Congestion Control AODV (CC-AODV)

Software Recommended: NetSim Standard v12.1 (64 bit), Visual Studio 2019

Reference: Y. Mai, F. M. Rodriguez and N. Wang, "CC-ADOV: An effective multiple paths congestion control AODV," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, 2018, pp. 1000-1004.

Introduction

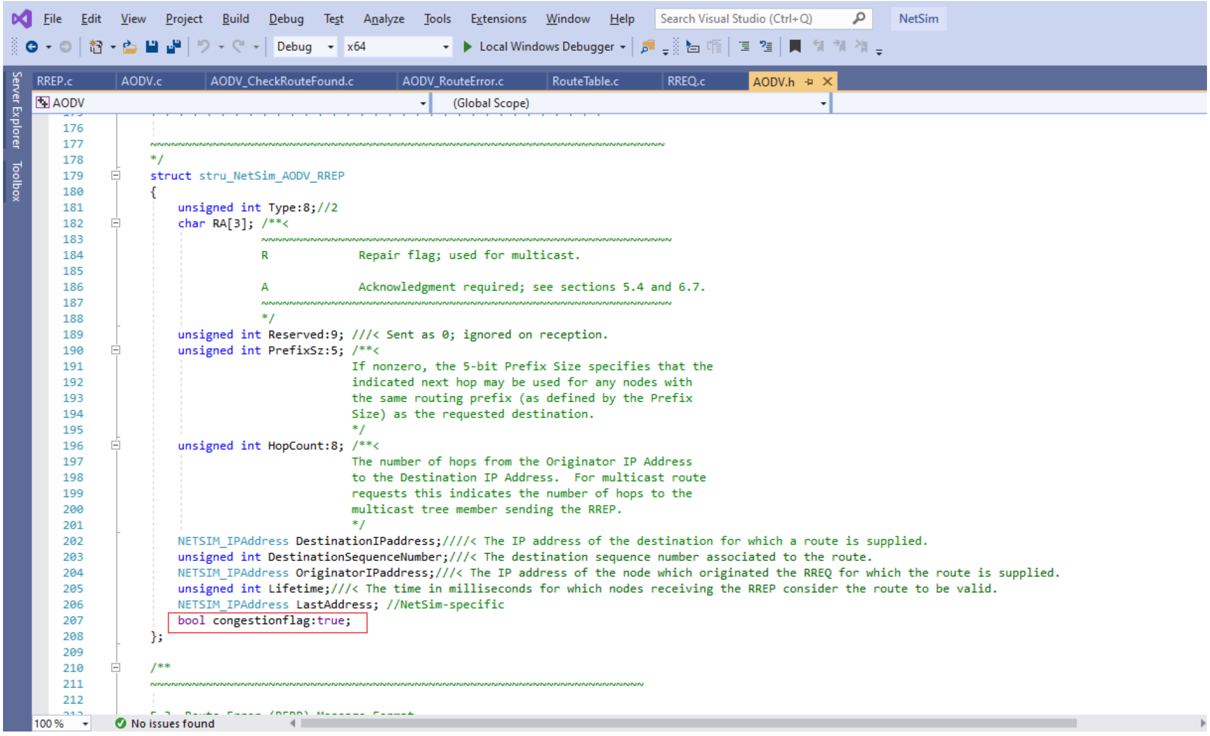
Ad hoc On-Demand Distance Vector (AODV) routing is one of the famous routing algorithms. Tremendous amounts of research on this protocol have been done to improve the performance. In this paper, a new control scheme, named congestion control AODV (CC-AODV), is proposed to manage the described routing condition. With this table entry, the package delivery rates are significantly increased while the package drop rate is decreased, however its implementation causes package overhead.

CC-ADOV aims to lower the performance degradation caused by the packets congestion while the data is delivered using AODV. Furthermore, CC-AODV determines a path for the data by using the congestion counter label. This is achieved by checking how stressed the current node is in a table, and once the RREP package is generated and transmitted through the nodes, the congestion counter adds one to the counter. The process of CC-AODV explains how to establish the route. First, the source node performs a flooding broadcast RREQ package in the entire network. When RREQ package arrives to the intermediate node, the router checks the congestion counter whether it is less than a certain predetermined value. If the comparison yields less than the counter, the routing table updates and forwarding to next router; otherwise, the router drops the RREQ package. Once the RREQ arrives to the corresponding destination, the RREP is generated by the router. In CC-AODV, the congestion flag is added to the RREP header. There are two cases of which a RREP is generated corresponding to a RREQ. One is from the source node to establish the route and the other is from the neighbour nodes to maintain the route. When the destination node receives the RREQ from the source node, it generates the RREP with the congestion flag set to true. While the RREP unicast back to the corresponding source node, passing by the intermediate node, the router checks the congestion flag. If it is true, the counter increases; otherwise, the counter keeps the same. Then, the router updates the routing information.

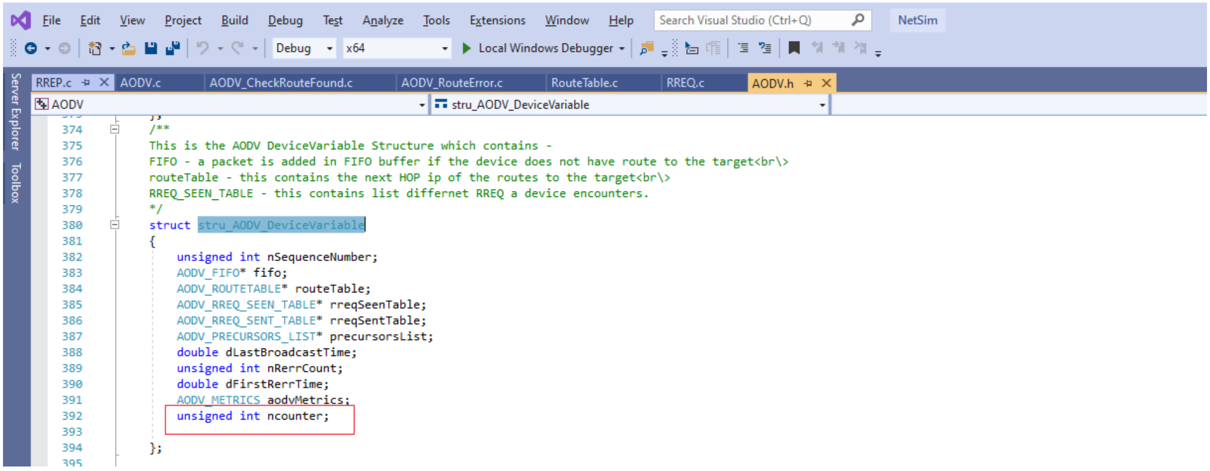
Procedure to implement CC-AODV in NetSim:

In order to implement CC-AODV following code modification done in AODV Protocol

1. The RREP structure stru_NetSim_AODV_RREP is defined in AODV.h has been modified to include a Congestion flag for implementing CC-AODV



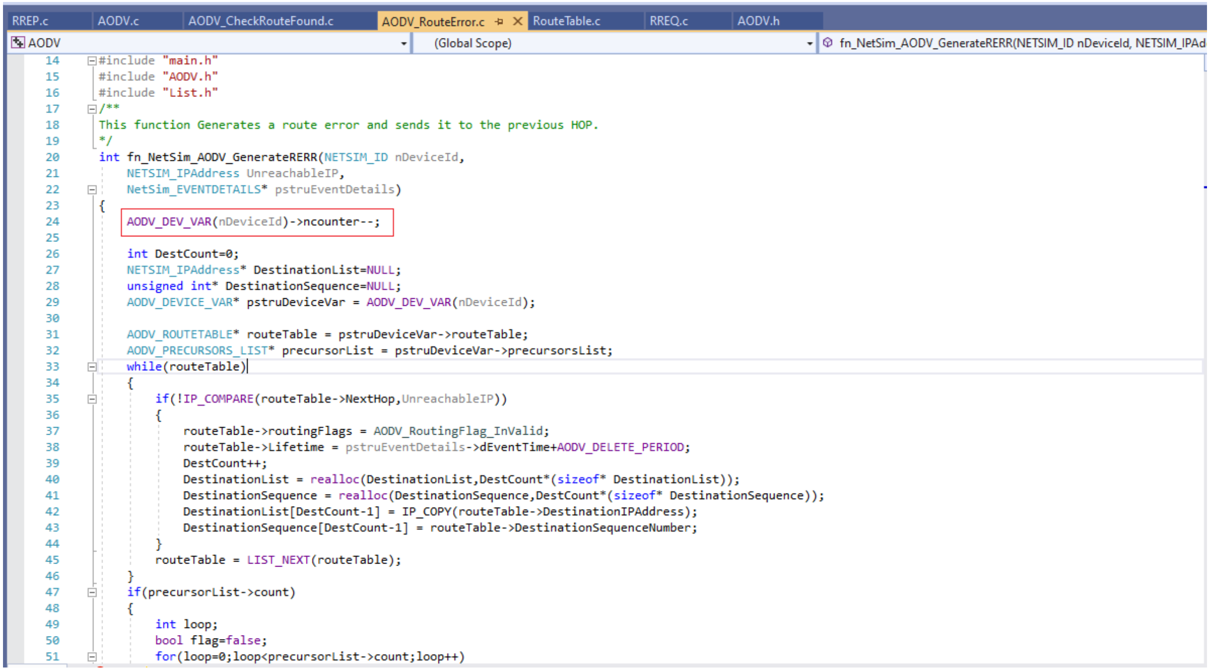
2. The DeviceVariable Structure stru_AODV_DeviceVariable is defined in AODV.h file has been modified to include a congestion counter for implementing CC-AODV



