**Software Defined WSN based Location Aware Routing Protocol**

**Software:** NetSim Standard v14.1, Visual Studio 2022, Python 3.11 and later

## Project Download Link:

<https://github.com/NetSim-TETCOS/SDWSN-based-Location-Aware-Routing-Protocol_v13.3/archive/refs/heads/main.zip>

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

## Location Aware Routing (LAR)

Routing for an ad-hoc wireless network is challenging, many routing strategies have been proposed in the literature. With the availability of affordable Global Position System equipped devices, Location-Aware Routing provides a promising foundation for developing an efficient and practical solution for routing in the ad-hoc wireless network.

## Most Forward within Fixed Radius R (MFR)

MFR protocol is a geographic Location-Aware Routing protocol. MFR forwards packets to the neighbor nodes within a set radius of the current node (not the route source) that makes the most forward progress (or the least backward progress) along the line drawn from the current node to the destination. Progress is calculated as the cosine of the distance from the current node to the neighbor node projected back onto the line from the current node to the destination.

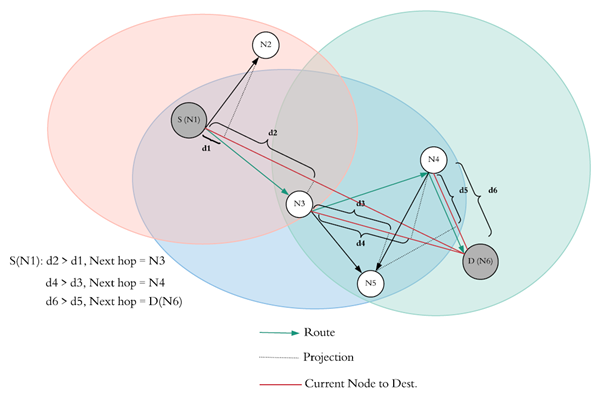


Figure 1: MFR Protocol Implementation

Here,

* S(N1) is the source node and D(N6) is the destination node.
* N2 and N3 are in the transmission radius of S(N1).
* So, according to MFR protocol, d1 and d2 are the projected distances of N2 and N3 respectively on the line drawn from the current node i.e., S(N1) and the destination node D(N6):
* d2 > d1, therefore the next route hop node will be N3.
* N4, N5 and S(N1) are in the transmission radius of N3. Since S (N1) is already present in the route list, skip it.
* So, according to MFR protocol, d3 and d4 are the projected distances of N5 and N4 respectively on the line drawn from the current node i.e., N3 and the destination node D(N6):
* d4 > d3, therefore the next route hop node will be N4.
* N5, D(N6) and N3 are in the transmission radius of N4. Since N3 is already present in the route list, skip it.
* So, according to MFR protocol, d5 and d6 are the projected distances of N5 and D(N6) respectively on the line drawn from the current node i.e., N4 and the destination node D(N6):
* d6 > d5, therefore the next route hop node will be D(N6).
* Route according to MFR: S(N1) -> N3 -> N4 -> D(N6).

## Real Time Interaction in NetSim

NetSim allows users to interact with the simulation at runtime via a socket or through a file. User Interactions make simulation more realistic by allowing command execution to view/modify certain device parameters during runtime.

## Python socket interface

* Python interfacing is a method to interface custom protocols like routing-based protocols with the NetSim engine.
* In this project, we input NetSimCore.exe with routes generated via our routing protocol i.e., Most Forward within Fixed Radius R (MFR) which is a geographic location-aware routing protocol. The interaction between the routing protocol and the NetSimCore.exe is happening via socket programming.
* The Real-Time Interaction has to be turned ‘True’ before running the simulation of the scenario. This lets the NetSimCore.exe (server) to wait for the client (Python script) to connect using the socket port.
* After the connection is established, we compute the routes based on our custom MFR protocol. These routes are passed as static routes to the NetSimCore.exe server by the python script.

