

# SDWSN based Location Aware Routing Protocol

---

**Software Recommended:** NetSim Standard v12.2 (32-bit/ 64-bit), Visual Studio 2017/2019

**Reference:** <https://ieeexplore.ieee.org/document/9118046>

Follow the instructions specified in the following link to clone/download the project folder from GitHub using Visual Studio:

<https://tetcos.freshdesk.com/support/solutions/articles/14000099351-how-to-clone-netsim-file-exchange-project-repositories-from-github->

Other tools such as GitHub Desktop, SVN Client, Sourcetree, Git from the command line, or any client you like to clone the Git repository.

**Note:** It is recommended not to download the project as an archive (compressed zip) to avoid incompatibility while importing workspaces into NetSim.

**Secure URL for the GitHub repository:**

**[https://github.com/NetSim-TETCOS/SDWSN-based-Location-Aware-Routing-Protocol\\_v12.2.git](https://github.com/NetSim-TETCOS/SDWSN-based-Location-Aware-Routing-Protocol_v12.2.git)**

## **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.

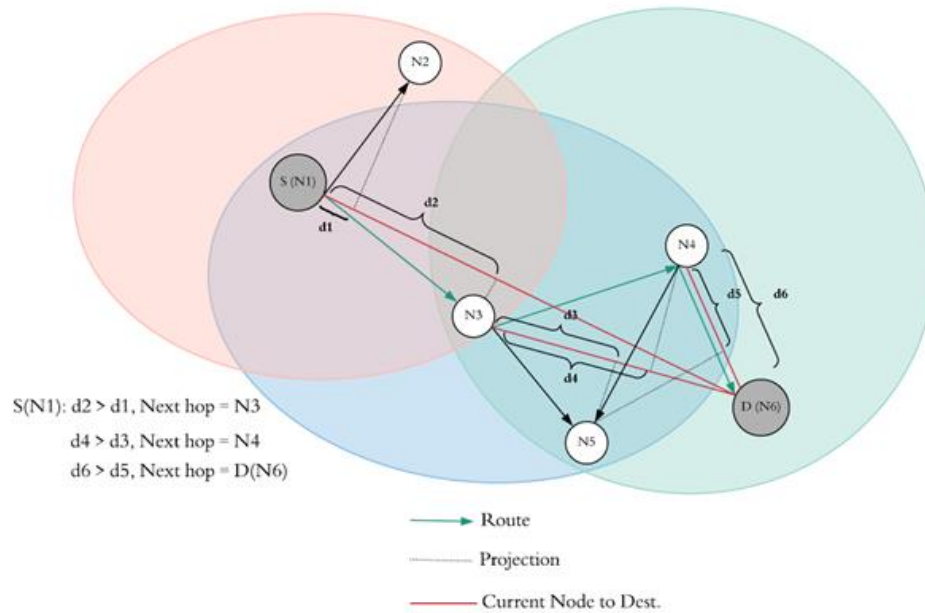


Figure: MFR Protocol Implementation

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  $N3$ . 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  $D(N6)$ :

- $d_6 > d_5$ , 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 are together part of mfrProtocol.py python script file.

### 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.
- Reads the device\_log.txt file written by NetSim:  
**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 written by NetSim, which has having data in the following format:  
  