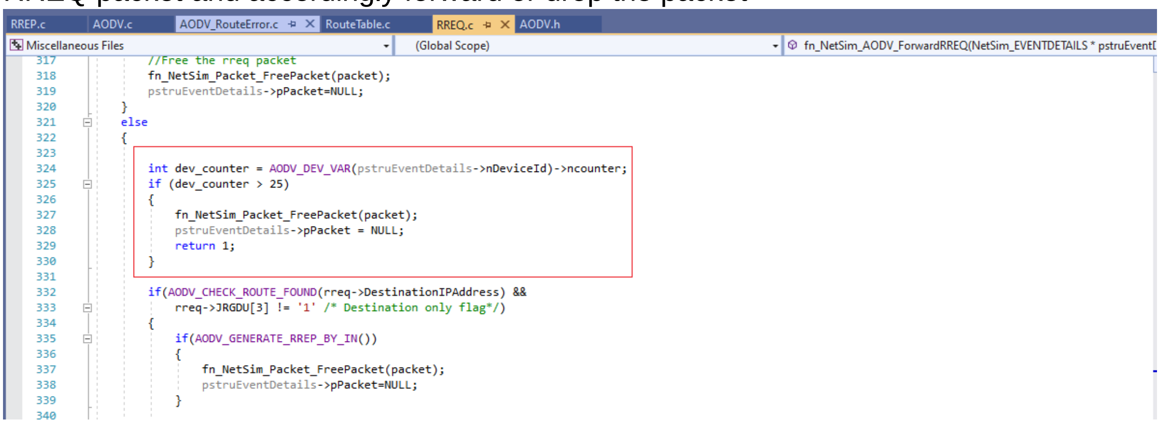
3. The source codes of functions in **RREP.c**, **RouteTable.c** and **AODV_RouteError.c** has been modified suitably to Increment, Decrement the congestion counter accordingly

```
Deletes the RREQ entry from sent table and forwards the rrep if the device is not
the source node.
*/
int fn_NetSim_AODV_ProcessRREP(NetSim_EVENTDETAILS* pstruEventDetails)
{
    AODV_ROUTETABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable;
    AODV_RREP* rrep = (AODV_RREP*)pstruEventDetails->pPacket->pstruNetworkData->Packet_RoutingProtocol;
    //Update the routing table
    if (rrep->DestinationIPaddress == aodv_get_curr_ip())
        return 0;

    if(rrep->congestionflag == true)
        AODV_DEV_VAR(pstruEventDetails->nDeviceId)->ncounter++;

    AODV_INSERT_ROUTE_TABLE(rrep->DestinationIPaddress,
        rrep->DestinationSequenceNumber,
        rrep->HopCount,
        rrep->LastAddress,
        pstruEventDetails->dEventTime+AODV_ACTIVE_ROUTE_TIMEOUT);
    //Transmit the buffer
    AODV_TRANSMIT_FIFO(AODV_DEV_VAR(pstruEventDetails->nDeviceId));
    //Update the precursor list
    AODV_INSERT_PRECURSOR(rrep->LastAddress);
    AODV_UPDATE_ROUTE_TABLE(rrep->LastAddress,rrep->Lifetime);
    if(!IP_COMPARE(aodv_get_curr_ip(),rrep->OriginatorIPaddress))
    {
        //Delete entry from sent table
        AODV_RREQ_SENT_TABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->rreqSentTable;
        while(table)
        {
            if(!IP_COMPARE(table->DestAddress,rrep->DestinationIPaddress))
            {
                IP_FREE(table->DestAddress);
                LIST_FREE((void**)&AODV_DEV_VAR(pstruEventDetails->nDeviceId)->rreqSentTable,table);
                break;
            }
            table = (AODV_RREQ_SENT_TABLE*)LIST_NEXT(table);
        }
```

```
/**
This function adds the timeout event of a Route Table which is equal to the table_LifeTime
*/
int fn_NetSim_AODV_ActiveRouteTimeout(NetSim_EVENTDETAILS* pstruEventDetails)
{
    int flag = 0;
    NETSIM_IPAddress dest = (NETSIM_IPAddress)pstruEventDetails->szOtherDetails;
    AODV_ROUTETABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable;
    while(table)
    {
        if(!IP_COMPARE(table->DestinationIPAddress,dest))
        {
            if(table->Lifetime <= pstruEventDetails->dEventTime)
            {
                AODV_ROUTETABLE* temp = LIST_NEXT(table);
                IP_FREE(table->DestinationIPAddress);
                IP_FREE(table->NextHop);
                LIST_FREE(&AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable,table);
                AODV_DEV_VAR(pstruEventDetails->nDeviceId)->ncounter--;
                table = temp;
                continue;
            }
            else
            {
                //Add new time out event
                pstruEventDetails->dEventTime = table->Lifetime;
                fnpAddEvent(pstruEventDetails);
                flag = 1;
            }
        }
        table=(AODV_ROUTETABLE*)LIST_NEXT(table);
    }
    if(!flag)
        IP_FREE(dest);
    return 1;
}
```

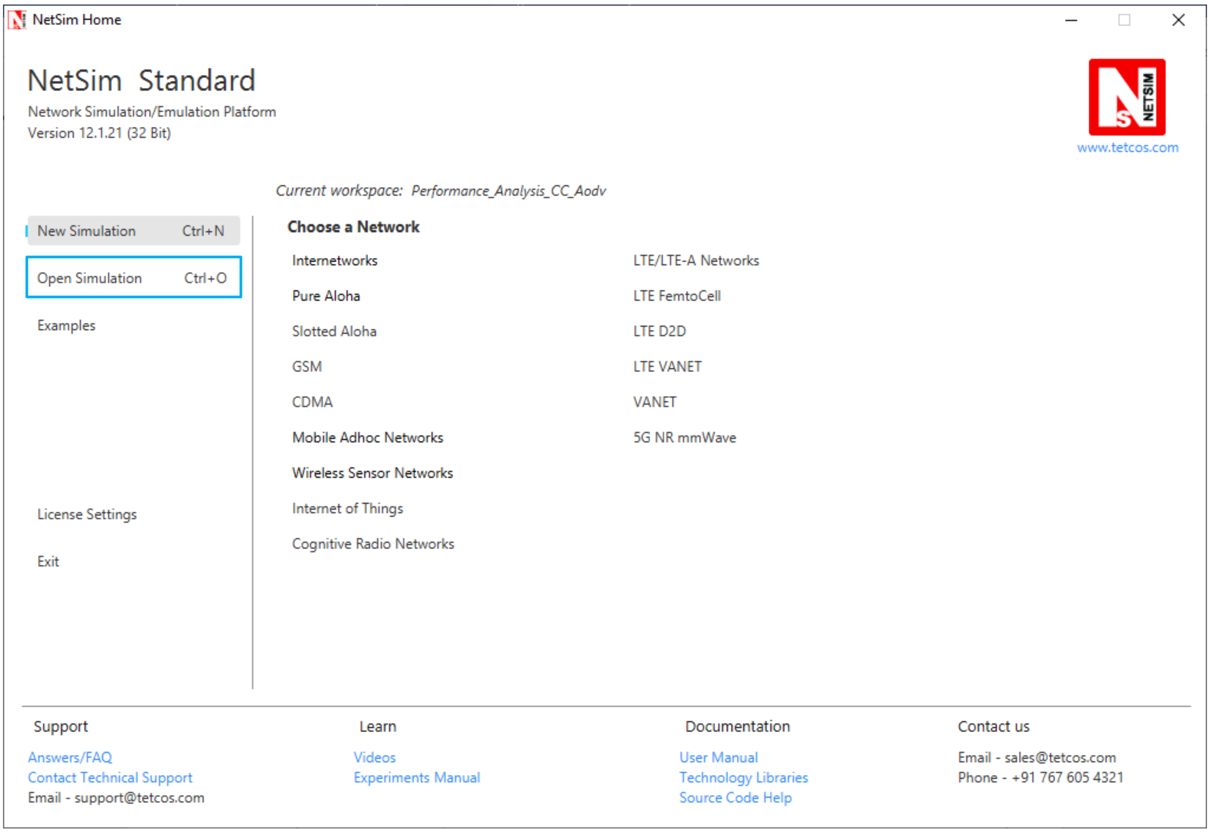


4. The source codes and functions related to Route request are defined in the file **RREQ.c.** The **fn_NetSim_AODV_ProcessRREQ()** function that is part of this file has been modified suitably to check the value of the congestion counter in the received RREQ packet and accordingly forward or drop the packet

