Congestion Control AODV (CC-AODV)

Software Recommended: NetSim Standard v13.0 (64 bit), Visual Studio 2019

Project Download Link:
https://github.com/NetSim-TETCOS/CC_AODV_Project_v13.0/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

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

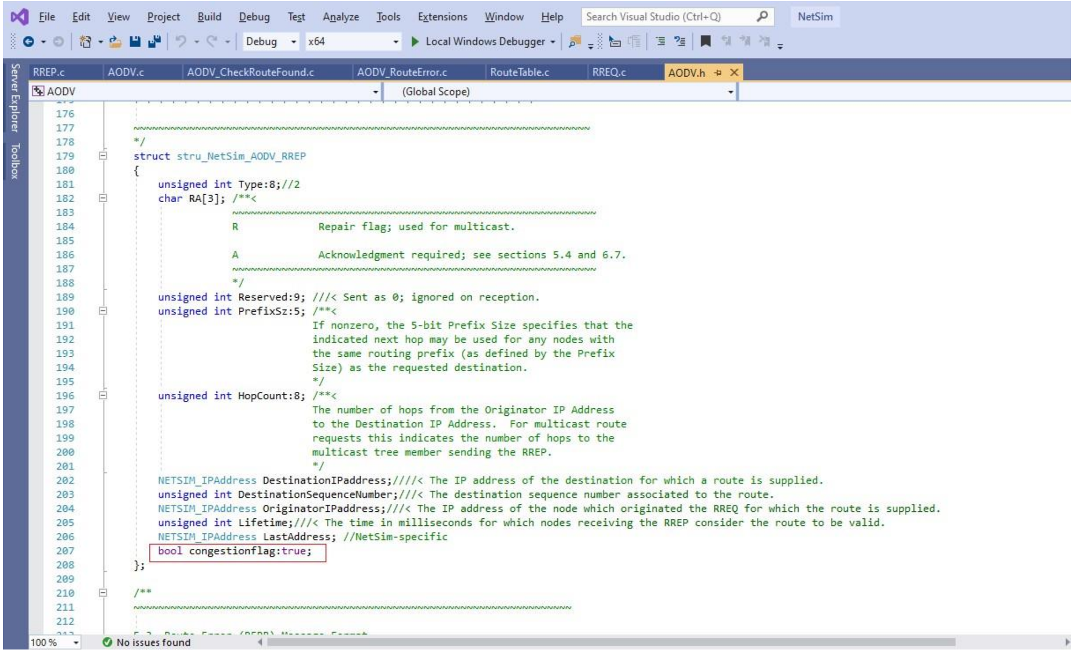
Ad hoc On-Demand Distance Vector (AODV) routing is one of the famous routing algorithms. Tremendous amounts of research on this protocol have been done to improve the performance. In this paper, a new control scheme, named congestion control AODV (CC-AODV), is proposed to manage the described routing 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.

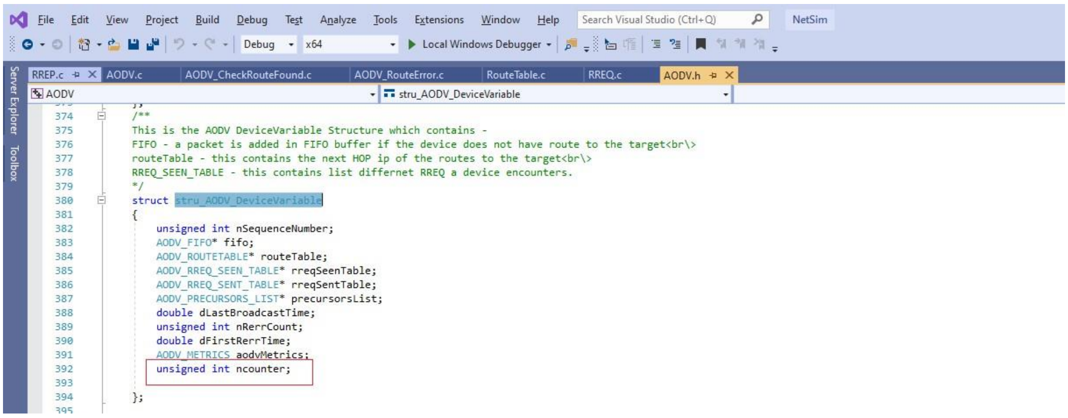
Procedure to implement CC-AODV in NetSim:

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

RREP.c | AODV.c | AODV_CheckRouteFound.c | AODV_RouteError.c | RouteTable.c | RREQ.c | AODV.h

AODV | (Global Scope) | fn_NetSim_AODV_ProcessRREP(NetSim_EVENTDETAILS * pstruEventD

```
61  Deletes the RREQ entry from sent table and forwards the rrep if the device is not
62  the source node.
63  */
64  int fn_NetSim_AODV_ProcessRREP(NetSim_EVENTDETAILS* pstruEventDetails)
65  {
66      AODV_ROUTETABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable;
67      AODV_RREP* rrep = (AODV_RREP*)pstruEventDetails->pPacket->pstruNetworkData->Packet_RoutingProtocol;
68      //Update the routing table
69      if (rrep->DestinationIPaddress == aodv_get_curr_ip())
70          return 0;
71
72      if(rrep->congestionflag == true)
73          AODV_DEV_VAR(pstruEventDetails->nDeviceId)->ncounter++;
74
75      AODV_INSERT_ROUTE_TABLE(rrep->DestinationIPaddress,
76          rrep->DestinationSequenceNumber,
77          rrep->HopCount,
78          rrep->LastAddress,
79          pstruEventDetails->dEventTime+AODV_ACTIVE_ROUTE_TIMEOUT);
80      //Transmit the buffer
81      AODV_TRANSMIT_FIFO(AODV_DEV_VAR(pstruEventDetails->nDeviceId));
82      //Update the precursor list
83      AODV_INSERT_PRECURSOR(rrep->LastAddress);
84      AODV_UPDATE_ROUTE_TABLE(rrep->LastAddress,rrep->Lifetime);
85      if(!IP_COMPARE(aodv_get_curr_ip(),rrep->OriginatorIPaddress))
86      {
87          //Delete entry from sent table
88          AODV_RREQ_SENT_TABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->rreqSentTable;
89          while(table)
90          {
91              if(!IP_COMPARE(table->DestAddress,rrep->DestinationIPaddress))
92              {
93                  IP_FREE(table->DestAddress);
94                  LIST_FREE((void**)&AODV_DEV_VAR(pstruEventDetails->nDeviceId)->rreqSentTable,table);
95                  break;
96              }
97              table = (AODV_RREQ_SENT_TABLE*)LIST_NEXT(table);
98          }
```

RREP.c | AODV.c | AODV_CheckRouteFound.c | AODV_RouteError.c | RouteTable.c | RREQ.c | AODV.h

AODV | (Global Scope) | fn_NetSim_AODV_ActiveRouteTimeout(NetSim_EVENTDETAILS * pstru