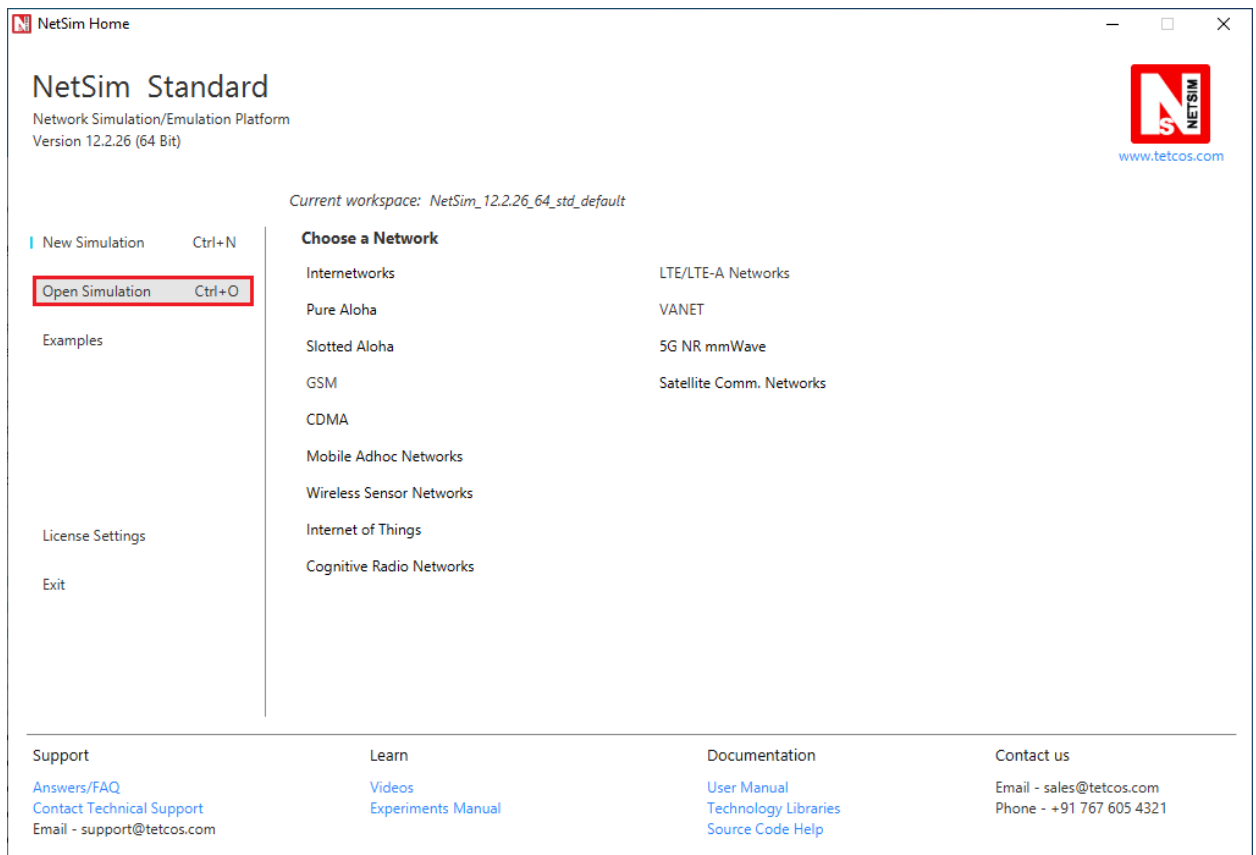
```
1      SENSOR_2      SENSOR_3
```
- Mention the Appinfo\_log.txt file name in the python script at File for Appinfo section:  
**with open('Appinfo\_log.txt','r') as f:**
- In the Declarations of MFR, change the Transmission range (meters) accordingly:

- **Tx = 170**

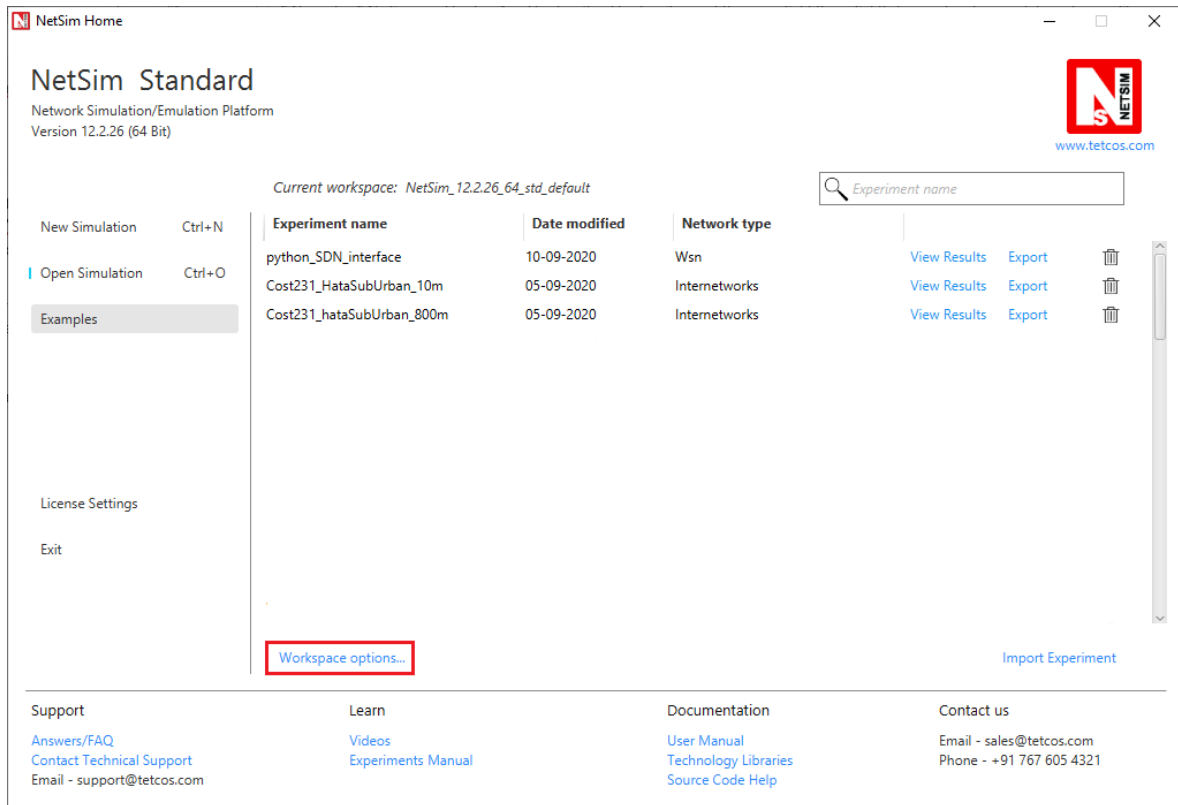
**Note:** The Transmission range is set to 170 based on the channel conditions and device properties for this example. This may vary if any network other than the one discussed in this example is considered.