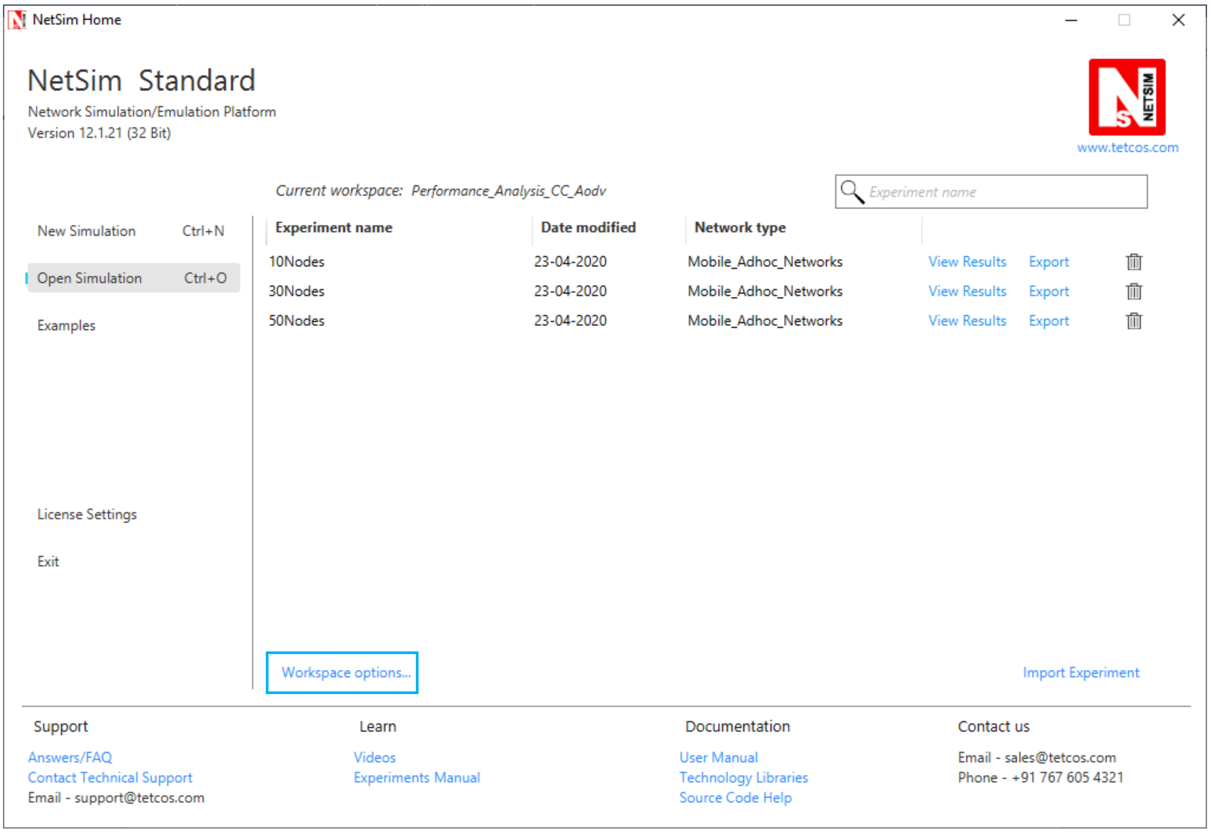


Steps:

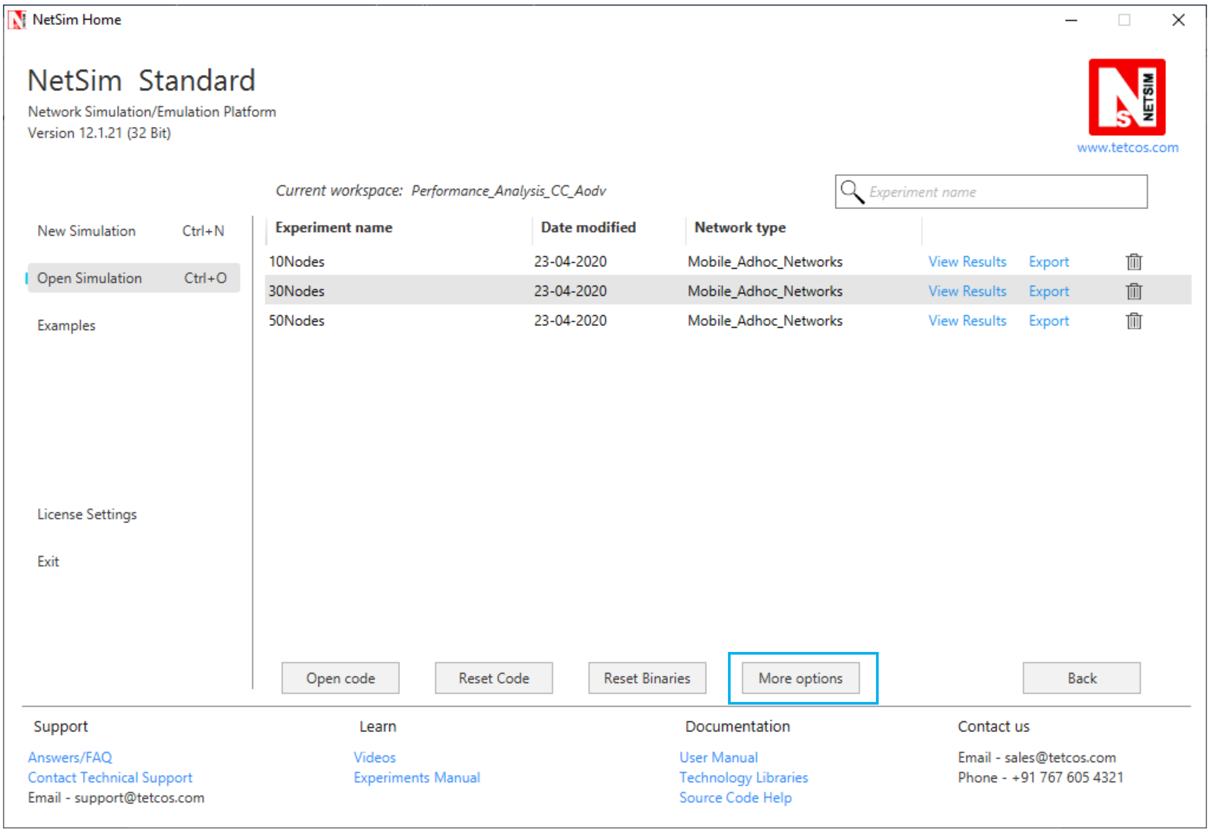
1. After you unzip the downloaded project folder, Open NetSim Home Page click on **Open Simulation**



