NetSim®

Accelerate Network R & D

Software Defined WSN Based Location-Aware Routing Protocol

Version 15.0

A Network Simulation & Emulation Software

By

TETCOS LLP

Contents

Software: NetSim Standard v15.0.21 (64 bit), Visual Studio 2022, and Python 3.11 or later.

Project download link:
https://github.com/NetSim-TETCOS/SDWSN_based_Location_Aware_Routing_Protocol_v15.0/archive/refs/heads/main.zip

Follow the instructions in the following link to download and set up the project in NetSim:
https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

1 Location-Aware Routing

Routing in an ad hoc wireless network must select forwarding nodes while links change with node position and radio reach. Location-Aware Routing (LAR) uses node coordinates to reduce route-search work and to select a next hop that advances a packet toward its destination.

1.1 Most Forward within Fixed Radius (MFR)

Most Forward within Fixed Radius is a geographic routing method. At each hop, MFR examines neighbors inside the configured transmission radius and chooses the node that provides the greatest projected progress along the line from the current node to the destination. A node already present in the route is skipped.

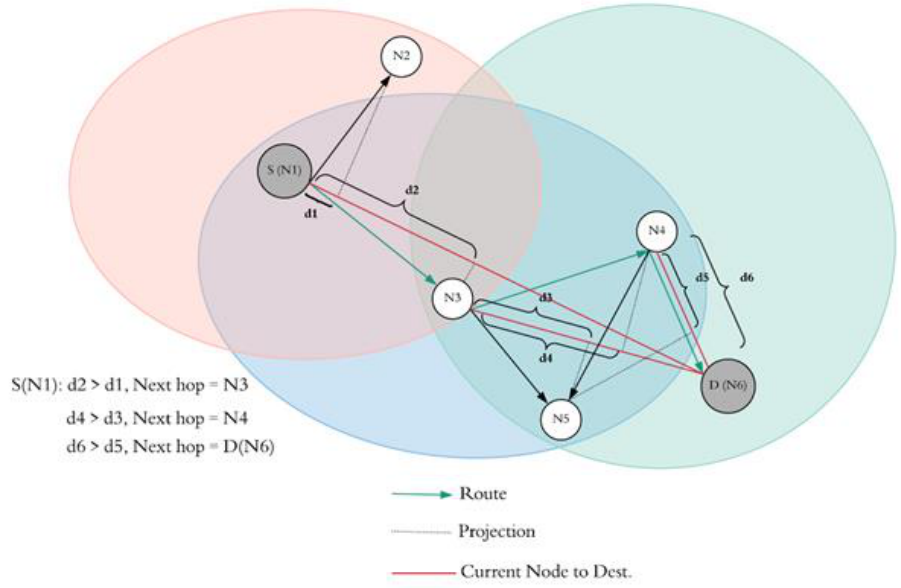

Figure 1-1: *MFR next-hop selection*

For Figure 1-1:

- $S(N1)$ is the source and $D(N6)$ is the destination.
- $N2$ and $N3$ are inside the transmission radius of $S(N1)$.
- d_1 and d_2 are the projected distances of $N2$ and $N3$ on the line from $S(N1)$ to $D(N6)$.

- Since $d_2 > d_1$, $N3$ is selected as the next hop.
- $N4$, $N5$, and $S(N1)$ are inside the transmission radius of $N3$. The source is skipped because it is already present in the route.
- d_3 and d_4 are the projected distances of $N5$ and $N4$. Since $d_4 > d_3$, $N4$ is selected.
- $N5$, $D(N6)$, and $N3$ are inside the transmission radius of $N4$. Node $N3$ is skipped because it is already present in the route.
- Since $d_6 > d_5$, the destination is selected as the next hop.
- The resulting route is $S(N1) \rightarrow N3 \rightarrow N4 \rightarrow D(N6)$.

2 NetSim–Python Run-Time Interaction

NetSim can accept commands during simulation through its run-time interaction socket. In this project, NetSimCore acts as the server and `mfrProtocol.py` acts as the client.

2.1 Python socket interface

Run-time interaction must be enabled before starting the With-SDN simulation. NetSimCore then listens for the Python client on port 8999. After the connection is established, the Python program calculates the routes and sends static route commands to NetSim.

The workflow is:

1. The modified Application library writes node coordinates and IP addresses to `Device_log.txt`.
2. It writes application ID, source, and destination data to `Appinfo_log.txt`.
3. The Python script reads both files and computes each MFR route.
4. The script connects to `127.0.0.1:8999`.
5. It sends route-add commands to the NetSim run-time interaction server.

2.2 Python script and input files

The socket client and MFR calculation are implemented in `mfrProtocol.py`. The calculation uses the following functions:

- `_distance()` calculates the distance between two nodes.
- `_area()` calculates the area of the triangle formed by the current node, a neighboring node, and the destination.
- `_height()` calculates the perpendicular distance used for projection.
- `_projDist()` calculates the forward projected distance.

`Device_log.txt` contains one device per line:

```
SINK      76.70   76.71   192.168.0.1
SENSOR_2  15.00  160.00   192.168.0.2
```

`Appinfo_log.txt` contains the application ID, source name, and destination name:

```
1  SENSOR_2   SENSOR_3
2  SENSOR_12  SENSOR_7
```

The Python script uses a transmission radius of 170 m:

```
tx = 170.00  # transmission radius in metres
```

Change this value when radio properties, channel settings, or node placement change.

For each route hop, the script sends a command in the following form:

```
<DEVICE> route ADD <DESTINATION_IP> MASK 255.255.255.255
         <NEXT_HOP_IP> METRIC 1 IF 1
```

3 Project Setup

1. Open the project workspace in NetSim v15.0.
2. Confirm that the **WITH_SDN** and **WITHOUT_SDN** experiments are listed under the workspace.
3. Copy `mfrProtocol.py` into:

```
<Workspace Path>\bin_x64\Python\
```

The Application library writes the two input files into the same folder. The required project files are:

```
<Workspace Path>\
  src\Simulation\Application\Application.c
  bin_x64\Python\mfrProtocol.py
  bin_x64\Python\Device_log.txt
  bin_x64\Python\Appinfo_log.txt
```

4 Example Network

The scenario contains 12 wireless sensors and one WSN sink.

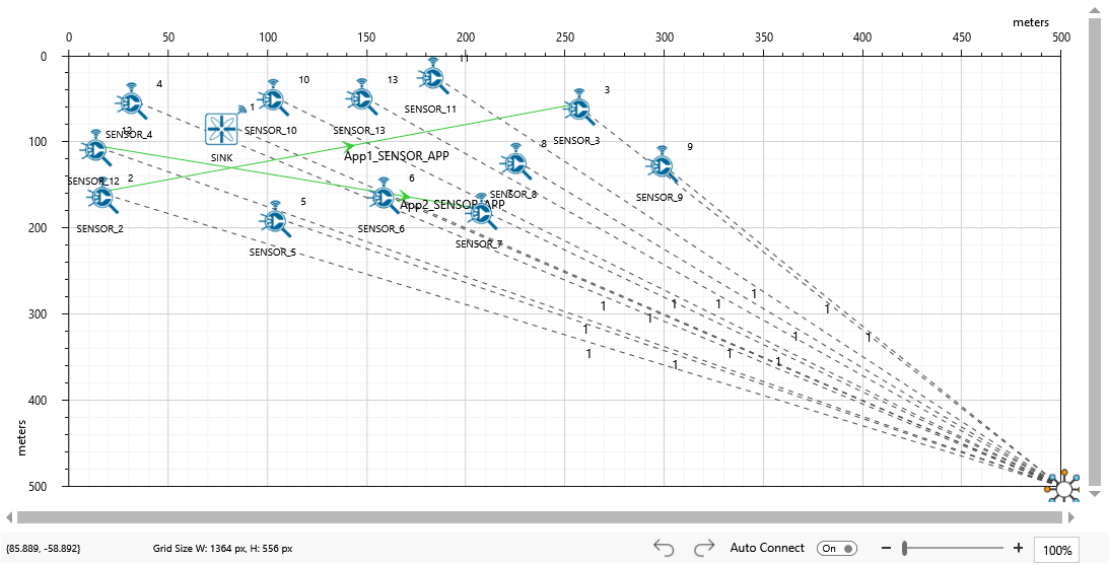

Figure 4-1: *WSN simulation scenario*

Table 4-1: *Application properties*

Application	Source ID	Destination ID	Transport protocol
1	2	3	UDP
2	12	7	UDP

Use the following scenario settings:

- Network-layer routing protocol: DSR on every wireless sensor and the WSN sink.
- Channel model: Log-distance path loss with path-loss exponent 2.
- Simulation time: 500 seconds.
- Packet trace: Enabled.
- With-SDN run-time interaction: True.
- Without-SDN run-time interaction: False.

5 Simulation Procedure

1. In NetSim, open **Options** and select **Run Time Interaction**.
2. Set **Interactive Simulation** to **True**, then select **OK**.

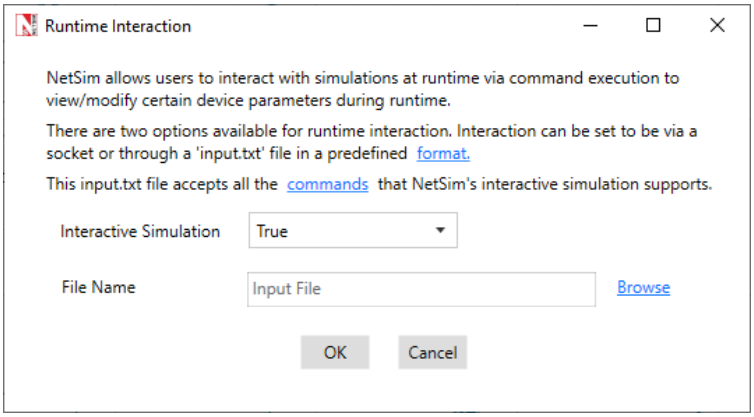

Figure 5-1: *Run-time interaction setting*

3. Start the 500-second simulation.
4. Wait until the simulation console reports `Waiting for client....`

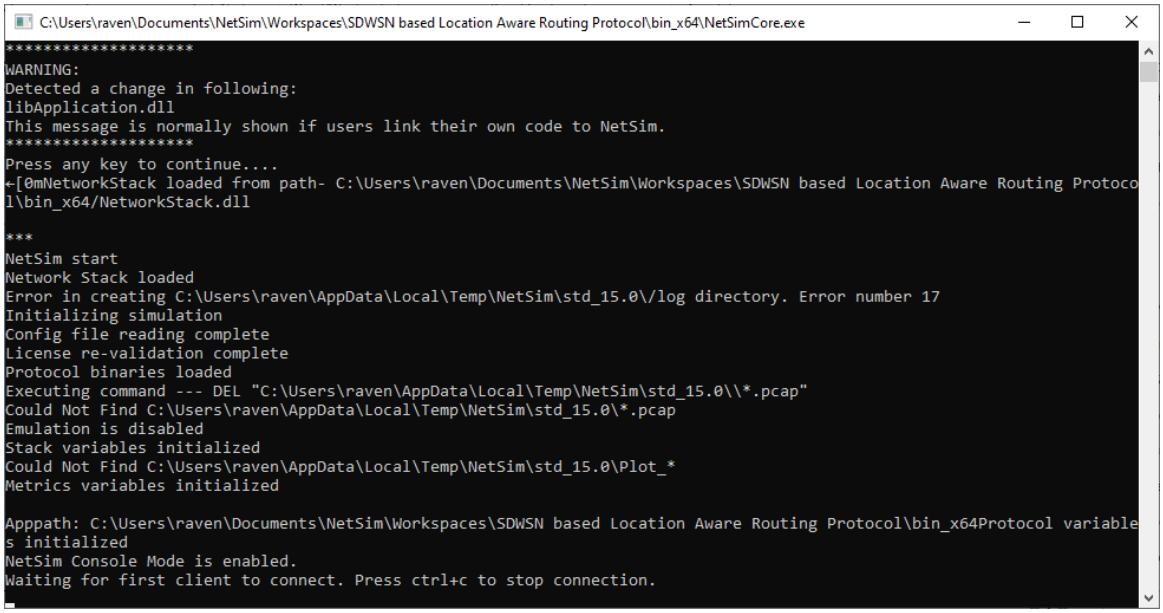

Figure 5-2: *NetSim simulation console waiting for the Python client*

5. Open PowerShell or Command Prompt and run:

```
cd "<Workspace Path>\bin_x64\Python"
python mfrProtocol.py
```

If `python` is not on `PATH`, use the installed interpreter:

```
"C:\Path\To\Python311\python.exe" mfrProtocol.py
```

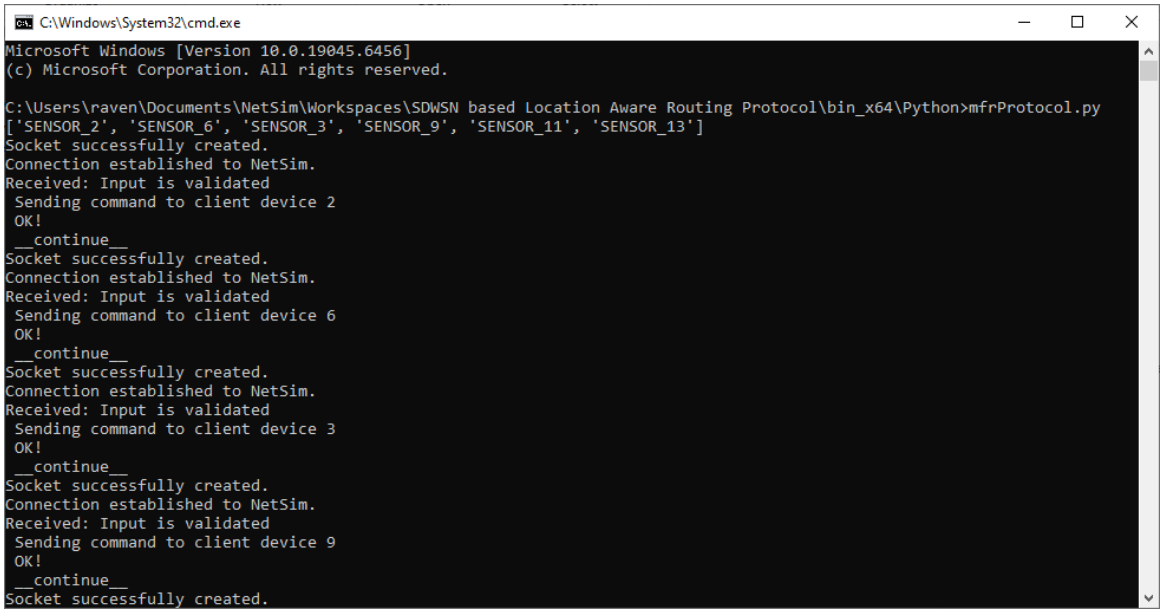

Figure 5-3: *MFR Python program connected to NetSim*

The MFR script computes the following routes:

- Application 1: `SENSOR_2 -> SENSOR_6 -> SENSOR_3`
- Application 2: `SENSOR_12 -> SENSOR_13 -> SENSOR_7`

6 Results

6.1 Route table verification

Open the Results Dashboard and select the route table for each forwarding device. The With-SDN run must contain the following host routes:

Table 6-1: *Static route entries installed by the Python program*

Device	Destination	Gateway	Interface
SENSOR_2	192.168.0.3	192.168.0.6	192.168.0.2
SENSOR_6	192.168.0.3	192.168.0.3	192.168.0.6
SENSOR_12	192.168.0.7	192.168.0.13	192.168.0.12
SENSOR_13	192.168.0.7	192.168.0.7	192.168.0.13

For Application 1, SENSOR_2 forwards packets for 192.168.0.3 to SENSOR_6. SENSOR_6 then forwards the packets to SENSOR_3. For Application 2, SENSOR_12 forwards packets for 192.168.0.7 to SENSOR_13, which forwards them to SENSOR_7.

6.2 Packet trace verification

Packet Trace must be enabled before the simulation starts. After the simulation:

1. Open **Packet Trace** from the Results Dashboard.
2. Filter the `CONTROL_PACKET_TYPE/APP_NAME` column for the required application.
3. Filter a value in the `PACKET_ID` column.
4. Read the `TRANSMITTER_ID` and `RECEIVER_ID` columns in time order to identify the route.

The packet trace shows the following routes.

Table 6-2: *Observed application routes*

Case	Application	Observed route
With SDN	1	`SENSOR_2 -> SENSOR_6 -> SENSOR_3`
With SDN	2	`SENSOR_12 -> SENSOR_13 -> SENSOR_7`
Without SDN	1	`SENSOR_2 -> SENSOR_13 -> SENSOR_3`; some packets used `SENSOR_2 -> SENSOR_12 -> SENSOR_13 -> SENSOR_3`.
Without SDN	2	`SENSOR_12 -> SENSOR_13 -> SENSOR_3 -> SENSOR_7`

The With-SDN case reduces the steady route to two hops for both applications. The Without-SDN DSR case uses a three-hop route for Application 2 and changes between two-hop and three-hop routes for Application 1.

6.3 Application metrics

Tables 6-3 and 6-4 show the application metrics from the two 500-second simulations.

Table 6-3: *Application metrics without SDN*

Application	Throughput (Mbps)	Delay (μs)
App1_SENSOR_APP	0.0003984	29,898.57
App2_SENSOR_APP	0.0004040	26,983.35

Table 6-4: *Application metrics with SDN*

Application	Throughput (Mbps)	Delay (μs)
App1_SENSOR_APP	0.0004192	27,607.03
App2_SENSOR_APP	0.0004080	21,303.51

Table 6-5: *Measured change from Without SDN to With SDN*

Application	Delay reduction
App1_SENSOR_APP	7.66%
App2_SENSOR_APP	21.05%

The With-SDN run has lower delay and higher throughput for both applications. Application 1 delay decreases by 7.66%, while Application 2 delay decreases by 21.05%.

7 Application Source Modification

Add the following file-generation code in `fn_NetSim_Application_Init()` before the Application library initialization calls.

```
_declspec(dllexport) int fn_NetSim_Application_Init(
    struct stru_NetSim_Network* NETWORK_Formal,
    NetSim_EVENTDETAILS* pstruEventDetails_Formal,
    char* pszAppPath_Formal,
    char* pszWritePath_Formal,
    int nVersion_Type,
    void** fnPointer)
{
    FILE* fp = NULL;
    int i = 0;
    char fileName[BUFSIZ];
    ptrAPPLICATION_INFO* appInfo =
        (ptrAPPLICATION_INFO*)NETWORK_Formal->appInfo;

    snprintf(fileName, sizeof fileName,
             "%s\\Python\\Device_log.txt", pszAppPath_Formal);
    fp = fopen(fileName, "w");
    if (fp)
    {
        for (i = 0; i < NETWORK_Formal->nDeviceCount; i++)
        {
            fprintf(fp, "%s\t%.2lf\t%.2lf\t%s\n",
                    DEVICE_NAME(i + 1),
                    DEVICE_POSITION(i + 1)->X,
                    DEVICE_POSITION(i + 1)->Y,
                    DEVICE_NWADDRESS(i + 1, 1)->str_ip);
        }
        fclose(fp);
    }

    snprintf(fileName, sizeof fileName,
             "%s\\Python\\Appinfo_log.txt", pszAppPath_Formal);
    fp = fopen(fileName, "w");
    if (fp)
    {
        for (i = 0; i < NETWORK_Formal->nApplicationCount; i++)
        {
            fprintf(fp, "%d\tSENSOR_%d\tSENSOR_%d\n",
                    appInfo[i]->id,
                    appInfo[i]->sourceList[0],
                    appInfo[i]->destList[0]);
        }
        fclose(fp);
    }

    init_application_generation_log();
    init_application_log();
    update_latency();
```

```
    return fn_NetSim_Application_Init_F();
}
```