Steps:

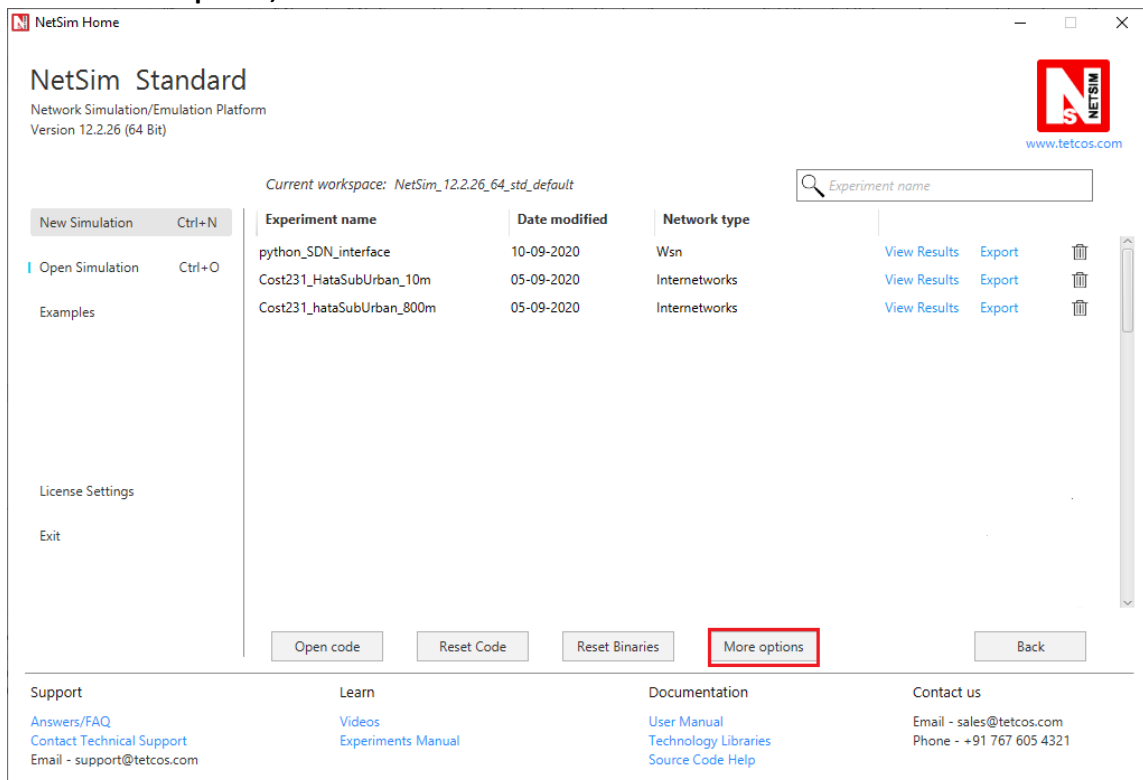
1. After you unzip the downloaded project folder, Open NetSim Home Page click **on Open Simulation** option,



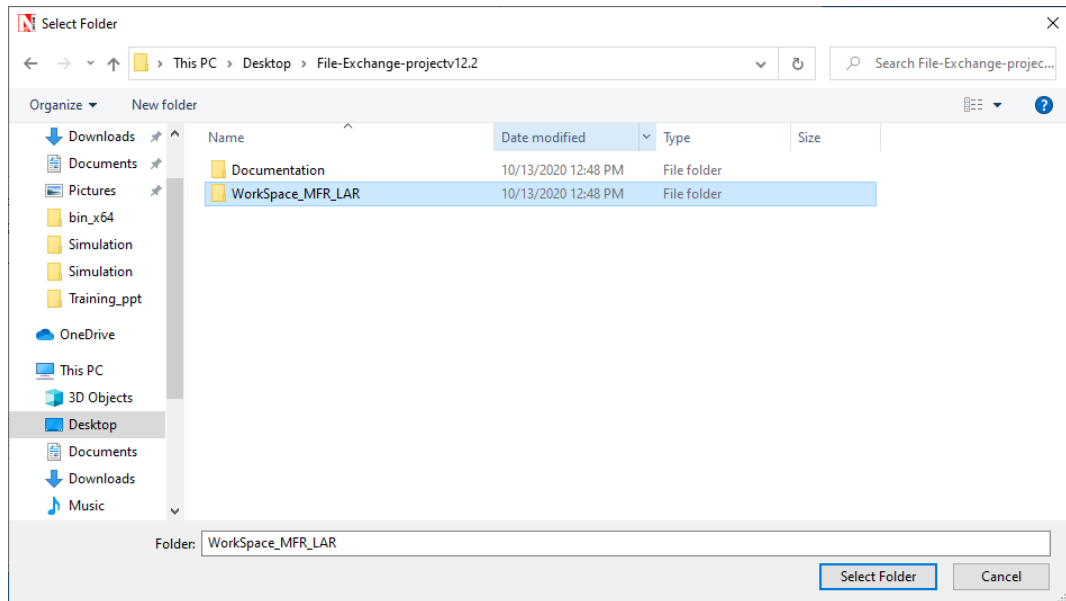
2. Click on **Workspace options**



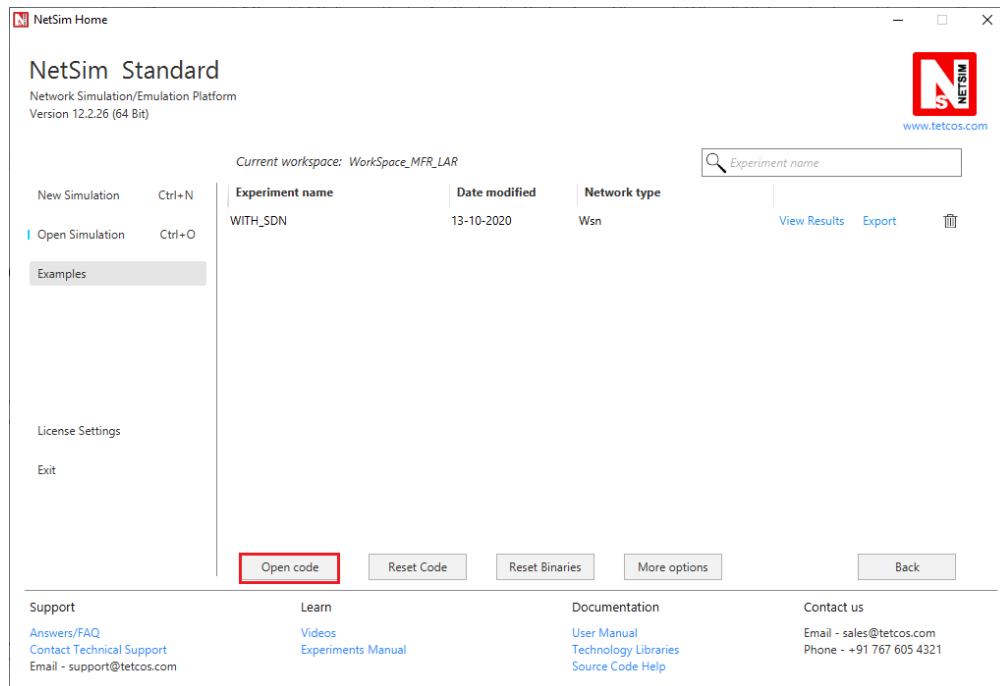
### 3. Click on **More Options**,



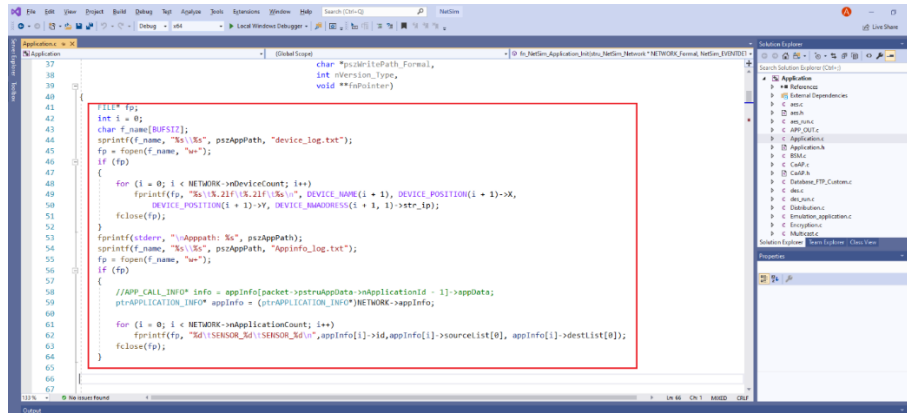
- Click on **Import**, browse the extracted folder path and go into the `WorkSpace_MFR_LAR` directory. Click on **Select folder** button and then on **OK**.