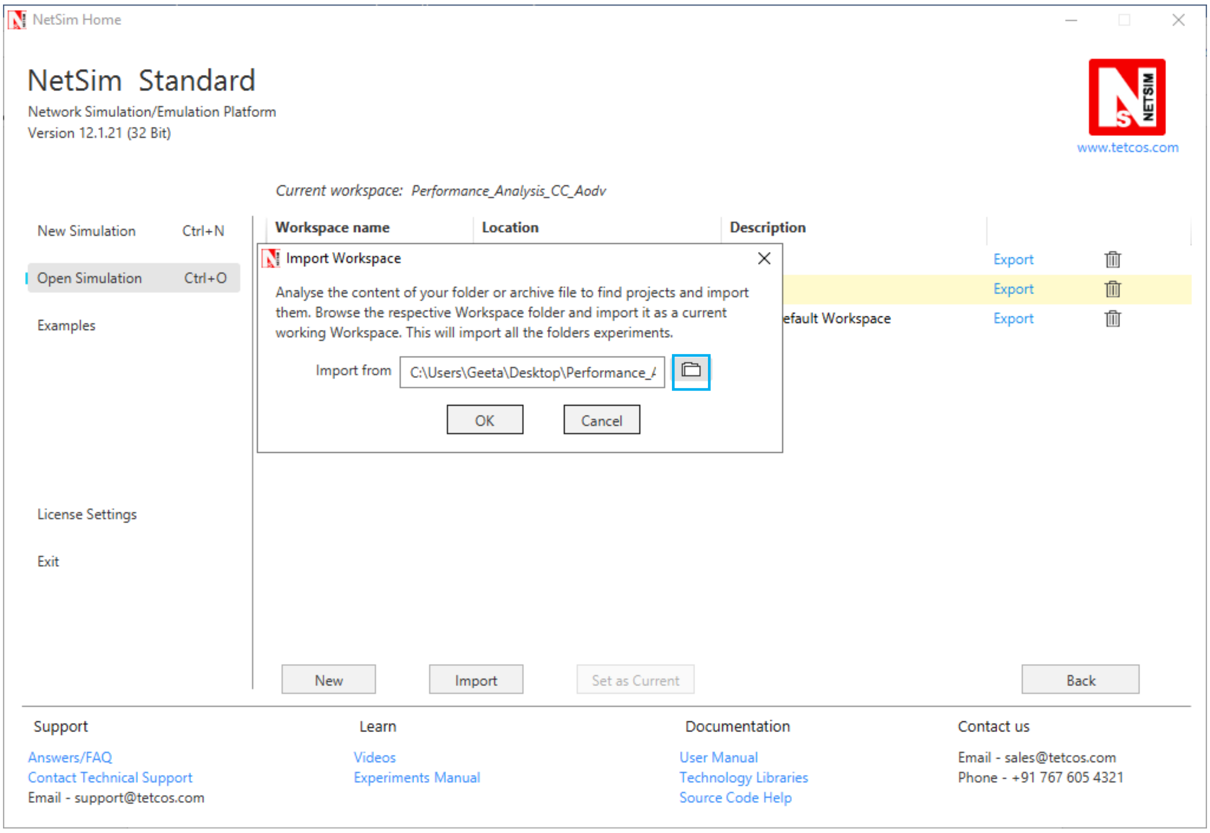
2. Click on **Workspace options**



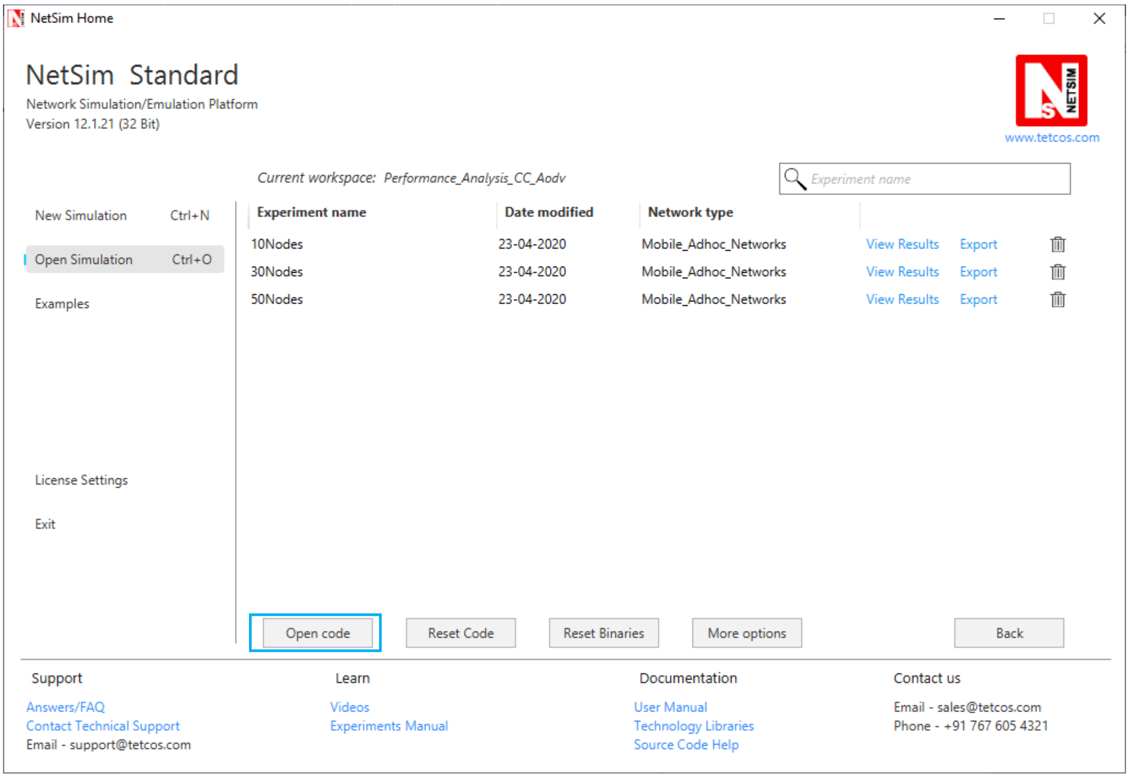
3. Click on **More Options,**



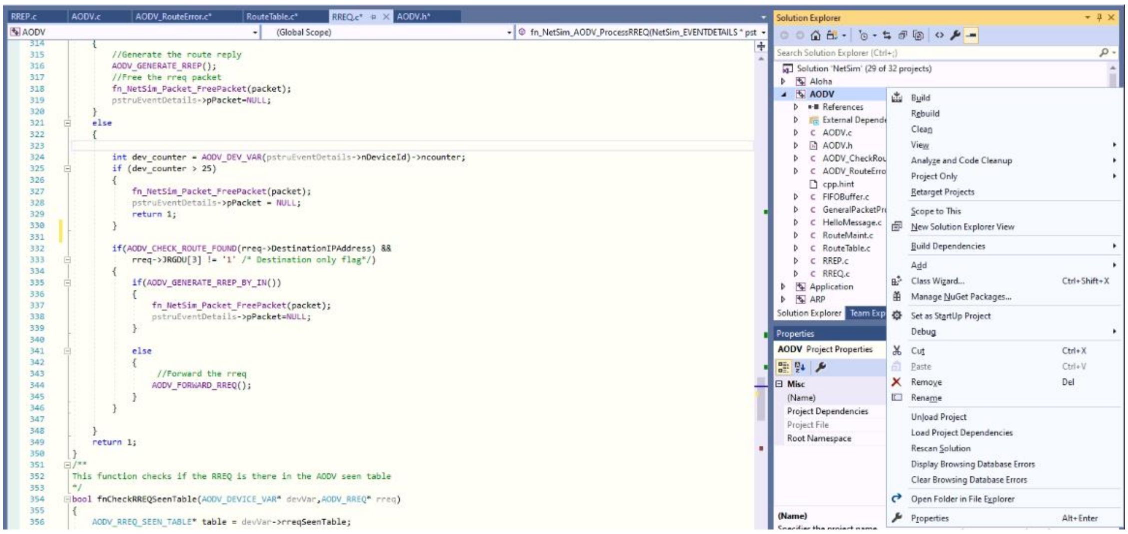
4. Click on **Import**, browse the extracted folder and go into the WorkSpace_Performance_Analysis_CC_AODV directory, click on select folder and then click on OK.



5. Go to home page, Click on **Open Simulation → Workspace options → Open code**



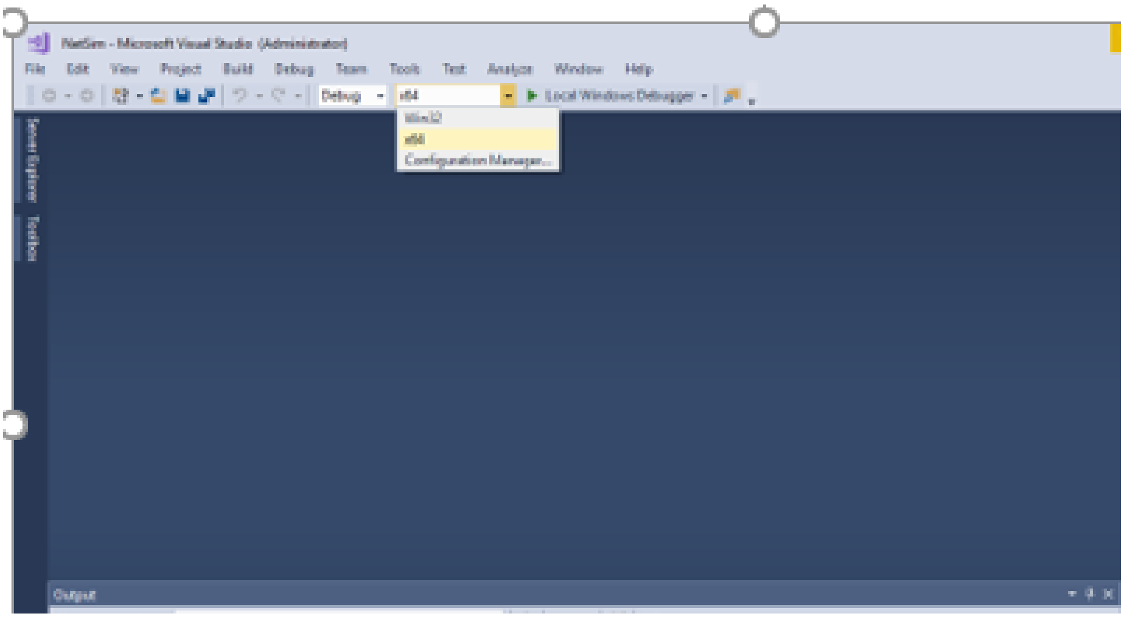
6. Right click on the AODV Project and select rebuild.



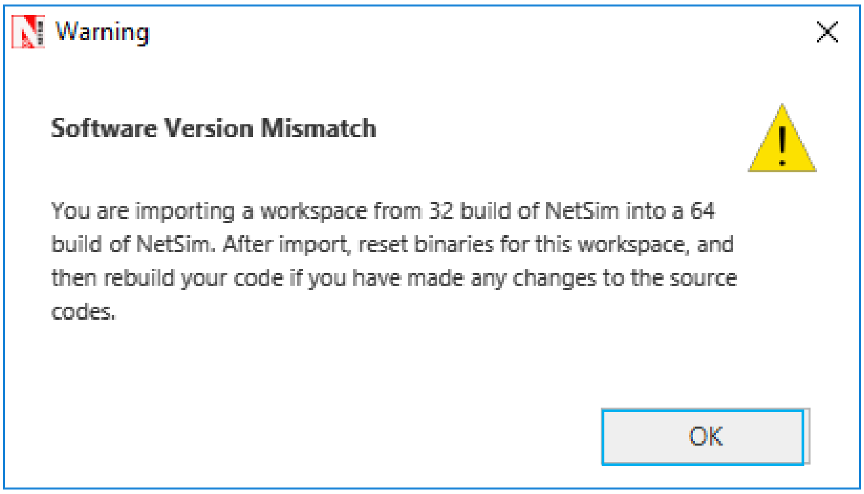
7. Upon rebuilding, **libAodv.dll** will automatically get updated in the respective bin folder of the current workspace.

Note:

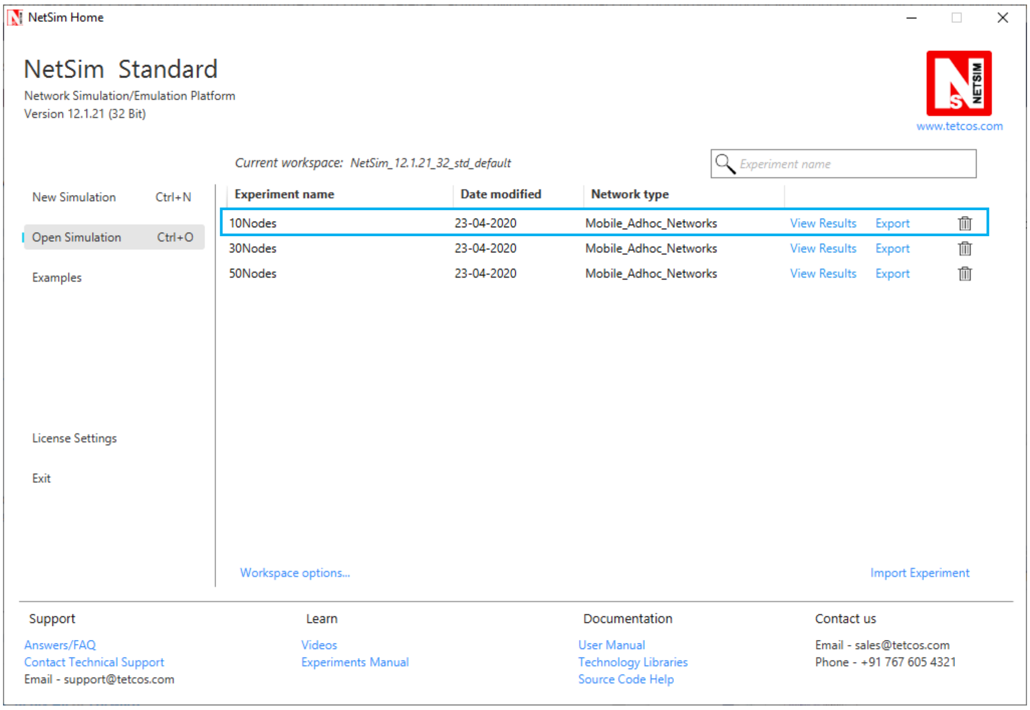
- Based on whether you are using NetSim 32 bit or 64 bit setup you can configure Visual studio to build 32 bit or 64 bit Dll files respectively as shown below:



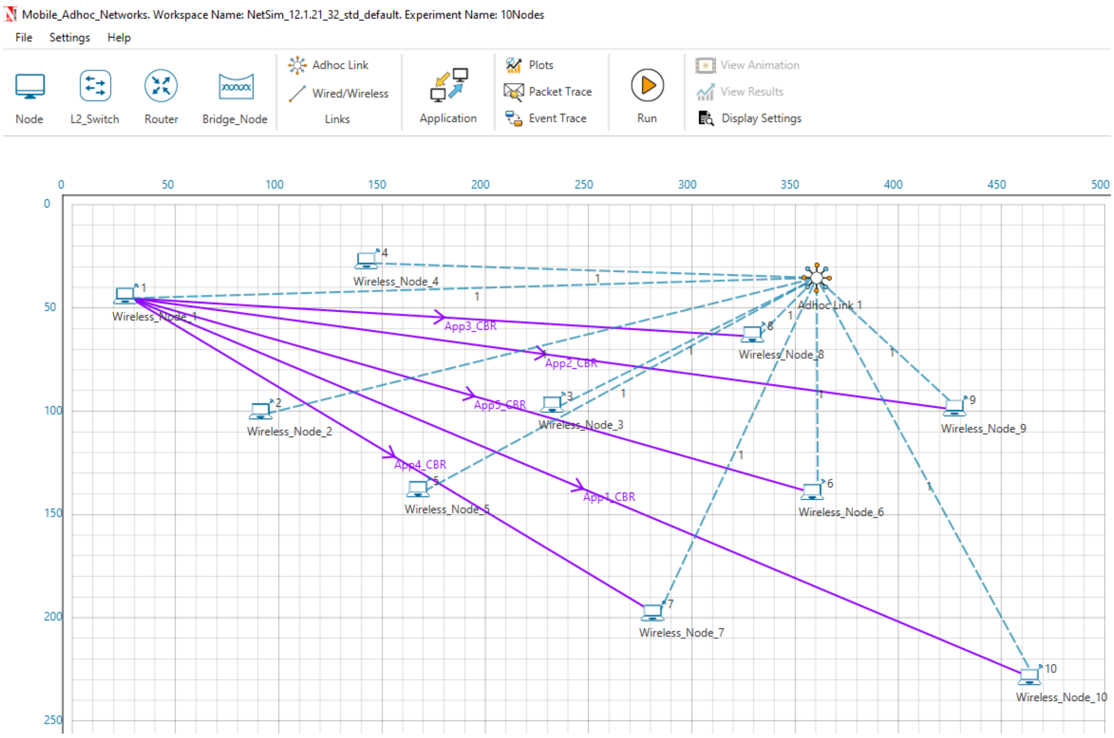
- While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



8. Go to NetSim home page, click on **Open Simulation**, Click on **10Nodes_Example**.



9. Now create a Network Scenario in NetSim Mobile Adhoc Network as per the screenshot below.



A sample Configuration.netsim file is also provided in the Config_File folder along with this project which can be opened in NetSim directly.

10. Run the simulation for 30 sec

Simulations have been carried out using a different number of nodes in a network to symbolize different practical applications of wireless network. For example, 10 nodes symbolize a small network that can be used in an agricultural setup. 30 nodes symbolize a medium size network that can be used in an industrial setup and a large 50 nodes network that can be used in an army base.

Result:

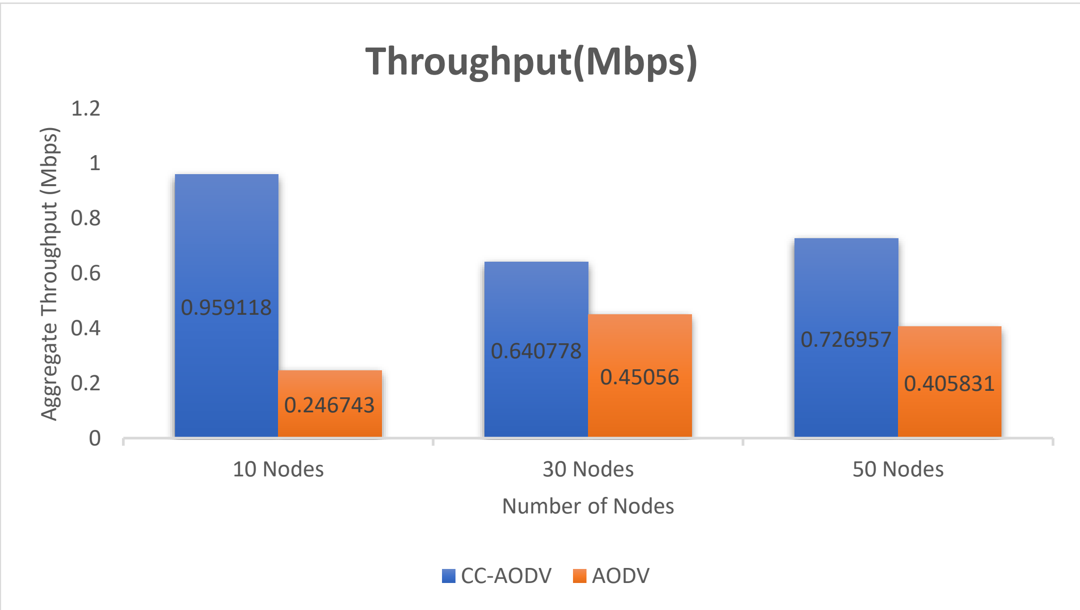
Performance of CC-AODV has been compared with other reactive protocol AODV based on different performance metrics such as Throughput, End to End delay etc.

Number of Nodes	AODV Aggregate Throughput (Mbps)	CC_AODV Aggregate Throughput (Mbps)
10Nodes	0.246743	0.959118
30Nodes	0.45056	0.640788
50Nodes	0.405831	0.726957

Table 1 : Aggregate Throughput comparison between AODV and CC_AODV

As per the Table 1 the proposed CC-AODV has higher throughput than the AODV. In CC-AODV, the internal nodes can be utilized much efficiently than AODV because the counter helps to reroute the path if the internal node is busy. This can increase the network channel utilization.

This can be further understood with the help of following graph:



Number of Nodes	AODV Average Delay (microseconds)	CC_AODV Average Delay (microseconds)
10Nodes	901640.64	1455064.42
30Nodes	3327557.09	3819669.58
50Nodes	2076527.25	3474913.66

Table 2: End to End delay comparison between AODV and CC_AODV

Table 2 demonstrate that CC-AODV has slightly higher End-to-End performance than the AODV, the result is achieved by rerouting the path of the data if the router is on a busy state.

This can be further understood with the help of following graph: