Congestion Control AODV (CC-AODV)

Software: 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.

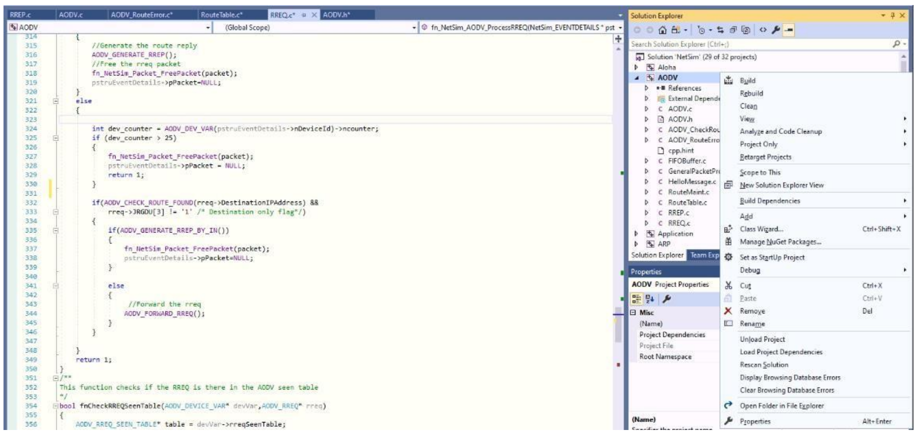
Introduction

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.

Steps to simulate

1. Open the Source codes in Visual Studio by going to Your work-> Workspace Options and Clicking on Open code button in NetSim Home Screen window.
2. In Visual Studio, Right click on the AODV Project and select rebuild.



```
	{
		//Generate the route reply
		AODV_GENERATE_RREP();
		//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;
			}

			else
			{
				 //Forward the rreq
				AODV_FORWARD_RREQ();
			}
		}

	}
	return 1;
}
/**
This function checks if the RREQ is there in the AODV seen table
*/
bool fnCheckRREQSeenTable(AODV_DEVICE_VAR* devVar,AODV_RREQ* rreq)
{
	AODV_RREQ_SEEN_TABLE* table = devVar->rreqSeenTable;
```

Figure 1: Screen shot of NetSim project source code in Visual Studio

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

Example

1. The CC_AODV_Workspace 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 10Nodes from the list of experiments.

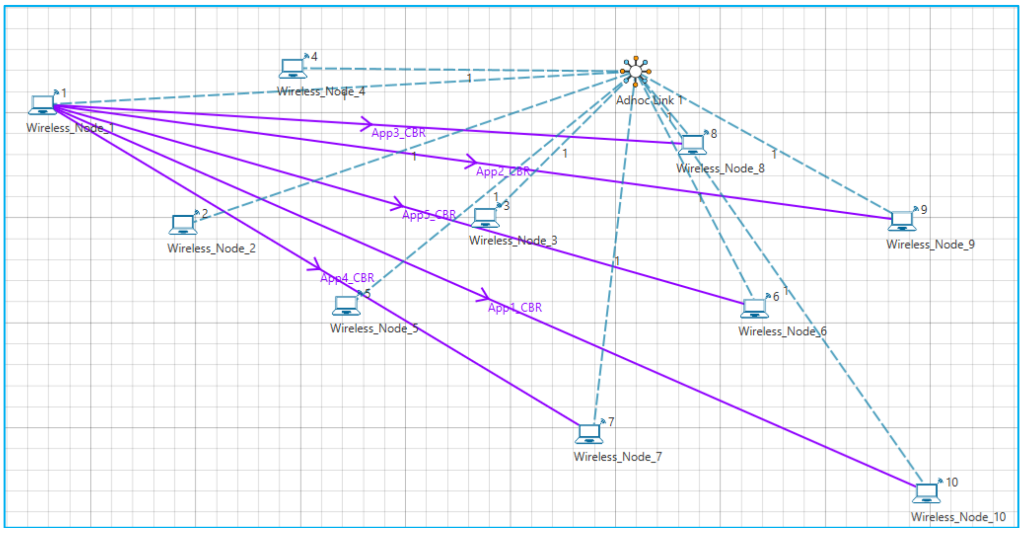


Figure 2: Network topology in this project

2. 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.

Results and discussion

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.

Throughput Comparison of CC-AODV and AODV

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.

Graph

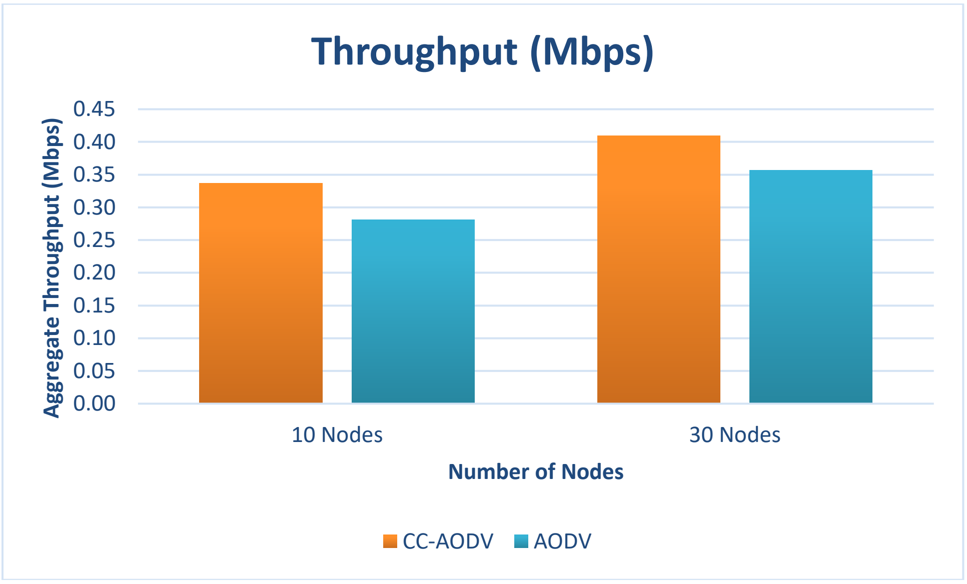


Figure 3: Plot comparison between of Aggregate Throughput AODV and CC_AODV

Delay Comparison of CC-AODV and AODV

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.

Graph

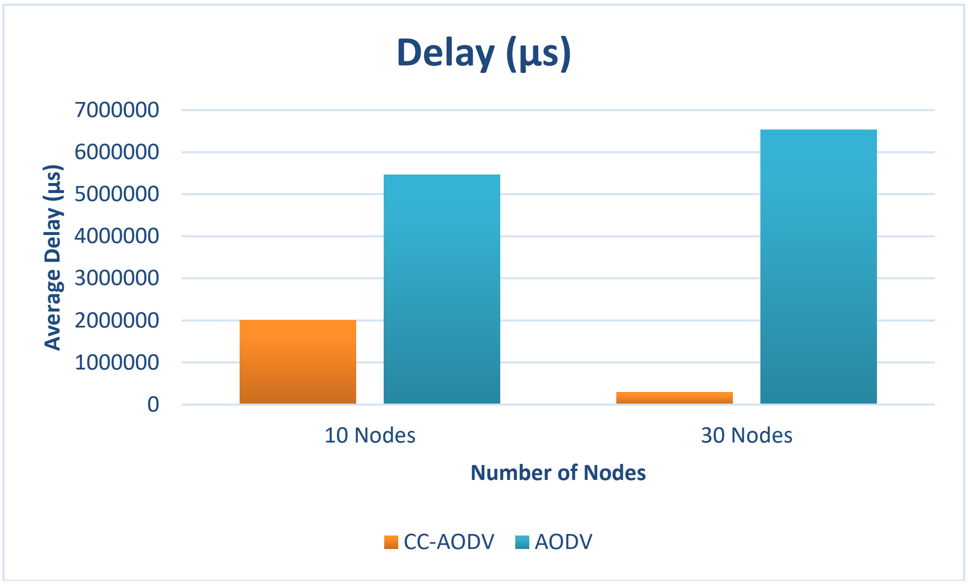


Figure 4: Plot comparison between of Aggregate Throughput AODV and CC_AODV

Appendix: NetSim source code modifications

Changes to RREP structure stru_NetSim_AODV_RREP, in AODV.h file, within AODV project

```
/* This is used to add a Congestion flag for implementing CC-AODV*/

struct stru_NetSim_AODV_RREP
{
unsigned int Type:8;//2
char RA[3]; /**<
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
R          Repair flag; used for multicast.

A          Acknowledgment required; see sections 5.4 and 6.7.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/
unsigned int Reserved:9; ///< Sent as 0; ignored on reception.
unsigned int PrefixSz:5; /**<
If nonzero, the 5-bit Prefix Size specifies that the
indicated next hop may be used for any nodes with
the same routing prefix (as defined by the Prefix
Size) as the requested destination.
*/
unsigned int HopCount:8; /**<
The number of hops from the Originator IP Address
to the Destination IP Address.  For multicast route
requests this indicates the number of hops to the
multicast tree member sending the RREP.
*/
NETSIM_IPAddress DestinationIPaddress;////< The IP address of the destination for which a route is
 supplied.
unsigned int DestinationSequenceNumber;///< The destination sequence number associated to the route.
NETSIM_IPAddress OriginatorIPaddress;///< The IP address of the node which originated the RREQ for
 which the route is supplied.
unsigned int Lifetime;///< The time in milliseconds for which nodes receiving the RREP consider the route to
 be valid.
NETSIM_IPAddress LastAddress; //NetSim-specific
bool congestionflag : true;
};
```

Changes to RREP structure stru_AODV_DeviceVariable, in AODV.h file, within AODV project

```
/* This is used to add a Congestion congestion counter for implementing CC-AODV*/

struct stru_AODV_DeviceVariable
{
        unsigned int nSequenceNumber;
        AODV_FIFO* fifo;
        AODV_ROUTETABLE* routeTable;
        AODV_RREQ_SEEN_TABLE* rreqSeenTable;
        AODV_RREQ_SENT_TABLE* rreqSentTable;
        AODV_PRECURSORS_LIST* precursorsList;
        double dLastBroadcastTime;
        unsigned int nRerrCount;
        double dFirstRerrTime;
        AODV_METRICS aodvMetrics;
        unsigned int ncounter;

};
```

Changes to fn_NetSim_AODV_ProcessRREP(), in RREP.c file, within AODV project

```
/* This is used to modified suitably to Increment, Decrement the congestion counter accordingly */

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);
```

Changes to fn_NetSim_AODV_ActiveRouteTimeout(), in RouteTable.c file, within AODV project

```
/*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;
                        pstruEventDetails->szOtherDetails = dest;
                        fnpAddEvent(pstruEventDetails);
                        flag = 1;
                }
        }
        table=(AODV_ROUTETABLE*)LIST_NEXT(table);
}
if(!flag)
        IP_FREE(dest);
return 1;
}
```

Changes to fn_NetSim_AODV_GenerateRERR (), in AODV_RouteError.c file, within AODV project

```
/*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;
```

Changes to fn_NetSim_AODV_ProcessRREQ(), in RREQ.c file, within AODV project

```
/* modified suitably to check the value of the congestion counter in the received RREQ packet and
accordingly forward or drop the packet */
{
        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;
                }

                else
                {
                        //Forward the rreq
                        AODV_FORWARD_RREQ();
                }
        }
}

        return 1;
```