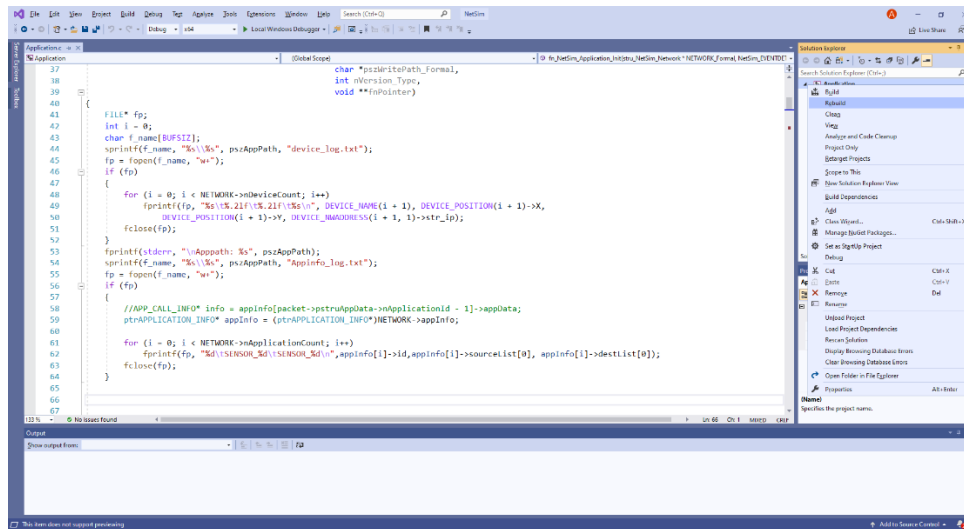
- Go to home page, Click on **Open Simulation >Workspace options** and click on the **Open code** button.



- Add the code that is highlighted in `Application.c` file



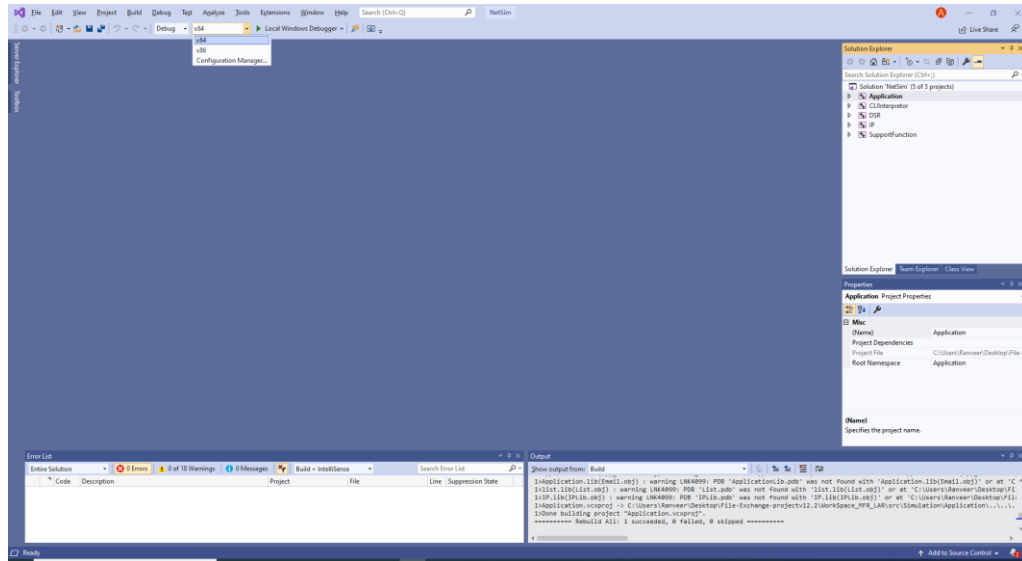
7. Now right click on Solution explorer and select Rebuild.



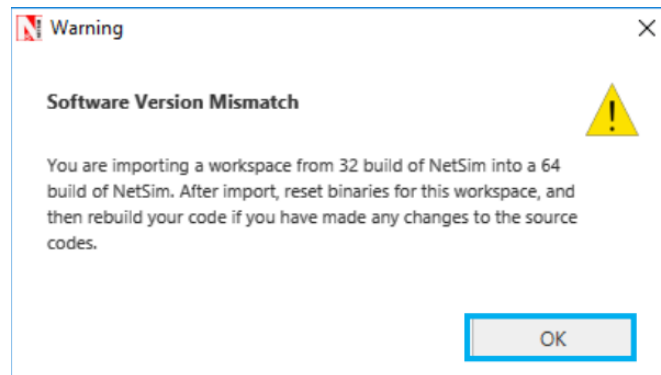
8. Upon rebuilding, **libApplication.dll** will automatically get replaced in the respective bin folders of the current workspace

**Note:**

1. Based on whether you are using NetSim 32 bit or 64-bit setup you can configure Visual studio to build 32 bit or 64 bit DLL files respectively as shown below:

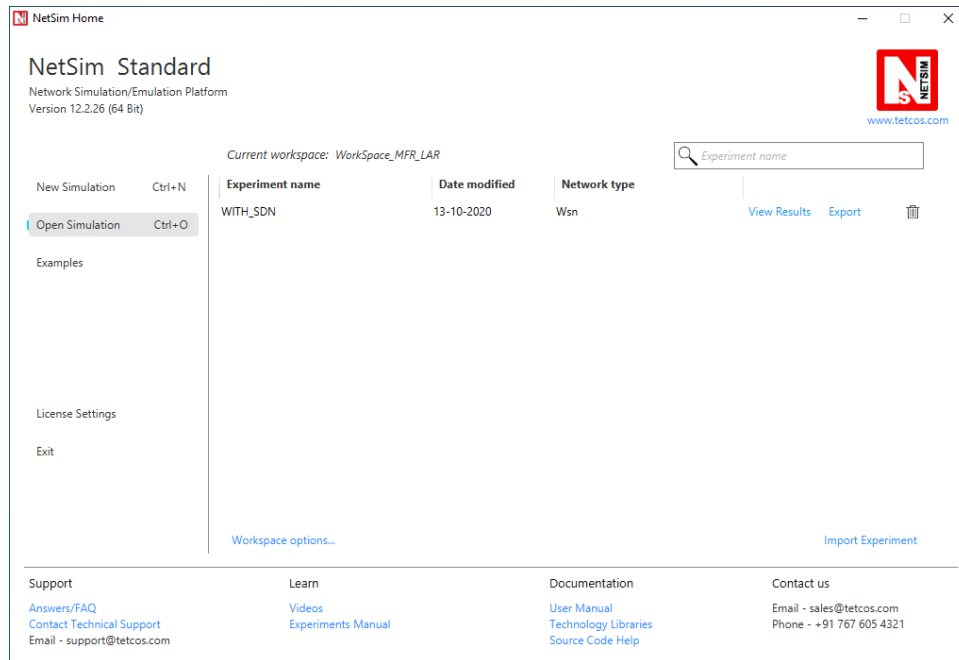


2. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



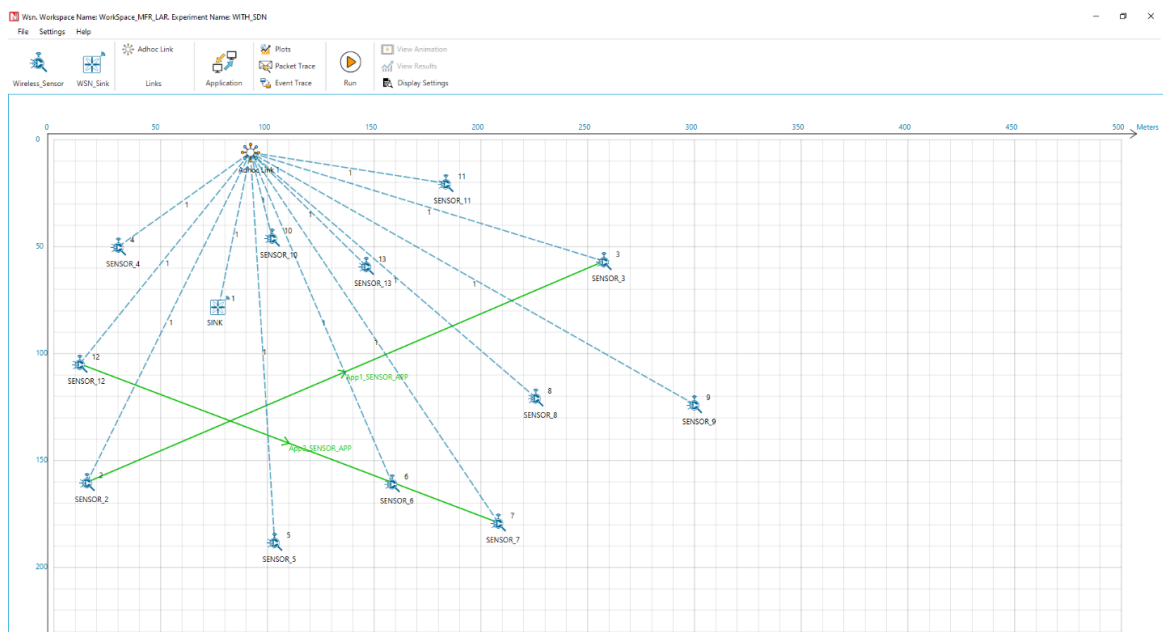
9. Go to NetSim home page, click on **Open Simulation**, Click on **WITH\_SDN**.