## Python Script

The Socket programming code and MFR protocol code has written only in one separate file (mfrProtocol.py). The protocols are written in a separate script file like here **mfrProtocol.py:**

* This python script reads the device coordinate and device ip address input from a file device\_log.txt having data in the following format:

SINK 76.70 76.71 11.1.1.1

* The protocol script has 4 functions to ultimately find the projected distance \_projDist() on the line drawn from the current node to the destination.
* Mention the Device\_log.txt file name in the python script at File I/P section:

**with open('Device\_log.txt','r') as f:**

* This python script reads the **Application ID, Source ID and Destination ID** input from a file Appinfo\_log.txt having data in the following format:

**1SENSOR\_2 SENSOR\_3**

* Mention the Appinfo\_log.txt file name in the python script at File for Appinfo section:

**with open(‘Appinfo\_log.txt’,'r') as f:**

* In the Declarations of MFR, change the Transmission range (meters) accordingly:

**Tx = 170**

**Note:** The Transmission range is set to 170 based on the channel conditions and device properties for this example. This may vary if any network other than the one discussed in this example is considered.

## Example:

1. The **SDWSN\_MFR\_LAR\_Workspace\_v14.1** comes with a sample network configuration that are already saved. To open this example, go to Your work in the home screen of NetSim and click on the **WITH**\_**SDN** from the list of experiments.
2. The saved network scenario consists of
3. 12 Wireless Sensor
4. 1 WSN Sink

A screenshot of a computer

Description automatically generated

Figure 2: WSN Network Topology

1. Application Properties

|  |  |
| --- | --- |
| Application Properties |  |
| For Application 1  Source ID  Destination ID | 2  3 |
| For Application 2  Source ID  Destination ID | 12  7 |
| Transport Layer Protocol | UDP |

Table 1: Application Properties

1. Set Network layer protocol to DSR in both Wireless sensor and WSN Sink Node.
2. Channel Characteristics: No Pathloss
3. Run the Simulation for 500sec.

## Results and discussion

* Upon running simulations with this configuration, Route from source to destination is as shown below:

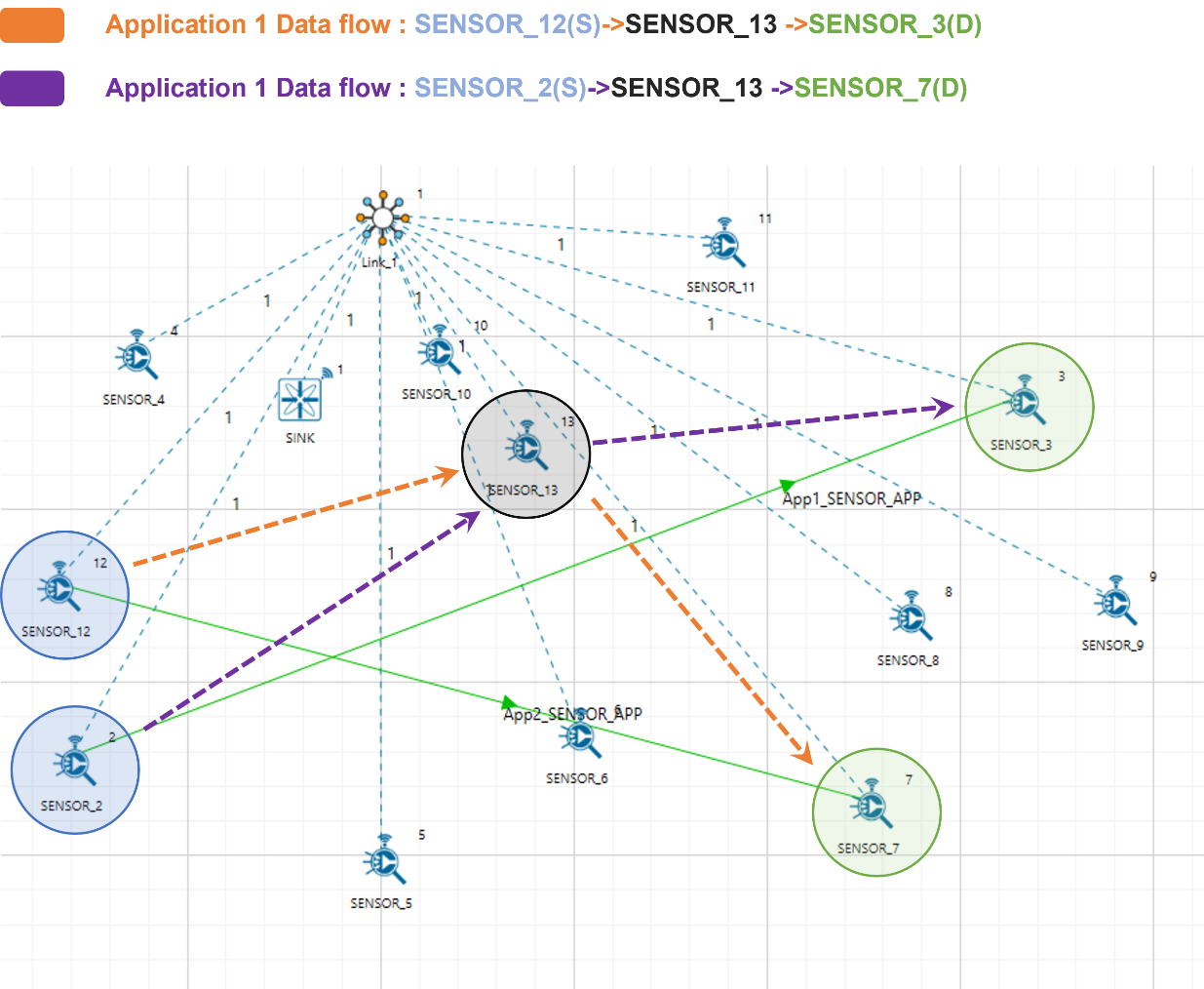


Figure 3: Application packet flow in the Network Topology

## Procedure to perform routing using python interface in NetSim

* For the python interface to interact with NetSim during the simulation, Interactive Simulation parameters must be set to 'True' under the Real-Time Interaction tab, before running the simulation.
* Click on **Options** tab and select Run Time Interaction option

A screenshot of a computer

Description automatically generated

Figure 4: Run time Interaction tab set Interactive Simulation as True

* In the Run time Interaction tab, Interactive Simulation option is set to **True** and click on **OK**

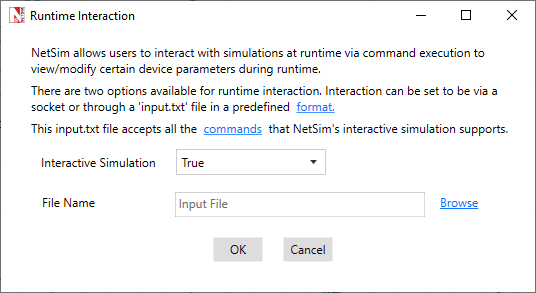


Figure 5: Interactive Simulation parameters set as TRUE

* This lets the NetSimCore.exe (server) to wait for the client (Python script) to connect using the socket port. After the connection is established, we compute the routes based on our custom MFR protocol. These routes are passed as static routes to the NetSimCore.exe server by the python script.
* Run simulation for 500 seconds. NetSim Simulation Console starts and “waiting for client to connect” and press any key to Continue.
* The MFR protocol and socket client code to connect to NetSimCore.exe is written in **mfrProtocol.py.**
* Open Command Prompt in the directory (Ex: **<Workspace Path>\bin\_x64\Python)** where the python codes are present and run the command python **mfrProtocol.py**

A black screen with white text

Description automatically generated

Figure 6: Run Python mfrProtocol.py using cmd prompt.

* Python interface interacts with NetSim Simulation and routes the packets from source to destination based on MFR protocols.

A screenshot of a computer

Description automatically generated

Figur*e* 7: Python interface interacts with NetSim Simulation

* Simulation continues and packets are routed from source to destination based on MFR protocol as shown below:
  + Application 1: SENSOR\_2(S)->SENSOR\_13->SENSOR\_3(D)
  + Application 2: SENSOR\_12(S)->SENSOR\_13->SENSOR\_7(D)

## Analyzing the device route tables in NetSim Results Dashboard

* NetSim Results Window contains route tables for each device from which we can identify the routes updated by the python interface as per MFR protocol. Since the route that is formed is from SENSOR\_2(S) -> SENSOR\_13 -> SENSOR\_3(D), route entries for packets with destination 192.168.0.3 are added in the nodes SENSOR\_2, SENSOR\_13, and SENSOR\_3 to forward packets to SENSOR\_13 and SENSOR\_3 respectively. In the nodes SENSOR\_2, SENSOR\_13, and SENSOR\_3 static route entries added based on MFR protocol by the python socket program can be found as shown below:

A screenshot of a computer

Description automatically generated

Figure 8: Route table for Wireless Sensor 2

The static route entry for SENSOR\_2 specifies the next hop as SENSOR\_13 which has the IP 192.168.0.13.

A screenshot of a computer

Description automatically generated

Figure 9: Route table for Wireless Sensor 13

The static route entry for SENSOR\_13 specifies the next hop as the destination node SENSOR\_3 which has the IP 192.168.0.3.

## Using NetSim Packet Trace to identify the route taken by packets from the source to the destination.

NetSim Packet trace log file can be obtained by enabling the packet trace option in NetSim GUI before running the simulation.

Upon running simulation with packet trace enabled, the packet trace log file can be accessed from the NetSim Results Window using the Open Packet Trace link.

Once the packet trace log file is loaded you can filter a specific packet id in the PACKET\_ID column to view the path that the packet has taken.

Upon filtering Packet with id 4 we can observe the following in the packet trace:



Figure 10: NetSim Packet Trace

### Case 1: Without SDN

A white background with black text

Description automatically generated

Figure 11: Application Metrics Table for Without SDN

### Case 2: With SDN

A close-up of a list

Description automatically generated

Figure 12: Application Metrics Table for with SDN

You can see from the Application Metrics table that in case 2, for creating route path the delay is less as compared in case 1.

**Note:** *Code Modifications are highlighted in red color.*

**Changes to fn\_NetSim\_Application\_Init(), in Application.c file, within Application project**

/\*\*

This function is used to initialize the parameter for all the application based on

the traffic type

\*/

\_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;

int i = 0;

char f\_name[BUFSIZ];

sprintf(f\_name, "%s\\%s\\%s", pszAppPath,"Python", "Device\_log.txt");

fp = fopen(f\_name, "w+");

if (fp)

{

for (i = 0; i < NETWORK->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);

}

fprintf(stderr, "\nApppath: %s", pszAppPath);

sprintf(f\_name, "%s\\%s\\%s", pszAppPath,"Python", "Appinfo\_log.txt");

fp = fopen(f\_name, "w+");

if (fp)

{

//APP\_CALL\_INFO\* info = appInfo[packet->pstruAppData->nApplicationId - 1]->appData;

ptrAPPLICATION\_INFO\* appInfo = (ptrAPPLICATION\_INFO\*)NETWORK->appInfo;

for (i = 0; i < NETWORK->nApplicationCount; i++)

fprintf(fp, "%d\tSENSOR\_%d\tSENSOR\_%d\n", appInfo[i]->id, appInfo[i]->sourceList[0], appInfo[i]->destList[0]);

fclose(fp);

}

return fn\_NetSim\_Application\_Init\_F();

}