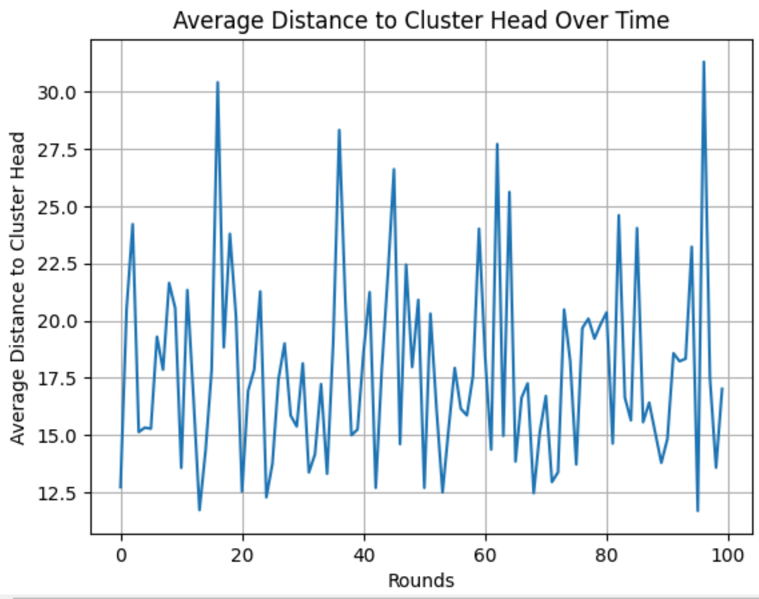




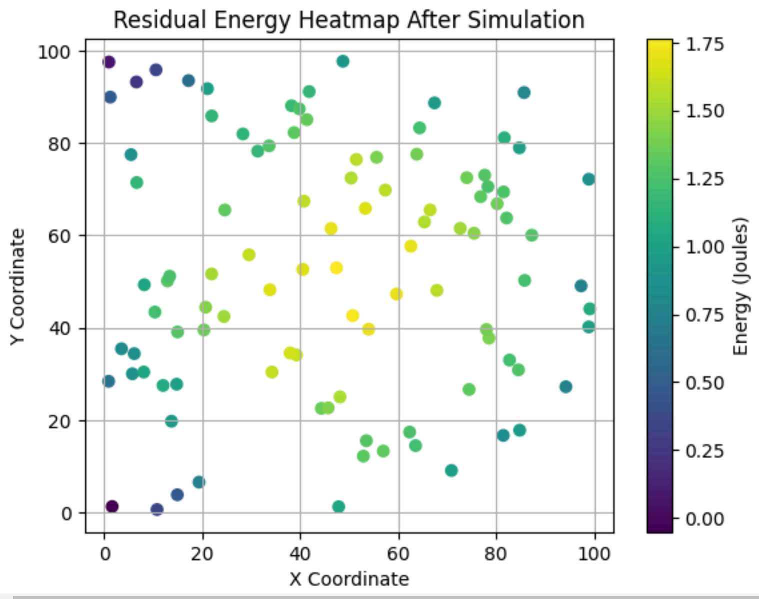
```
def plot_cluster_heads_over_time(rounds):
    global nodes
    cluster_heads_count = []
    for _ in range(rounds):
        simulate_round()
        cluster_heads_count.append(sum(1 for node in nodes if node.is_cluster_head))

    plt.plot(range(rounds), cluster_heads_count)
    plt.xlabel('Rounds')
    plt.ylabel('Number of Cluster Heads')
    plt.title('Number of Cluster Heads Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_cluster_heads_over_time(rounds)
```



```
def plot_avg_distance_to_cluster_head_over_time(rounds):
    global nodes
    avg_distances = []
    for _ in range(rounds):
        simulate_round()
        total_distance = 0
        count = 0
        for node in nodes:
            if not node.is_cluster_head:
                ch = min((other_node for other_node in nodes if other_node.is_cluster_head),
                         key=lambda ch: node.distance(ch))
                total_distance += node.distance(ch)
                count += 1
        avg_distances.append(total_distance / count if count > 0 else 0)

    plt.plot(range(rounds), avg_distances)
    plt.xlabel('Rounds')
    plt.ylabel('Average Distance to Cluster Head')
    plt.title('Average Distance to Cluster Head Over Time')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_avg_distance_to_cluster_head_over_time(rounds)
```

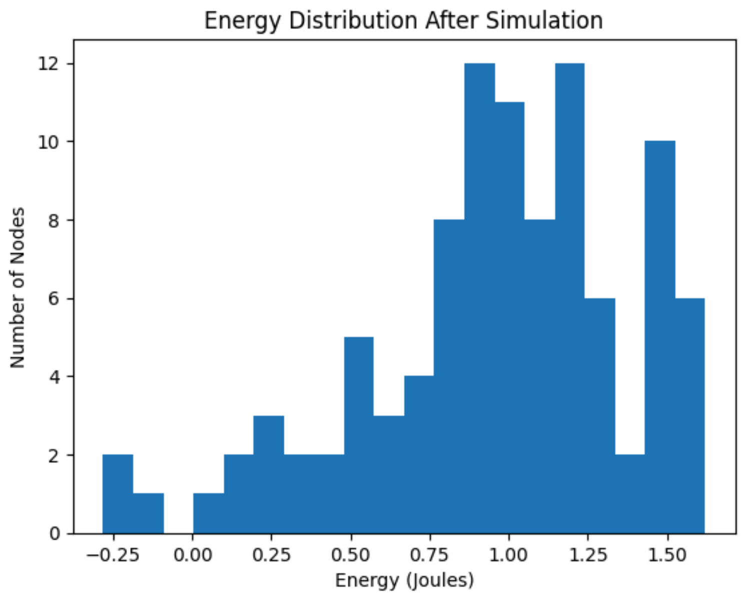


```
def plot_residual_energy_heatmap():
    global nodes
    x = [node.x for node in nodes]
    y = [node.y for node in nodes]
    energy = [node.energy for node in nodes]

    plt.scatter(x, y, c=energy, cmap='viridis', marker='o')
    plt.colorbar(label='Energy (Joules)')
    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.title('Residual Energy Heatmap After Simulation')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_residual_energy_heatmap()
```

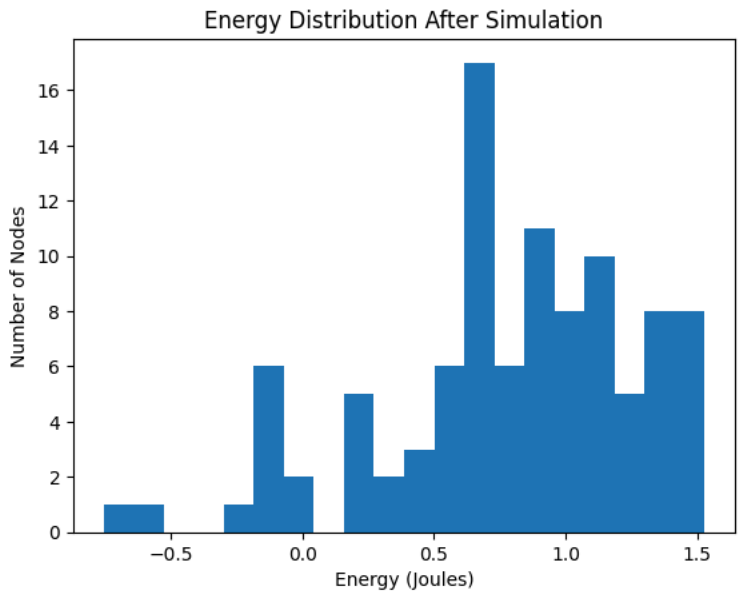


```
def plot_energy_vs_base_station_distance():
    global base_station, nodes
    distances = []
    energies = []
    for bs_distance in range(10, area_size, 10):
        base_station = (bs_distance, bs_distance)
        run_simulation(rounds)
        remaining_energy = np.mean([node.energy for node in nodes])
        distances.append(bs_distance)
        energies.append(remaining_energy)

    plt.plot(distances, energies)
    plt.xlabel('Base Station Distance from Origin')
    plt.ylabel('Average Energy (Joules)')
    plt.title('Impact of Base Station Distance on Energy Consumption')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_energy_vs_base_station_distance()
```

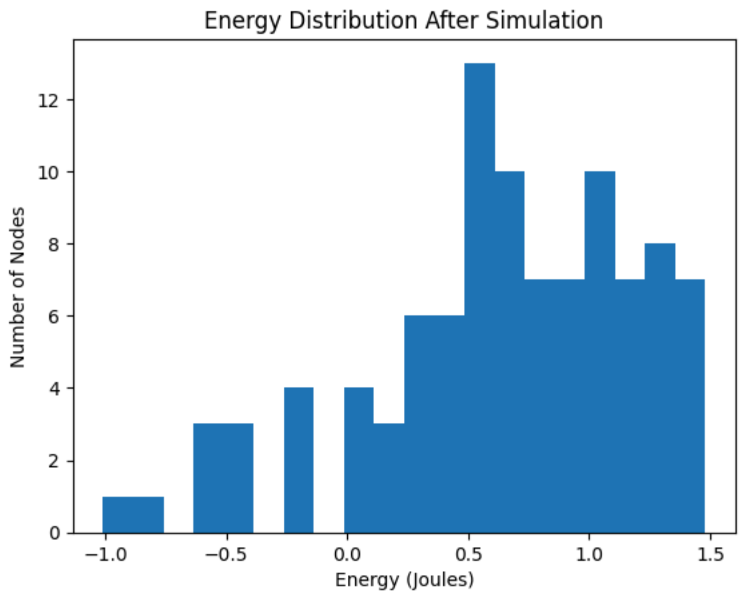
```
Remaining Energy: 0.9584672072341085 J
Dead Nodes: 3
```



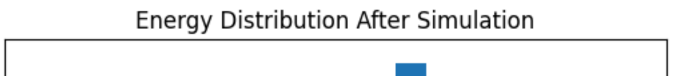
```
Remaining Energy: 0.7812399957269635 J
Dead Nodes: 10
```

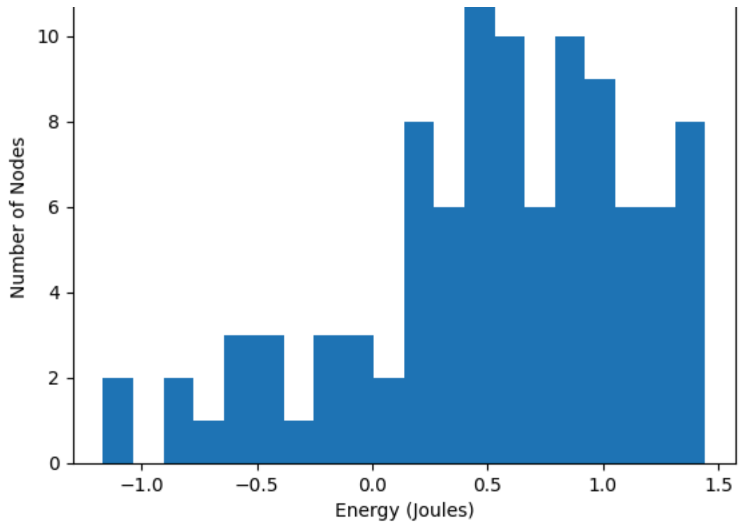


```
Remaining Energy: 0.6439998454068732 J
Dead Nodes: 13
```

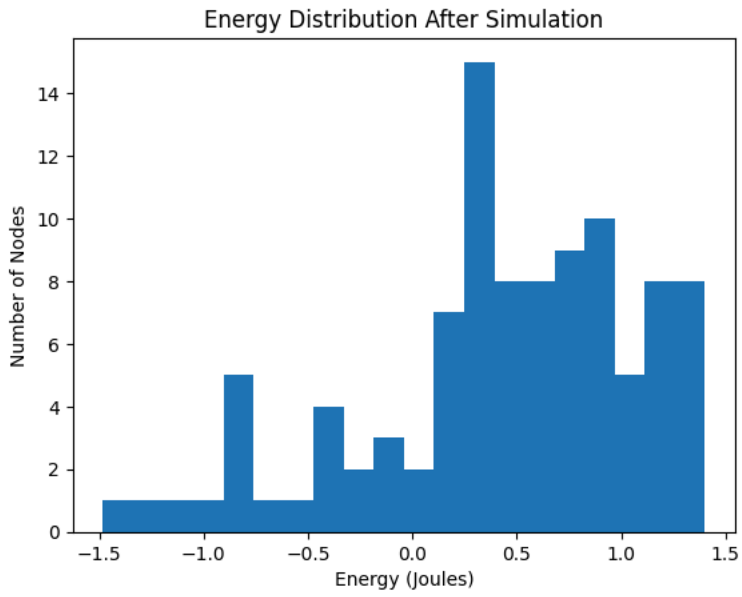


```
Remaining Energy: 0.5297315557137366 J
Dead Nodes: 18
```

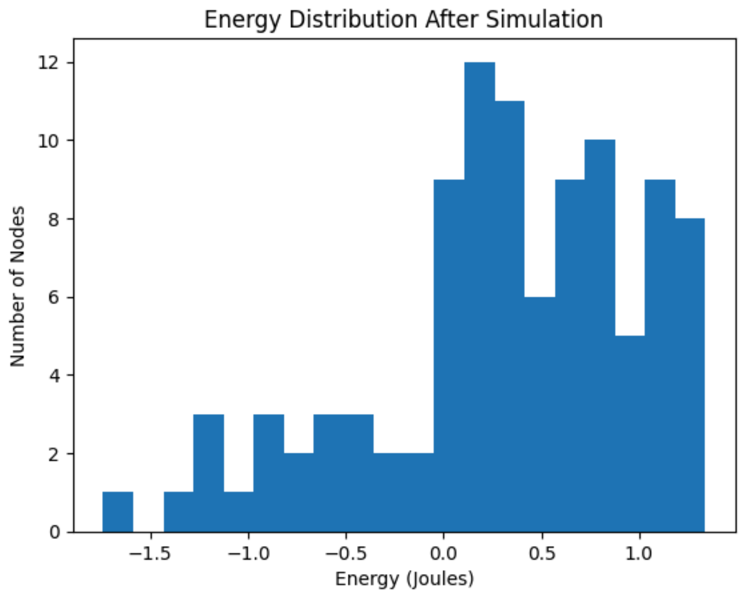




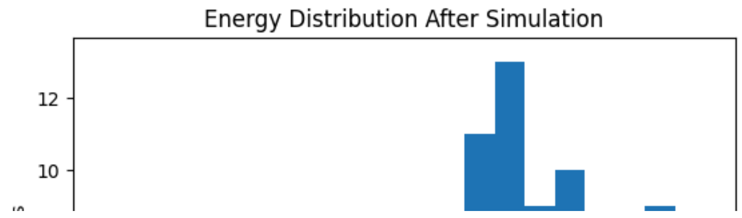
```
Remaining Energy: 0.42510507107505746 J
Dead Nodes: 21
```

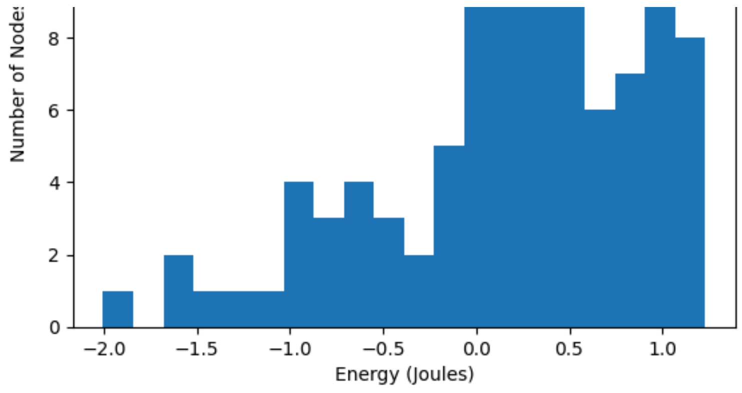


```
Remaining Energy: 0.3180231549655865 J
Dead Nodes: 23
```

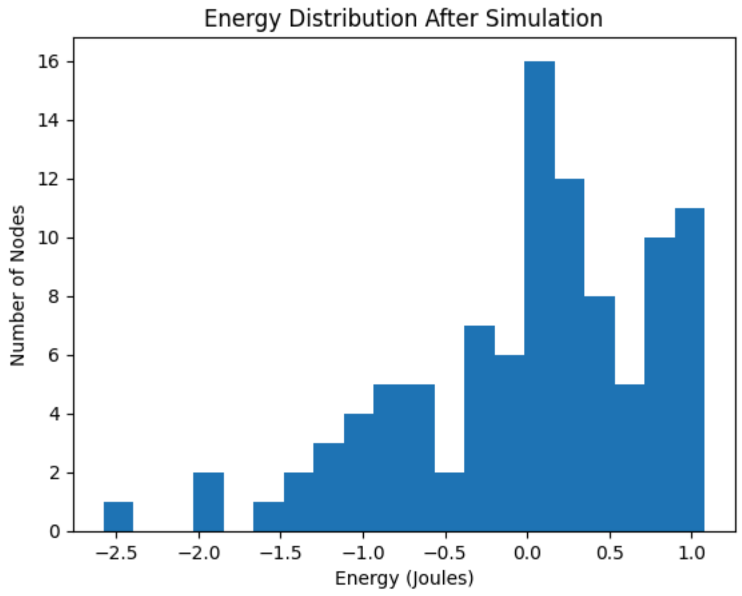


```
Remaining Energy: 0.1821766602285425 J
Dead Nodes: 29
```

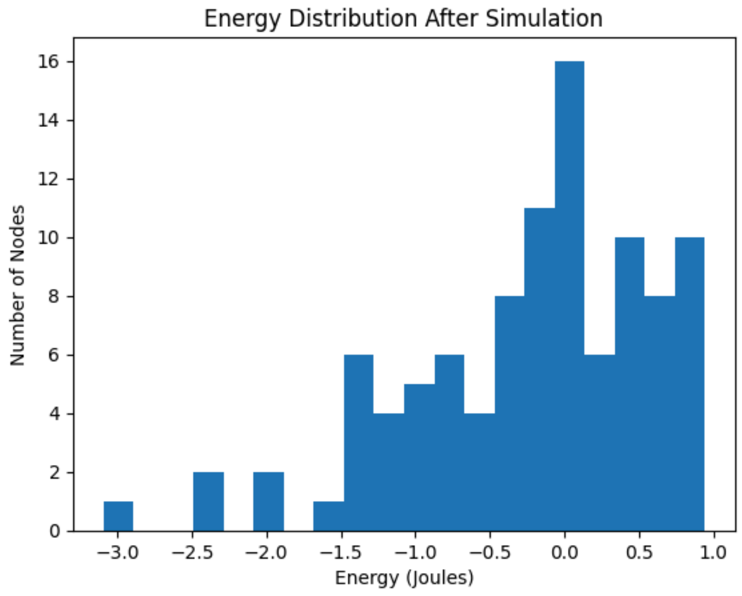


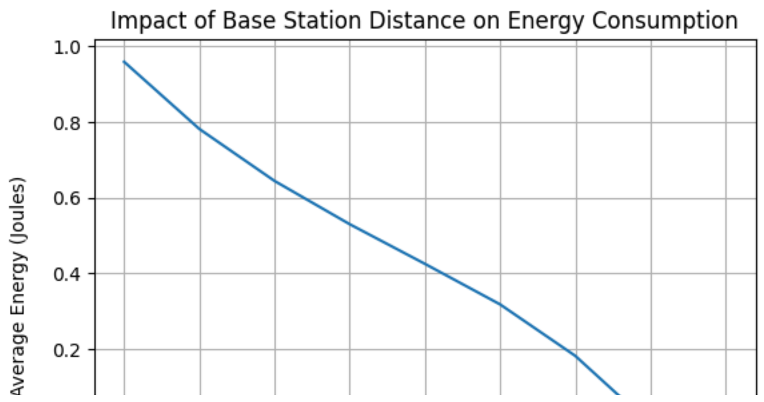


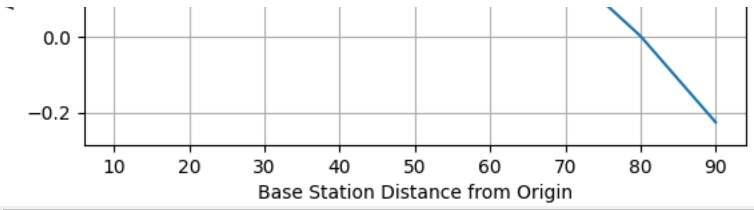
```
Remaining Energy: 0.0011976770453591867 J
Dead Nodes: 40
```



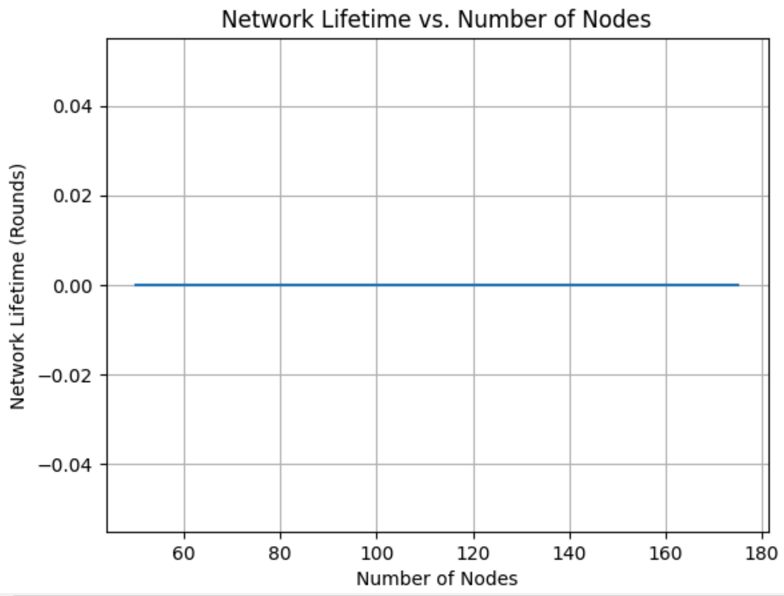
```
Remaining Energy: -0.22602319322977352 J
Dead Nodes: 55
```







```
def plot_network_lifetime_vs_num_nodes():
    global num_nodes, nodes
    num_nodes_list = range(50, 200, 25)
    lifetimes = []

    for n in num_nodes_list:
        num_nodes = n
        nodes = [Node(np.random.uniform(0, area_size), np.random.uniform(0, area_size)) for _ in range(num_nodes)]
        lifetime = 0
        for i in range(rounds):
            simulate_round()
            if any(node.energy <= 0 for node in nodes):
                lifetime = i
                break
        lifetimes.append(lifetime)

    plt.plot(num_nodes_list, lifetimes)
    plt.xlabel('Number of Nodes')
    plt.ylabel('Network Lifetime (Rounds)')
    plt.title('Network Lifetime vs. Number of Nodes')
    plt.grid(True)
    plt.show()

# Generate the plot
plot_network_lifetime_vs_num_nodes()
```



```
# Required Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Simulation Parameters
num_devices = 500
time_intervals = 100
energy_levels = np.random.uniform(low=0.2, high=1.0, size=num_devices)  # Random energy levels for devices
device_coordinates = np.random.rand(num_devices, 2) * 100  # Random coordinates in a 100x100 area

# Initial Cluster Formation using KMeans for simplicity
num_clusters = int(np.sqrt(num_devices) / 2)  # Initial number of clusters
kmeans = KMeans(n_clusters=num_clusters, random_state=42).fit(device_coordinates)
cluster_labels = kmeans.labels_

# Adaptive Routing Protocol Simulation (Simplified Version)
def adaptive_routing_simulation(time_intervals, device_coordinates, energy_levels, cluster_labels):
    energy_consumption_log = []
    performance_log = []

    for t in range(time_intervals):
        # Randomly simulate network conditions and device usage leading to energy depletion
        energy_levels -= np.random.uniform(low=0.01, high=0.05, size=num_devices)

        # Re-cluster if necessary based on energy levels or network conditions change
        if t % 10 == 0:  # Re-cluster every 10 intervals for this simulation
            num_clusters = int(np.sqrt(num_devices) / 2) + np.random.randint(-2, 3)  # Adjust cluster count
            kmeans = KMeans(n_clusters=num_clusters, random_state=42+t).fit(device_coordinates)
            cluster_labels = kmeans.labels_

        # Log energy consumption and silhouette score as a proxy for network performance/efficiency
        energy_consumption_log.append(np.sum(energy_levels))
        performance_log.append(silhouette_score(device_coordinates, cluster_labels))

    return energy_consumption_log, performance_log

energy_consumption_log, performance_log = adaptive_routing_simulation(
    time_intervals, device_coordinates, energy_levels, cluster_labels)

# Visualization of Simulation Results
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(energy_consumption_log, label='Total Energy Consumption')
plt.title('Energy Consumption Over Time')
plt.xlabel('Time Interval')
plt.ylabel('Energy Units')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(performance_log, label='Network Performance', color='green')
plt.title('Network Performance Over Time')
plt.xlabel('Time Interval')
plt.ylabel('Silhouette Score')
plt.legend()

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
```