## Settings that were done to create the network scenario for Location Aware Routing Protocol:

1. Create a network scenario in WSN (Wireless Sensor Network) with UDP running in the Transport Layer and DSR in Network Layer.
2. For example, you can create a scenario as shown in the following screenshot:



### For Application 1:

- Source – Device id 2
- Destination – Device id 3

#### For Application 2:

- Source – Device id 12
- Destination – Device id 7

#### Link Properties (Adhoc Link 1)

Channel Characteristics – Path Loss only

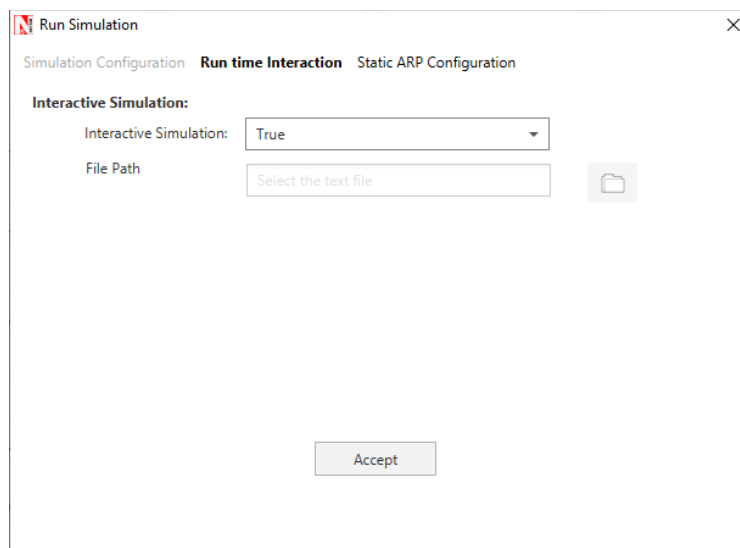
Path Loss model – LOG DISTANCE

Path Loss Exponent- 2

3. Run the Simulation for 500sec.
4. Upon running simulations with this configuration, Route from source to destination as shown below:
  - a. Application 1: SENSOR\_2(S)->SENSOR\_13->SENSOR\_3(D)
  - b. Application 2: SENSOR\_12(S)->SENSOR\_13->SENSOR\_7(D)

#### Procedure to perform routing using python interface in NetSim:

- For the python interface to interact with NetSim during the simulation, Interactive Simulation parameters has to be set to 'True' under the Real-Time Interaction tab, before running the simulation.



- 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 waits for client application to connect as shown below:

```

Select C:\Users\Ranveer\Documents\Workspace_MFR_LAR\bin\bin_x64\NetSimCore.exe
Heartbeat status = 0 (0 indicates successful)
+{33m-[1m
*****
WARNING:
Detected a change in following:
libApplication.dll
This message is normally shown if users link their own code to NetSim.
*****
Press any key to continue....
+{0mNetworkStack loaded from path- C:\Users\Ranveer\Documents\Workspace_MFR_LAR\bin\bin_x64\NetworkStack.dll
***
NetSim start
Network Stack loaded
Error in creating C:\Users\Ranveer\AppData\Local\Temp\NetSim\std12.2.26_x64\log directory. Error number 17
Initializing simulation
Config file reading complete
License re-validation complete
Protocol binaries loaded
Stack variables initialized
Could Not Find C:\Users\Ranveer\AppData\Local\Temp\NetSim\std12.2.26_x64\Plot_*
Metrics variables initialized
Apppath: C:\Users\Ranveer\Documents\Workspace_MFR_LAR\bin\bin_x64\Protocol variables initialized
Executing command --- DEL "C:\Users\Ranveer\AppData\Local\Temp\NetSim\std12.2.26_x64\*.pcap"
Could Not Find C:\Users\Ranveer\AppData\Local\Temp\NetSim\std12.2.26_x64\*.pcap
Emulation is disabled
NetSim Console Mode is enabled.
Waiting for first client to connect. Press ctrl+c to stop connection.

```

- The MFR protocol and socket client code to connect to NetSimCore.exe is written in **mfrProtocol.py**.
- Open Command Prompt in the directory where the python codes are present and run the command **python mfrProtocol.py**

```

Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Ranveer\Desktop\File-Exchange-projectv12.2\Workspace_MFR_LAR\bin\bin_x64>python mfrProtocol.py

```

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

```

C:\Windows\System32\cmd.exe
['SENSOR_2', 'SENSOR_13', 'SENSOR_3']
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 2
OK!
__continue__
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 13
OK!
__continue__
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 3
OK!
__continue__
['SENSOR_12', 'SENSOR_13', 'SENSOR_7']
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 12
OK!
__continue__
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 13
OK!
__continue__
Socket successfully created.
Connection established to NetSim.
Received: Input is validated
Sending command to client device 7
OK!
__continue__

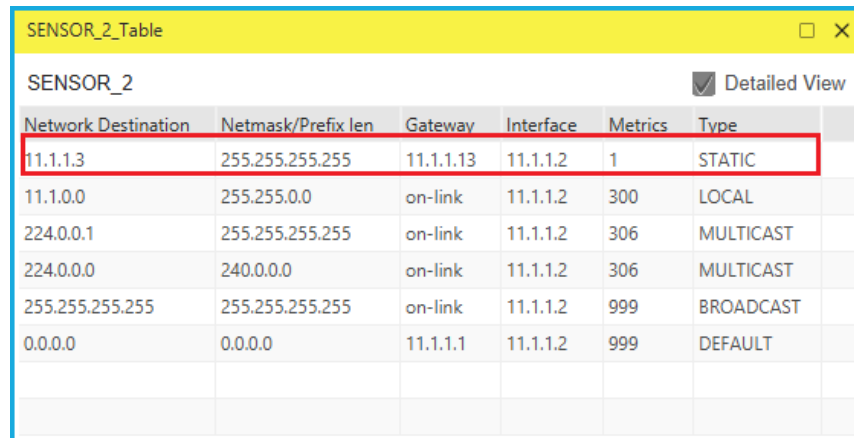
```

