Simulation of Congestion Control AODV (CC-AODV) in NetSim

Software Recommended: NetSim Standard v12.2 (64 bit), Visual Studio 2019 or Higher version.

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

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

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

Other tools such as GitHub Desktop, SVN Client, Source tree, 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/CC-AODV-v12.2.git

Introduction

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is a prominent algorithm in the field of wireless communication. Extensive research has been conducted to optimize its performance. In this referenced paper, the authors introduce a novel control scheme, termed Congestion Control AODV (CC-AODV), designed to enhance routing efficiency. Implementing this scheme substantially improves package delivery rates and reduces package drop rates, albeit at the cost of package overhead.

CC-AODV's primary objective is to mitigate performance degradation due to packet congestion during data transmission via AODV. The protocol establishes a data path using a metric we refer to as the "congestion counter label". This metric quantifies the congestion level of a node, which is then stored in a table. Upon the generation and transmission of a Route Reply (RREP) packet, the congestion counter is incremented.

The CC-AODV procedure to set up a route is as follows:

1. The source node initiates a Route Request (RREQ) broadcast throughout the network.
2. As the RREQ is received by an intermediate node, the router consults the congestion counter. If the value is below a predefined threshold, the routing table is updated and the RREQ is forwarded to the subsequent router. Otherwise, the RREQ is discarded.
3. On reaching the intended destination, an RREP is produced. Within the CC-AODV framework, an additional 'congestion flag' is integrated into the RREP header.

There are two scenarios in which an RREP is generated in response to an RREQ:

- By the source node to establish a route.

- By neighbouring nodes to maintain an existing route.

When the destination node intercepts the RREQ from the source node, it issues an RREP with the congestion flag activated. As the RREP is unicasted back towards the source node via intermediate nodes, each router checks the congestion flag. If activated, the counter increments. Otherwise, it remains unchanged. Subsequently, the router updates its routing information.

Procedure to implement CC-AODV in NetSim:

To implement CC-AODV following code modification 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

