BMSim

IoT Fundamentals Course

Amirreza Hosseini 9820363

Further Improvements for Bluetooth mesh simulator:

As we discussed before about the implementation and simulation of how this simulator works, here is some suggestions for further improvement of this platform:

We could modify the implementation of initializer.py file like this for better performance.

Here are some examples of how we can implement the better version for simulator code:

1. Using functions: we can use functional structures for more modularity of the code.

```
def read_log_file(file_name):
    allNums = []
    with open(file_name+'.log', "r+") as f:
        data = f.readlines()
        for line in data:
            allNums += line.strip().split("    ")
    return allNums

def extract_metrics(allNums, relay_node, NUMBER_NODES):
    total = []
    source_node = []
    receive_time = []
    seq_number = []
    generation_time = []
    node = []
    nodes = []
    max_seq = [0 for x in range(NUMBER_NODES)]

    for num in allNums:
        if num == '':
            continue
        else:
            total.append(float(num))

    for i_m in range(NUMBER_NODES):
        max_seq[NUMBER_NODES-i_m-1] = total[len(total)-1]
        total.pop(len(total)-1)

    for i in range(0, len(total), 5):
```

```
        source_node.append(total[i])

    for i1 in range(2, len(total), 5):
        receive_time.append(total[i1])

    for i2 in range(3, len(total), 5):
        seq_number.append(total[i2])

    for i3 in range(4, len(total), 5):
        generation_time.append(total[i3])

    j = 0
    while j < NUMBER_NODES:
        for i4 in range(len(source_node)):
            if source_node[i4] == j:
                node.append([source_node[i4], generation_time[i4],
receive_time[i4], seq_number[i4]])
        nodes.append(node)
        node = []
        j += 1

    nodes_PDR = []
    all_latency = []
    burst_packet_loss_all = []

    for s in range(NUMBER_NODES):
        latency = []
        seq = []
        burst_packet_loss = []
        for d in range(len(nodes[s])):
            latency.append(nodes[s][d][2]-nodes[s][d][1])
        if len(latency) == 0:
            latency.append(0)
        all_latency.append(latency)

        for d1 in range(len(nodes[s])):
            seq.append(nodes[s][d1][3])
            if d1 >= (len((nodes[s]))-1):
                burst_packet_loss.append(0)
            else:
                burst_packet_loss.append((nodes[s][d1+1][3]-nodes[s][d1][3])-1)
        if len(burst_packet_loss) == 0:
            burst_packet_loss.append(0)
        burst_packet_loss_all.append(burst_packet_loss)
```

```
        if max_seq[s] != 0:
            PDR = (len(Counter(seq).keys())/max_seq[s]) * 100
        else:
            PDR = 0
        nodes_PDR.append(PDR)

    return nodes_PDR, all_latency, burst_packet_loss_all, max_seq

allNums = read_log_file('network_detail')
```

2. Code optimization:

```
# instead of using loops to iterate over arrays, we can use numpy operations
max_seq = np.zeros(NUMBER_NODES)
for i_m in range(NUMBER_NODES):
    max_seq[NUMBER_NODES-i_m-1] = total.pop()

# or list comprehension
source_node = [total[i] for i in range(0, len(total), 5)]
receive_time = [total[i] for i in range(2, len(total), 5)]
seq_number = [total[i] for i in range(3, len(total), 5)]
generation_time = [total[i] for i in range(4, len(total), 5)]

nodes_PDR, all_latency, burst_packet_loss_all, max_seq = extract_metrics(allNums,
relay_node=1, NUMBER_NODES=49)
```

we could add other factors in performance_analyzer.py for better understanding and analysis of our network too. here is some of them:

1. Network throughput: To calculate network throughput, we can divide the total amount of data transmitted over a given period of time by the duration of that period. Here's an example code snippet in Python using pandas to calculate network throughput:

```
import pandas as pd

# read in log file with data transmission details
df = pd.read_csv('data_transmission_log.csv')

# calculate total bytes transmitted
total_bytes = df['bytes_sent'].sum()

# calculate duration of transmission period in seconds
```

```
start_time = df['timestamp'].min()
end_time = df['timestamp'].max()
duration = (end_time - start_time).total_seconds()

# calculate network throughput in bits per second
throughput = (total_bytes * 8) / duration
print(f'Network throughput: {throughput} bps')
```

2. Network capacity: To estimate network capacity, we can perform a stress test on the network and measure the maximum amount of data that can be transmitted within a given time period without exceeding certain performance thresholds. Here's an example code snippet in Python using the iperf3 library to perform a stress test and measure network capacity:

```
import iperf3

client = iperf3.Client()
client.duration = 10  # test for 10 seconds
client.server_hostname = '192.168.0.1'  # IP address of server machine
result = client.run()

if result.error:
    print(result.error)
else:
    throughput = result.sent_bytes / result.elapsed_secs
    print(f"Network capacity: {throughput} bps")
```

3. Jitter: To measure jitter, we can calculate the difference in delay between each pair of adjacent packets in a data stream and then calculate the standard deviation of these differences. Here's an example code snippet in Python using numpy to calculate jitter:

```
import numpy as np

# read in log file with packet delay details
packet_delays = np.loadtxt('packet_delay_log.txt')

# calculate difference between adjacent packets
differences = np.diff(packet_delays)

# calculate standard deviation of differences
jitter = np.std(differences)
print(f"Jitter: {jitter} ms")
```

4. Packet delivery ratio (PDR) over time: To visualize how PDR changes over time, we can plot a graph of PDR vs time using a library like matplotlib. Here's an example code snippet in Python using pandas and matplotlib to create a PDR vs time plot:

```
import pandas as pd
import matplotlib.pyplot as plt

# read in log file with PDR details
df = pd.read_csv('pdr_log.csv')

# plot PDR vs time
plt.plot(df['timestamp'], df['pdr'])
plt.xlabel('Time')
plt.ylabel('PDR')
plt.show()
```

5. End-to-end delay: To calculate end-to-end delay, we can subtract the timestamp at which a packet was sent from the timestamp at which it was received. Here's an example code snippet in Python using pandas to calculate end-to-end delay:

```
import pandas as pd

# read in log file with packet transmission details
df = pd.read_csv('packet_transmission_log.csv')

# calculate end-to-end delay
df['e2e_delay'] = df['receive_time'] - df['send_time']
print(df['e2e_delay'])
```

6. Packet loss rate: To calculate packet loss rate, we can divide the number of packets lost by the total number of packets transmitted. Here's an example code snippet in Python using pandas to calculate packet loss rate:

```
import pandas as pd

# read in log file with packet transmission details
df = pd.read_csv('packet_transmission_log.csv')

# calculate packet loss rate
packet_loss_rate = df['packets_lost'].sum() / df['total_packets'].sum()
print(f"Packet loss rate: {packet_loss_rate}")
```

7. Network reliability: To measure network reliability, we can combine several performance metrics such as latency, packet loss rate, and PDR into a single score. Here's an example code snippet in Python using numpy to calculate network reliability:

```
import numpy as np

# read in log files with performance metrics
latency = np.loadtxt('latency_log.txt')
pdr = np.loadtxt('pdr_log.txt')
packet_loss_rate = np.loadtxt('packet_loss_log.txt')

# calculate network reliability score
reliability_score = (1 - pdr) * (1 - packet_loss_rate) * np.exp(-
np.mean(latency))
print(f"Network reliability score: {reliability_score}")
```

one of the other features that we can add to this simulator is adding packet tracing visualization by python libraries to trace a packet over a time. To add packet tracing visualization to this code, we need to modify the node class to include a variable for tracking the packets and their states. We can then use the matplotlib library to create a plot of the network topology and overlay the packet paths on top of it.
Here's an example implementation:

1. Modify the node class:

Add a new attribute "packets" that will be used to track the packets. This can be a list of dictionaries, where each dictionary represents a packet and its state (e.g., source node, destination node, TTL, current position).

```
class node:
    def __init__(self, ID, Xposition, Yposition):
        ...
        self.packets = []
```

2. Update the packet sending function in the node class:

Whenever a node sends a packet, add it to the "packets" list with its initial state. We can use a dictionary to represent the packet, and include attributes such as "source", "destination", "TTL", and "current_position".

```
def send_packet(self, dest_node):
    packet = {"source": self.ID, "destination": dest_node.ID, "TTL": NETWORK_TTL,
"current_position": self.ID}
    self.packets.append(packet)
```

3. Update the packet forwarding function in the node class:

Whenever a node forwards a packet, update its state in the "packets" list. This can be done by iterating over the list, finding the packet with the matching ID, and updating its attributes (e.g., decrementing its TTL, updating its position).

```
  def forward_packet(self, packet):

    packet["TTL"] -= 1
    packet["current_position"] = self.ID
    for p in self.packets:
```

```
        if p["source"] == packet["source"] and p["destination"] ==
packet["destination"]:
            p.update(packet)
```

4. Update the simulation loop:
After each time step, plot the network topology and overlay the packet paths on top of it. We can do this by iterating over all the nodes in the network, and plotting their positions as dots. Then iterate over all the packets in the "packets" list of each node, and plot a line connecting the current position of the packet to its destination.

```
for i in range(EXECUTION_TIME):
    ...
    # Plot network topology and packet paths
    fig = figure()
    ax = fig.add_subplot(1, 1, 1)
    for n in nodes:
        ax.plot(n.Xposition, n.Yposition, 'bo')
        for p in n.packets:
            if p["TTL"] > 0:
                dest_node = next((x for x in nodes if x.ID == p["destination"]),
None)
                if dest_node is not None:
                    ax.plot([n.Xposition, dest_node.Xposition], [n.Yposition,
dest_node.Yposition], 'k-')
    plt.savefig(f'packet_trace_{i}.png', dpi=200, bbox_inches='tight')
    plt.close(fig)
```

This will generate a sequence of images (one for each time step) showing the network topology and packet paths at that point in time. We can then use an external tool (e.g., ffmpeg) to combine the images into a video.

References:

1-https://en.wikipedia.org/wiki/Network_calculus

2-https://leboudec.github.io/netcal/

3-https://www.mdpi.com/1424-8220/20/12/3590