- 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)

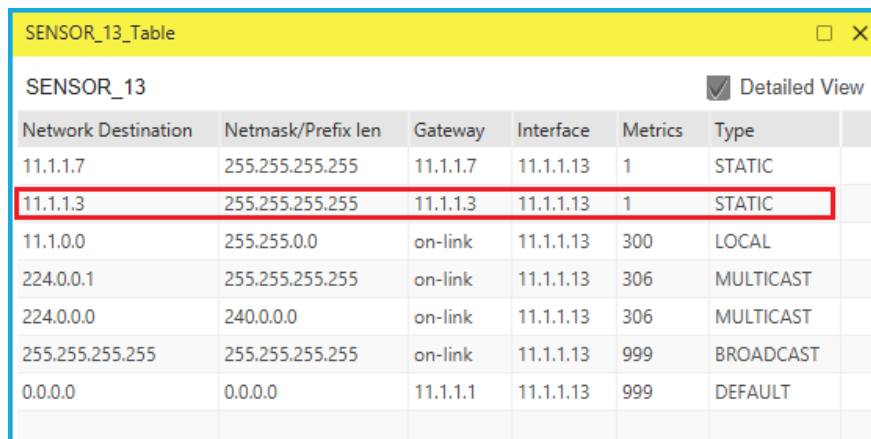
### Analyzing the device route tables in NetSim Results Dashboard

- 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 11.1.1.3 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:



Network Destination	Netmask/Prefix len	Gateway	Interface	Metrics	Type
11.1.1.3	255.255.255.255	11.1.1.13	11.1.1.2	1	STATIC
11.1.0.0	255.255.0.0	on-link	11.1.1.2	300	LOCAL
224.0.0.1	255.255.255.255	on-link	11.1.1.2	306	MULTICAST
224.0.0.0	240.0.0.0	on-link	11.1.1.2	306	MULTICAST
255.255.255.255	255.255.255.255	on-link	11.1.1.2	999	BROADCAST
0.0.0.0	0.0.0.0	11.1.1.1	11.1.1.2	999	DEFAULT

The static route entry for SENSOR\_2 specifies the next hop as SENSOR\_13 which has the IP 11.1.1.13.



Network Destination	Netmask/Prefix len	Gateway	Interface	Metrics	Type
11.1.1.7	255.255.255.255	11.1.1.7	11.1.1.13	1	STATIC
11.1.1.3	255.255.255.255	11.1.1.3	11.1.1.13	1	STATIC
11.1.0.0	255.255.0.0	on-link	11.1.1.13	300	LOCAL
224.0.0.1	255.255.255.255	on-link	11.1.1.13	306	MULTICAST
224.0.0.0	240.0.0.0	on-link	11.1.1.13	306	MULTICAST
255.255.255.255	255.255.255.255	on-link	11.1.1.13	999	BROADCAST
0.0.0.0	0.0.0.0	11.1.1.1	11.1.1.13	999	DEFAULT

The static route entry for SENSOR\_13 specifies the next hop as the destination node SENSOR\_3 which has the IP 11.1.1.3.

### 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:

A	B	C	D	E	F	G	H
PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID
4	0	Sensing	App1_SENSOR_APP	SENSOR-2	SENSOR-3	SENSOR-2	SENSOR-13
4	0	Sensing	App1_SENSOR_APP	SENSOR-2	SENSOR-3	SENSOR-13	SENSOR-3

## Result

### Case 1: Without SDN

Application_Metrics_Table							
Application_Metrics							
Application Id	Application Name	Packet generated	Packet received	Throughput (Mbps)	Delay(microsec)	Jitter(microsec)	
1	App1_SENSOR_APP	500	508	0.000406	28775.301575	18859.522288	
2	App2_SENSOR_APP	500	505	0.000404	28520.851089	22156.538889	

### Case 2: With SDN

Application_Metrics_Table							
Application_Metrics							
Application Id	Application Name	Packet generated	Packet received	Throughput (Mbps)	Delay(microsec)	Jitter(microsec)	
1	App1_SENSOR_APP	500	510	0.000408	25040.325490	17987.185069	
2	App2_SENSOR_APP	500	520	0.000416	22638.447308	18235.739114	

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