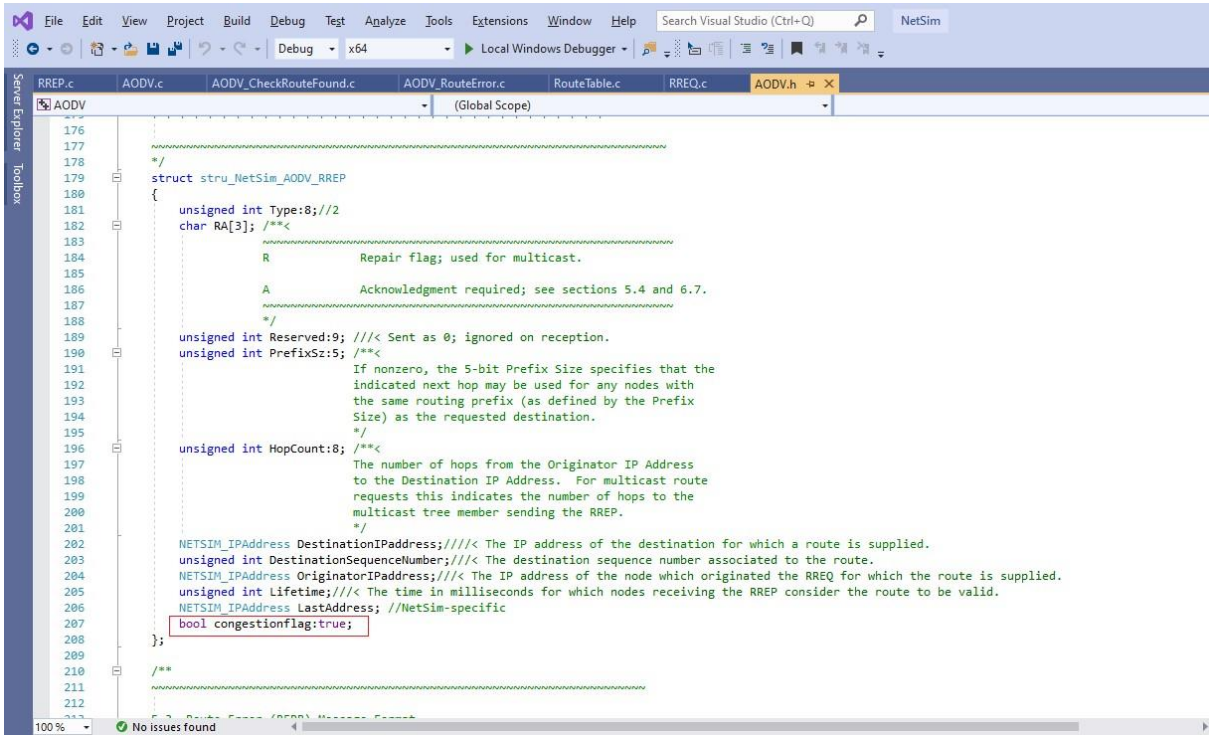


Figure 1: Changes inside stru_NetSim_AODV_RREP function in AODV.h

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

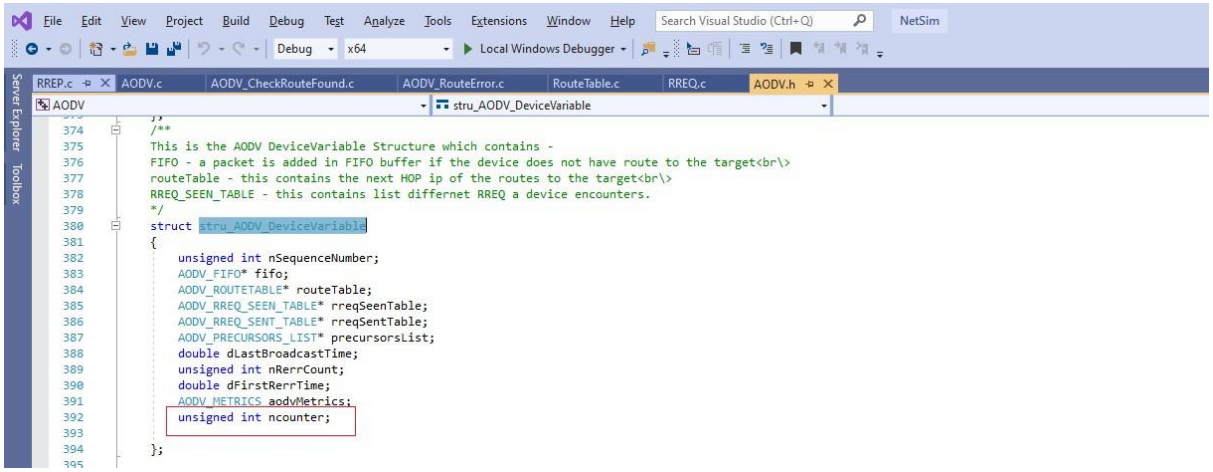
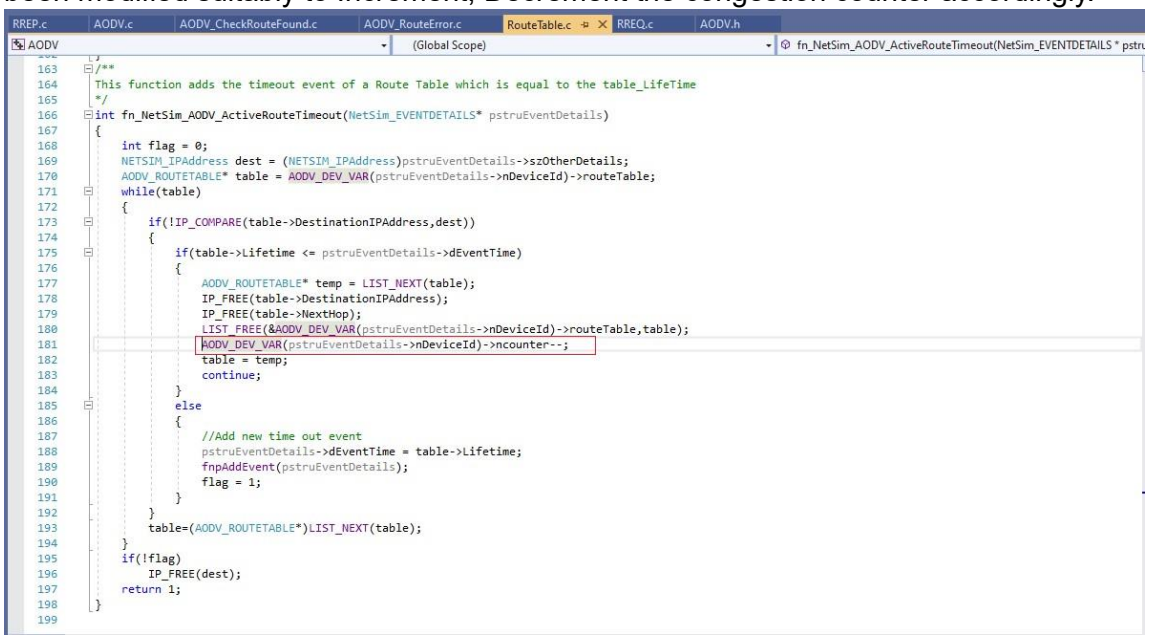


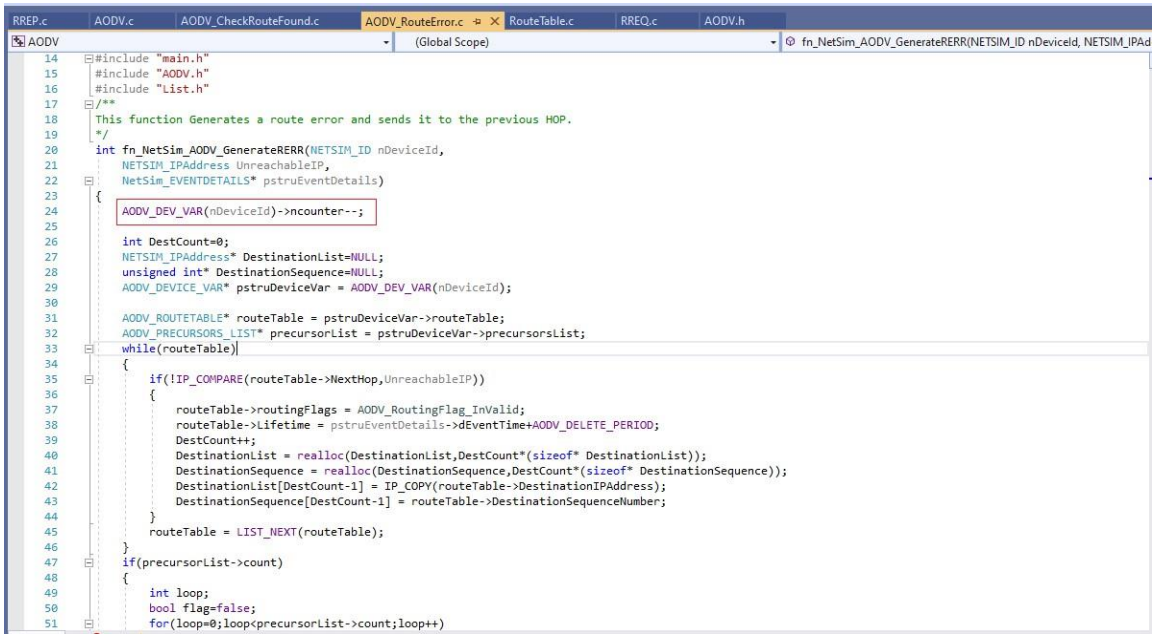
Figure 2: Adding congestion counter in stru_AODV_DeviceVariable

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.



```
/**
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;
}
```

Figure 3: Changes in RREP.c



```
#include "main.h"
#include "AODV.h"
#include "List.h"
/**
This function Generates a route error and sends it to the previous HOP.
*/
int fn_NetSim_AODV_GenerateRERR(NETSIM_ID nDeviceId,
    NETSIM_IPAddress UnreachableIP,
    NetSim_EVENTDETAILS* pstruEventDetails)
{
    AODV_DEV_VAR(nDeviceId)->ncounter--;

    int DestCount=0;
    NETSIM_IPAddress* DestinationList=NULL;
    unsigned int* DestinationSequence=NULL;
    AODV_DEVICE_VAR* pstruDeviceVar = AODV_DEV_VAR(nDeviceId);

    AODV_ROUTETABLE* routeTable = pstruDeviceVar->routeTable;
    AODV_PRECURSORS_LIST* precursorList = pstruDeviceVar->precursorsList;
    while(routeTable)
    {
        if(!IP_COMPARE(routeTable->NextHop,UnreachableIP))
        {
            routeTable->routingFlags = AODV_RoutingFlag_InValid;
            routeTable->Lifetime = pstruEventDetails->dEventTime+AODV_DELETE_PERIOD;
            DestCount++;
            DestinationList = realloc(DestinationList,DestCount*(sizeof* DestinationList));
            DestinationSequence = realloc(DestinationSequence,DestCount*(sizeof* DestinationSequence));
            DestinationList[DestCount-1] = IP_COPY(routeTable->DestinationIPAddress);
            DestinationSequence[DestCount-1] = routeTable->DestinationSequenceNumber;
        }
        routeTable = LIST_NEXT(routeTable);
    }
    if(precursorList->count)
    {
        int loop;
        bool flag=false;
        for(loop=0;loop<precursorList->count;loop++)
```

Figure 4: Changes in AODV_RouteError.c

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.

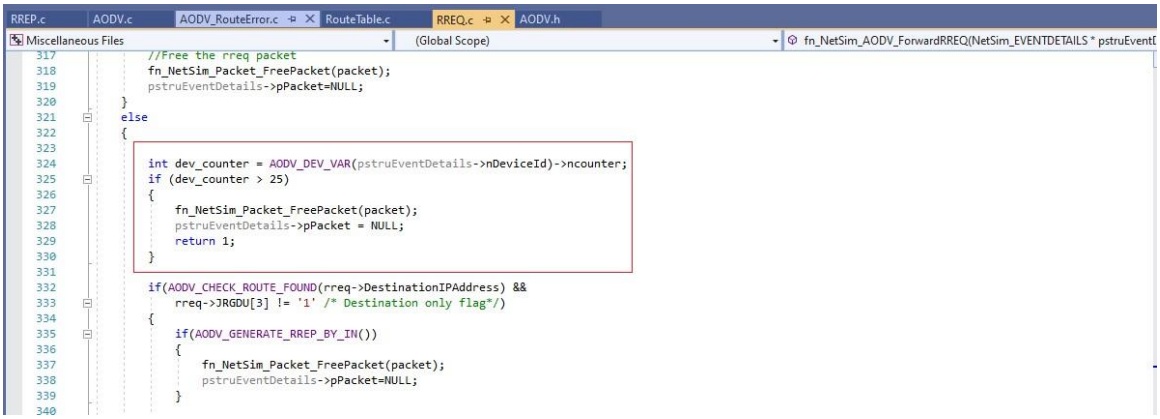


Figure 5: Modifications within the fn_NetSim_AODV_ProcessRREQ() in RREQ.c

The above code modifications are done to the provided workspace, user can follow the below procedure for importing the workspace and run the simulation.

Steps to simulate:

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

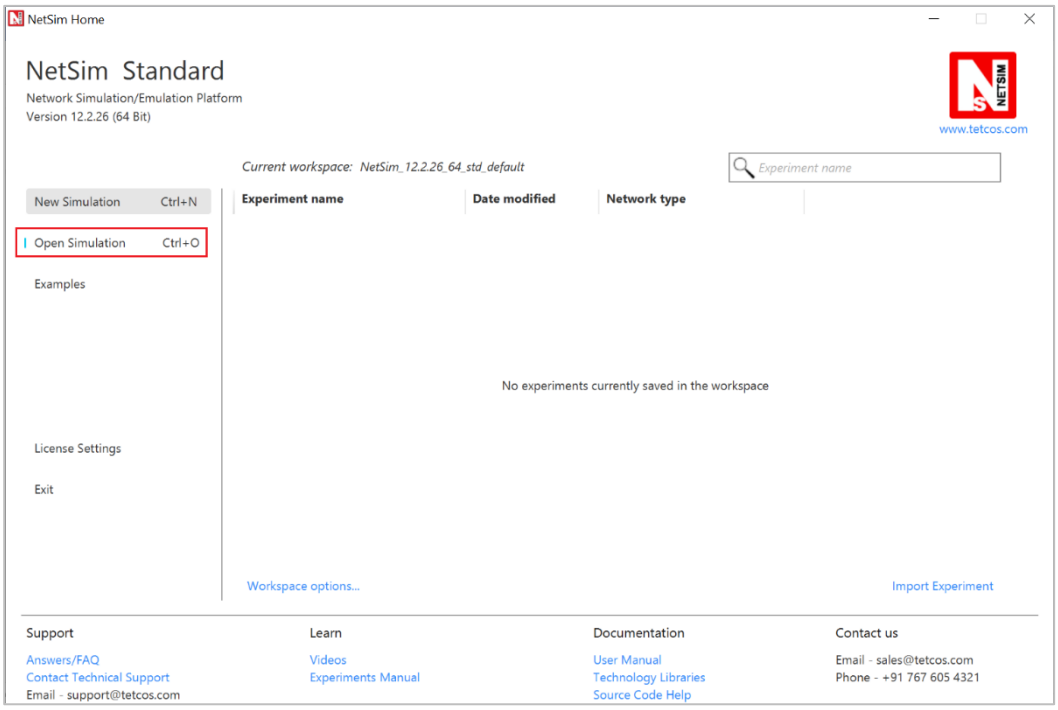


Figure 6: Selecting “Open simulation” from NetSim home page

2. Click on **Workspace options.**

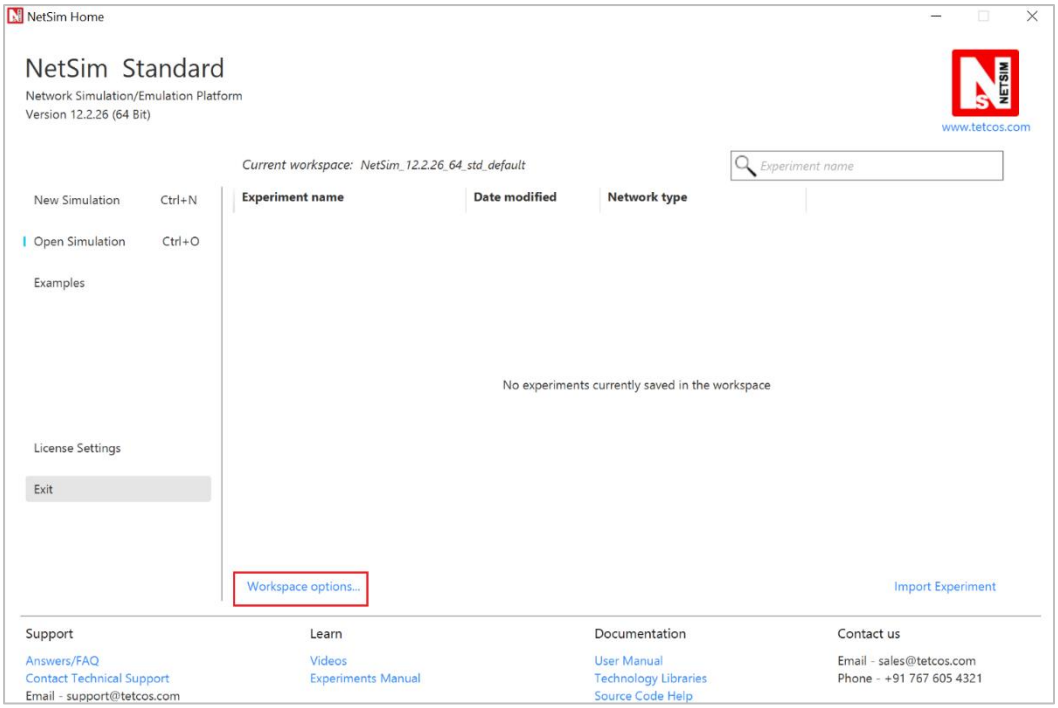


Figure 7: Selecting workspace option from NetSim home page.

3. Click on **More Options**

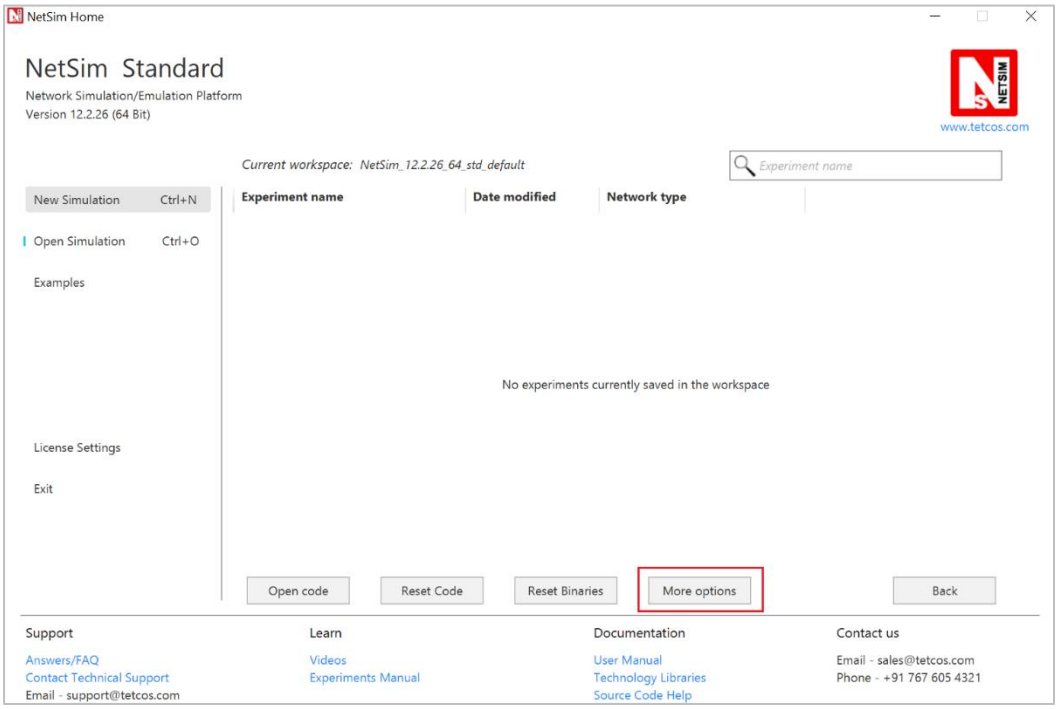


Figure 8: Selecting more options to import the workspace into NetSim

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

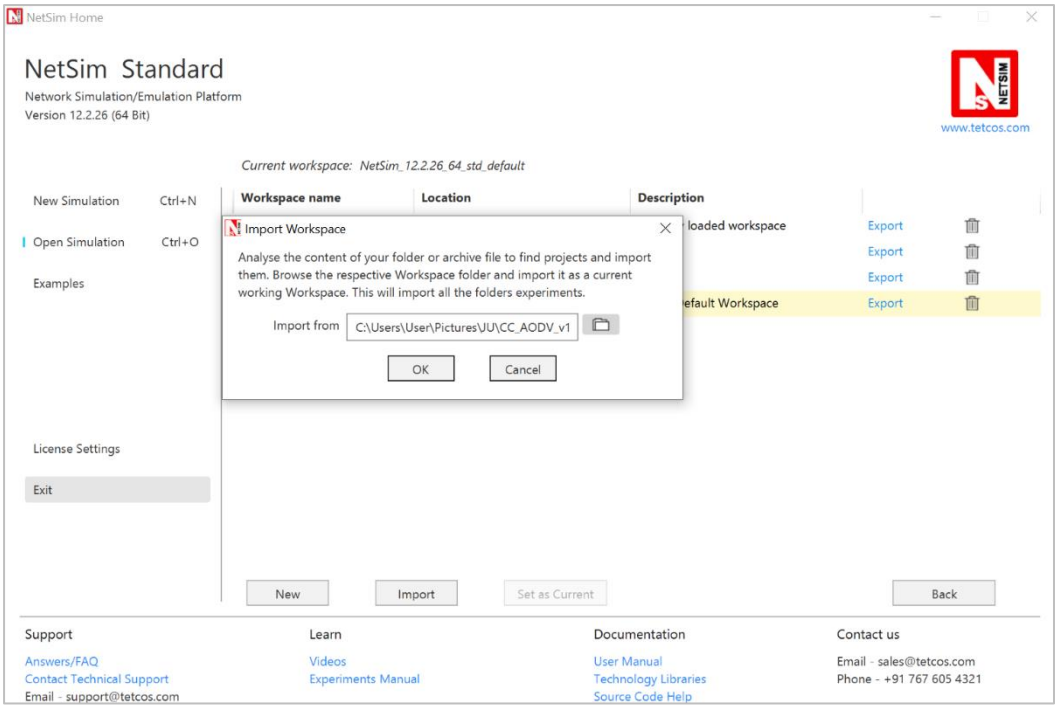


Figure 9: Importing the performance analysis of CC AODV workspace into NetSim.

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

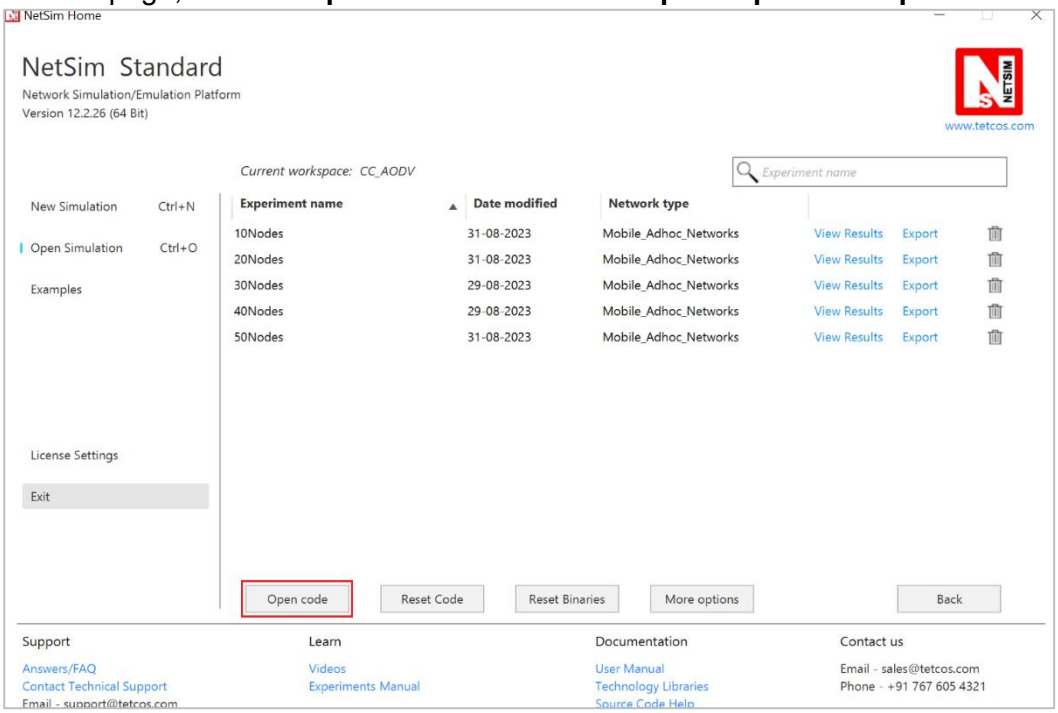


Figure 10: List of experiments in CC AODV workspace

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

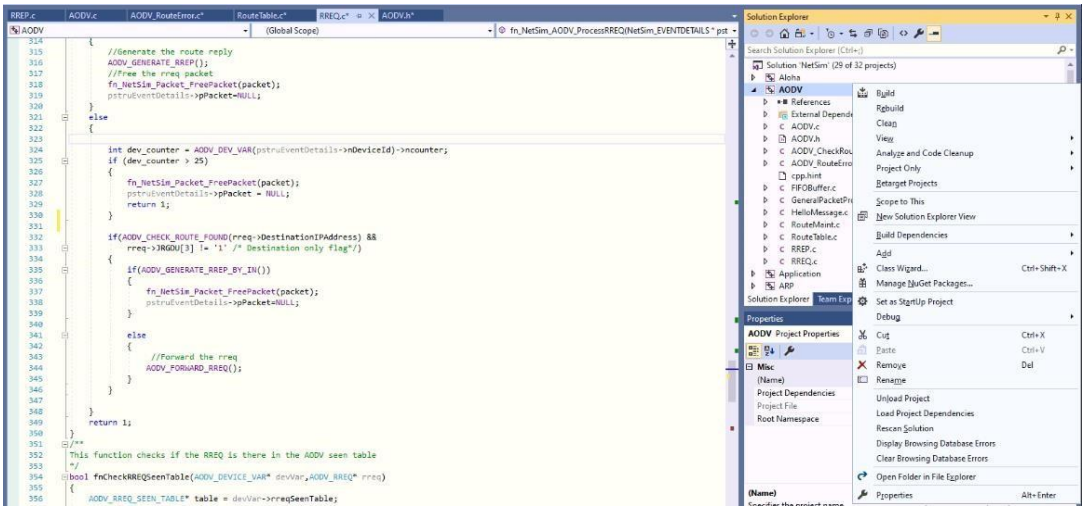


Figure 11: Screenshot of rebuilding the AODV project.

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

Note:

a) 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:

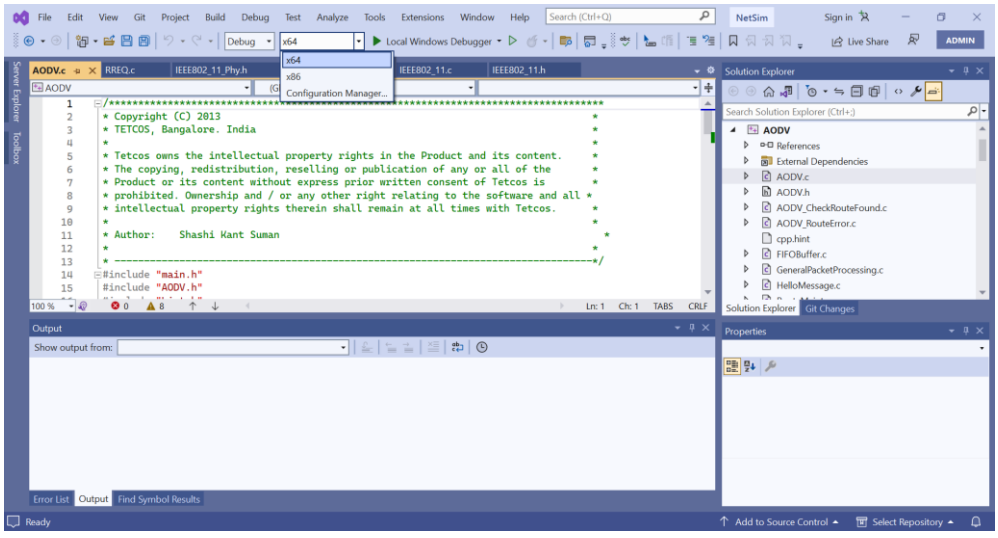


Figure 12: Screenshot showing changing of the solution platform for 32 and 64-bit version of NetSim.

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

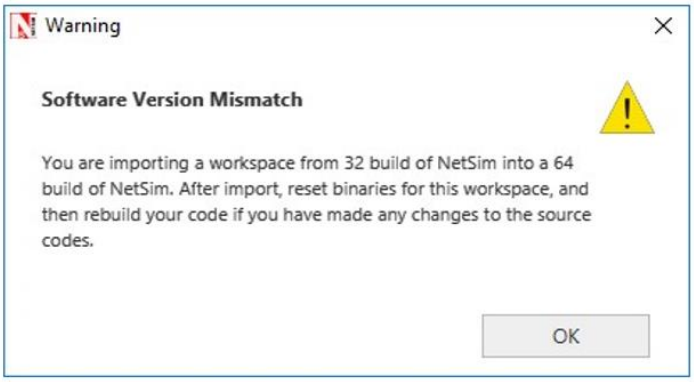


Figure 13: Software mismatch warning message

8. Go to NetSim home page, click on **Open Simulation**, Click on **20Nodes Example**.

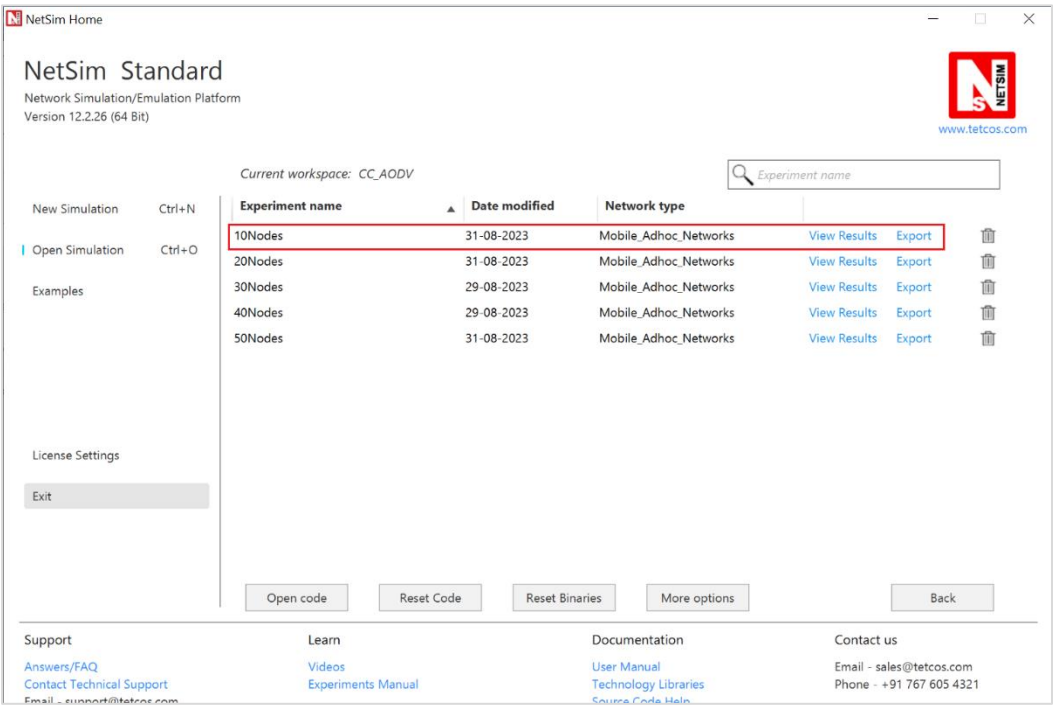


Figure 14: Opening 10nodes example from CC AODV workspace

9. 10 Node example is created in MANET as per the below screenshot.

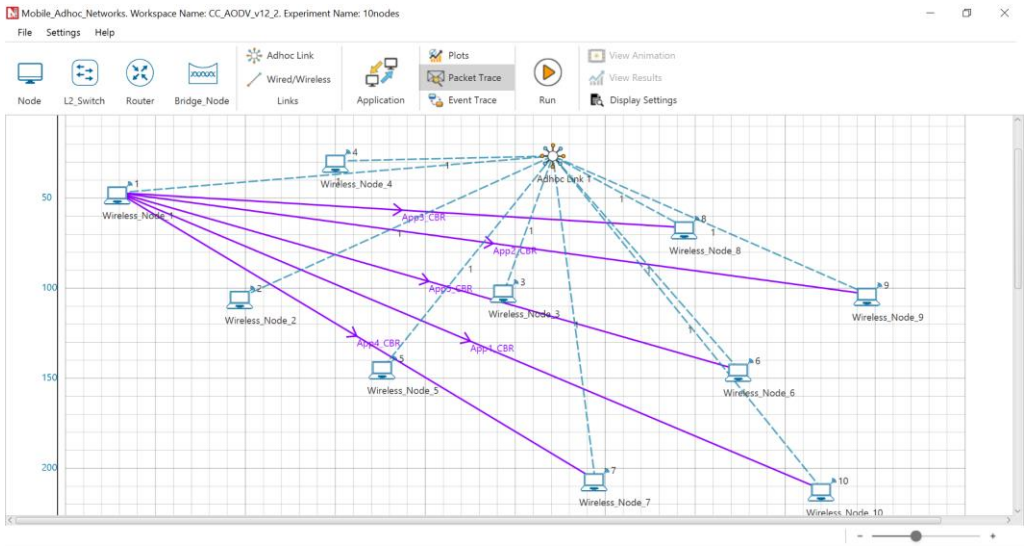


Figure 15: Network topology in this project

10. Run the simulation for 100sec.

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 to 20 nodes symbolize a small network that can be used in an agricultural setup. 30 to 40 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 and discussion:

Performance of CC-AODV has been compared against AODV with respect to Throughput and End to End delay.

Number of Nodes	AODV Aggregate throughput (Mbps)	CC AODV Aggregate throughput (Mbps)
10 Nodes	0.67	0.69
20 Nodes	0.69	0.70
30 Nodes	0.96	1.04
40 Nodes	1.09	1.20
50 Nodes	1.55	1.64

Table 1: Aggregate Throughput comparison between AODV and CC-AODV

Per Table 1 we see that CC-AODV has higher throughput than the AODV. This enhancement in CC-AODV can be attributed to its more efficient utilization of internal nodes. Specifically, the incorporated counter in CC-AODV facilitates path rerouting if an internal node is occupied, leading to improved network channel utilization.

The results of Table *1* are plotted below.

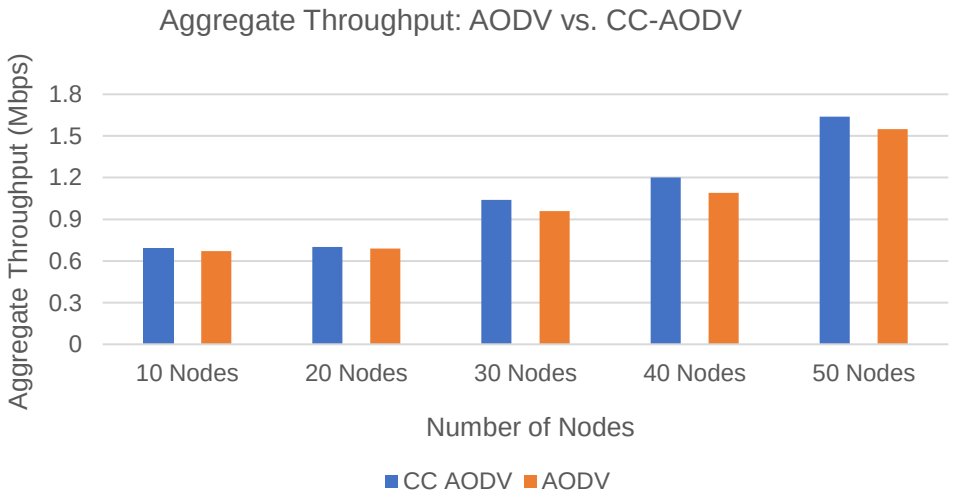


Figure 16: Comparison of aggregate throughputs of AODV and CC-AODV

Number of Nodes	AODV Average Delay (seconds)	CC AODV Average Delay (seconds)
10 Nodes	0.542	0.512
20 Nodes	0.522	0.462
30 Nodes	0.523	0.517
40 Nodes	0.508	0.430
50 Nodes	0.511	0.476

Table 2: End to End delay comparison between AODV and CC-AODV

In Table 2 we observe that CC-AODV has a lower delay that AODV due to re-routing the path of the data if the router is on a busy state. The results of Table *2* are plotted below.

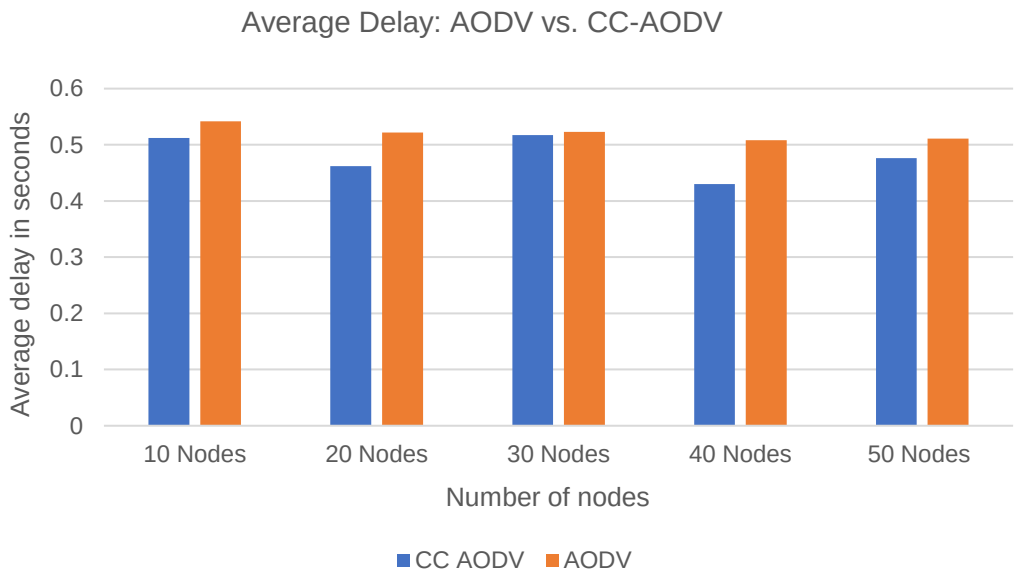


Figure17: Comparison between of Average delay of AODV and CC-AODV

From the table below, we observe that the number of collided packets is higher in the case of AODV than in CC AODV. This is because CC AODV selects paths with a lower collision rate, leading to improved overall packet delivery.

Number of Nodes	Collision packets in AODV	Collision packets in CC AODV
10 Nodes	1664	1220
20 Nodes	2202	1178
30 Nodes	1028	578
40 Nodes	1020	584
50 Nodes	108	58

Table 3: Comparison of collided packets between AODV and CC-AODV

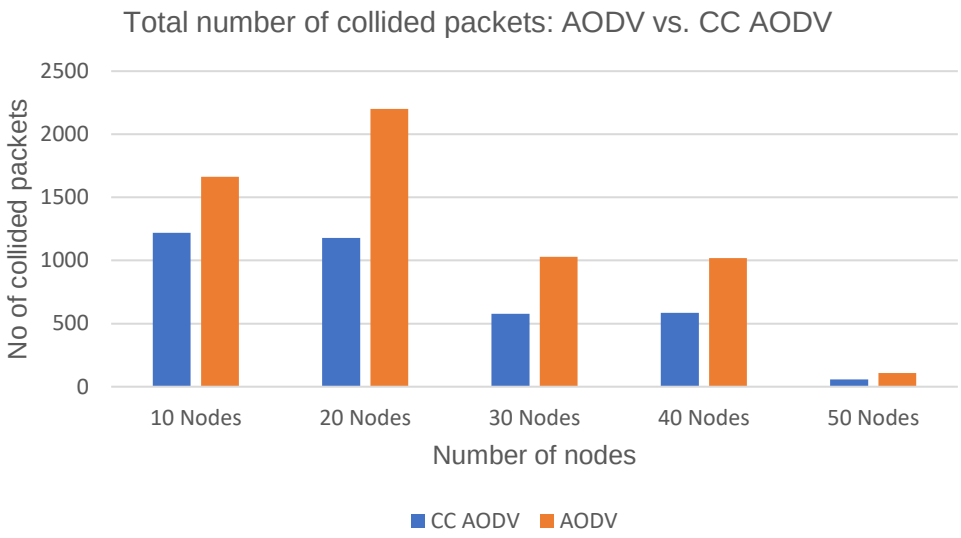


Figure 18: Comparison between AODV and CC-AODV collided packets