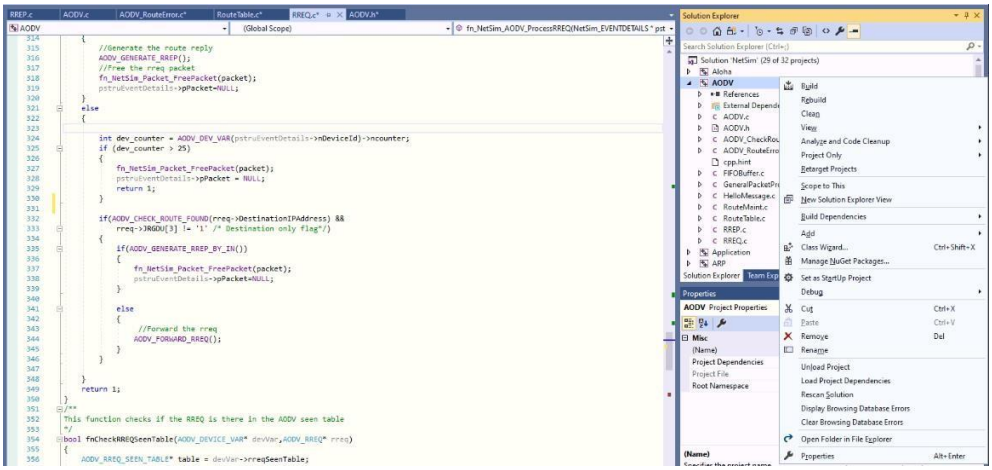
```
163 /**
164 This function adds the timeout event of a Route Table which is equal to the table_LifeTime
165 */
166 int fn_NetSim_AODV_ActiveRouteTimeout(NetSim_EVENTDETAILS* pstruEventDetails)
167 {
168     int flag = 0;
169     NETSIM_IPAddress dest = (NETSIM_IPAddress)pstruEventDetails->szOtherDetails;
170     AODV_ROUTETABLE* table = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable;
171     while(table)
172     {
173         if(!IP_COMPARE(table->DestinationIPAddress,dest))
174         {
175             if(table->Lifetime <= pstruEventDetails->dEventTime)
176             {
177                 AODV_ROUTETABLE* temp = LIST_NEXT(table);
178                 IP_FREE(table->DestinationIPAddress);
179                 IP_FREE(table->NextHop);
180                 LIST_FREE(&AODV_DEV_VAR(pstruEventDetails->nDeviceId)->routeTable,table);
181                 AODV_DEV_VAR(pstruEventDetails->nDeviceId)->ncounter--;
182                 table = temp;
183                 continue;
184             }
185             else
186             {
187                 //Add new time out event
188                 pstruEventDetails->dEventTime = table->Lifetime;
189                 fnpAddEvent(pstruEventDetails);
190                 flag = 1;
191             }
192         }
193         table=(AODV_ROUTETABLE*)LIST_NEXT(table);
194     }
195     if(!flag)
196         IP_FREE(dest);
197     return 1;
198 }
199
```

```
#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++)
```

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

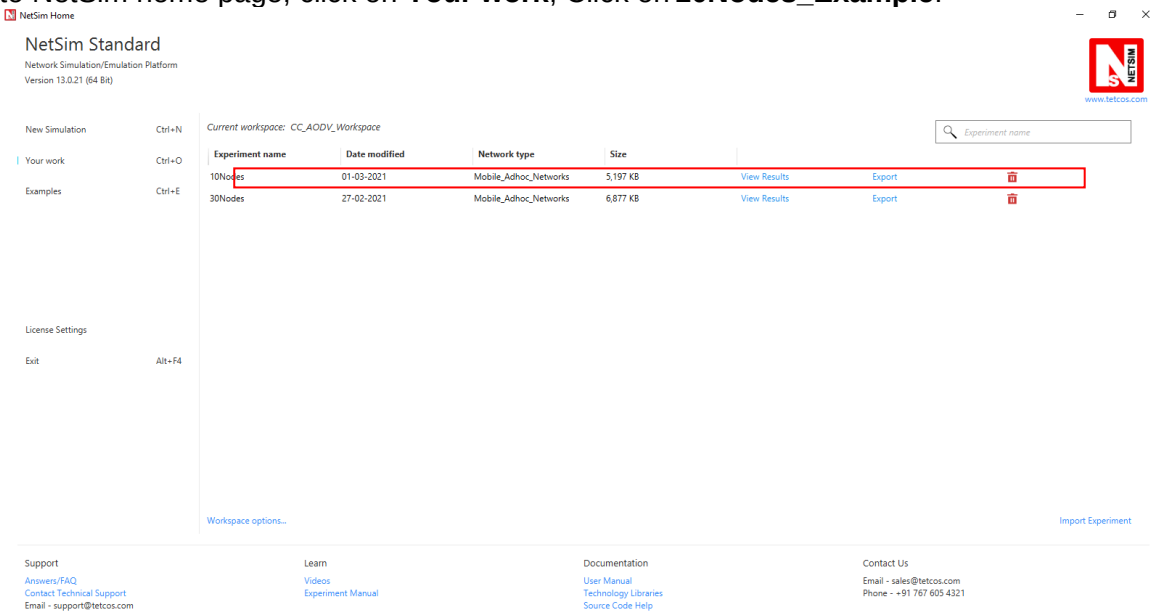
```
        //Free the rreq packet
        fn_NetSim_Packet_FreePacket(packet);
        pstruEventDetails->pPacket=NULL;
    }
    else
    {

        int dev_counter = AODV_DEV_VAR(pstruEventDetails->nDeviceId)->ncounter;
        if (dev_counter > 25)
        {
            fn_NetSim_Packet_FreePacket(packet);
            pstruEventDetails->pPacket = NULL;
            return 1;
        }

        if(AODV_CHECK_ROUTE_FOUND(rreq->DestinationIPAddress) &&
            rreq->JRGDU[3] != '1' /* Destination only flag*/)
        {
            if(AODV_GENERATE_RREP_BY_IN())
            {
                fn_NetSim_Packet_FreePacket(packet);
                pstruEventDetails->pPacket=NULL;
            }
```

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



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

7. Go to NetSim home page, click on **Your work**, Click on **10Nodes_Example**.



8. 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.

Result:

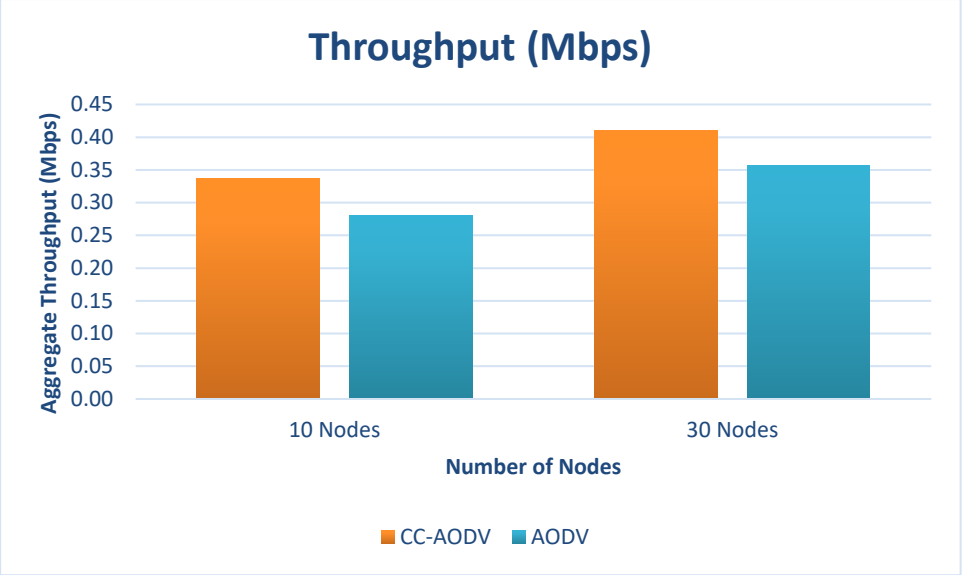
Performance of CC-AODV has been compared with other reactive protocol AODV based on different performance metrics such as Throughput, End to End delay etc.

Number of Nodes	AODV Aggregate Throughput (Mbps)	CC_AODV Aggregate Throughput (Mbps)
10Nodes	0.28	0.33
30Nodes	0.35	0.40

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	5462760.29	2004123.19
30Nodes	6534879.47	293415.94

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

Table 2 demonstrate that AODV has higher End-to-End performance than the CC-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: