الملاحق

Source Code :الملاحق أ

```
#include "h/aodv.h"
// add MD5 definition
#include "MD5.c"
#include "stdio.h"
#include "h/aodvCostants.h"

#define max(a,b) (a>b ? a : b)
Define_Module_Like(AODV,Routing);
// ****************costructors and destructors of secondary objects****************
AODV::~AODV{}()
WaitingPkt::WaitingPkt؛{}()
WaitingPkt::~WaitingPkt؛{}()
OldReqs::OldReqs؛{}()
OldReqs::~OldReqs؛{}()
PrecursorElement::PrecursorElement؛{}()
PrecursorElement::~PrecursorElement؛{}()
BlackListElement::BlackListElement؛{}()
BlackListElement::~BlackListElement؛{}()
WaitingRREP::WaitingRREP؛{}()
WaitingRREP::~WaitingRREP؛{}()
RouteTableElement::RouteTableElement؛{}()
RouteTableElement::~RouteTableElement؛{}()
PartialStat::PartialStat(double lat, double th)
{
        latencySum = lat؛
        throughSum = th؛
        samples = 1؛
}؛
PartialStat::~PartialStat؛{}()
Statistics::Statistics()
{
        hopsSum = 0؛
        deliveredDataMsg = 0؛
        sendDataMsg = 0؛
        sentCtrlPkt =0؛
        sentDataPkt =0؛
        maxHop =0؛
}
Statistics::~Statistics؛{}()
//function used by  queue.inset  to set up a oredered queue
int compareFunc(cObject* a, cObject *b)
{
        RouteTableElement* l = (RouteTableElement*)a؛
        RouteTableElement* r = (RouteTableElement*)b؛
        return ( l->destId - r->destId)؛
}
//*****************************************************************
void AODV::initialize()
{
        d("AODV protocol simulator based on the IEEE-MANET Internet Draft  v.10")؛
        //initialize the local variables
        sequenceNumber = 0؛
        //counter to generate unique  RREQs
        reqId = 0؛
        pktHistogram.setName("paket kind histogram")؛
        pktHistogram.setRange(0,13);
        hopsHistogram.setName("hops number histogram")؛
        hopsHistogram.setRange(1,20)؛
        //give to the queue the sorting capability
        routeTab.setup(compareFunc)؛

        //let some vars to be editable from the TkEnv environment
        WATCH(sequenceNumber)؛
        WATCH(statistics.sentCtrlPkt)؛
        //schedule the first message tho initialize the send hello chain
        helloEvent  = new cMessage("sendHello",MK_SEND_HELLO,0,P_SEND_HELLO)؛
        char  *str = (char*) helloEvent؛
        scheduleAt(simTime()+0.5, helloEvent)؛
}

void AODV::handleMessage(cMessage *msg)
{
 cMessage* reply = NULL؛
 int test = NULL  ؛
 d("HANDLE message routine")؛
 if (msg->arrivedOn("fromApp"))
 {
          d("messasge arrived from app")؛
          reply = sendData(msg)؛
          broadcast(reply)؛
 }
 else
 {
        // collect the message kind
         pktHistogram.collect( msg->kind())؛
         switch(msg->kind())
        {
         case MK_SEND_HELLO:
                 d("sendHello")؛
                 reply = generateHELLOmsg()؛
                 printf("\n MK_SEND_HELLO \n")؛
                 broadcast(reply)؛
                 printf("\n END MK_SEND_HELLO \n")؛
                 break؛
         case MK_DELETE:
                 */ Note that the Lifetime field in the routing table plays a dual role  for an
                  *  active route it is the expiry time, and for an invalid route it
                  *  If a data packet is received for an  invalid route, the Lifetime field is
                  *  is the deletion time.updated to current time plus  DELETE_PERIOD.
```

```
            /*
             d("delete")؛
             reply = handleDelete(msg)؛
             broadcast(reply)؛
             break؛
 case HELLO:
             d("hello")؛
             printf("\n handle hello  \n")؛
             handleHELLO(msg)؛
             delete msg؛
             break؛
 case MK_FLUSH:
             d("flush")؛
            // A RREQ has been timed out
            // so do what has to be done
             reply = handleFlush(msg)؛
             broadcast(reply)؛
             break؛
 case RREQ:
             d("rreq "<<msg->name())؛
             reply = handleRREQ(msg)؛
             broadcast(reply)؛
             delete msg؛
             break؛
 case RREP:
            d("rrep")؛
            reply = handleRREP(msg)؛
            broadcast(reply)؛
            delete msg؛
            break؛
case RERR:
            d("rerr")؛
            reply = handleRERR(msg)؛
            broadcast(reply)؛
            delete msg؛
            break؛
case DATA:
            d("data")؛
            reply = handleData(msg)؛
            broadcast(reply)؛
            delete msg؛
            break؛
case RREP_ACK:
            d("ack")؛
            handleACK(msg)؛
            delete msg؛
            break؛
case MK_ESP_ACK:
            d("esp_ack")؛
            reply = handleESP_ACK(msg)؛
```

```
                broadcast(reply)؛
                break؛
        case MK_BLK_LIST:
                d("black list")؛
                handleBLK_LIST(msg)؛
                delete msg؛
                break؛
        }
  }
}
void AODV::finish()
{
        //I will write on a file instead of usa recordScalar() beacuse of a bug in this
        //function that rewrites the file on each run
        double lost=0؛
        dd("Hosts number..........."<< (int)parentModule()->par("numHost"))؛
        dd("Sent control pakets...."<<statistics.sentCtrlPkt)؛
        dd("Sent data pakets......."<<statistics.sentDataPkt)؛
        dd("Delivered data pakets.."<<statistics.deliveredDataMsg)؛
        if(statistics.deliveredDataMsg > 0)
                dd("Hops Avarage..........."<<statistics.hopsSum / statistics.deliveredDataMsg )؛
                PartialStat* cell؛
                recordScalar("Hosts number:............ ",(int)parentModule()->par("numHost"))؛
                recordScalar("Sent control pakets......... ",statistics.sentCtrlPkt)؛
                recordScalar("Sent data pakets............ ",statistics.sentDataPkt)؛
                recordScalar("Delivered data pakets....... ",statistics.deliveredDataMsg)؛
                lost= statistics.sentCtrlPkt+ statistics.sentDataPkt - statistics.deliveredDataMsg ؛
                for(int i=0; i<= statistics.maxHop; i++)
                {
                        cell = (PartialStat*) statistics.hopsV[i]؛
                        if(cell)
                        {
                                recordScalar("Hosts Id.......", i)؛
                                recordScalar("Per-Hop throughput misured... ",
                                cell->throughSum/cell->samples)؛
                                dd("Per-Hop throughput misured..."<<i<<" "<<
                                cell->throughSum / cell->samples);
                                recordScalar("Per-Hop latency misured..... ",
                                cell->latencySum / cell->samples)؛
                                dd("Per-Hop latency misured....."<<i<<" "<<
                                cell->latencySum / cell->samples)؛
                        }
                }
                if(statistics.deliveredDataMsg > 0)
                        recordScalar("Hops Avarage................",statistics.hopsSum /
                        statistics.deliveredDataMsg)؛
}
void AODV::broadcast(cMessage* reply)
{
        if(reply !=NULL)
```

```
    {
        int ttl؛
        d("send to mac:"<<reply->name()<<" "<<reply->kind؛(()
        ttl = (int) reply->par("ttl")-1؛
        if( ttl >= 0 )
        {
            reply->par("ttl") = ttl؛
            reply->par("hopCount") = 1+ (int)reply->par("hopCount")؛
            //add the source parmeter that is common to all the messages
            if(reply->hasPar("source"))
                reply->par("source") = (int)parentModule()->id()؛
            else
                reply->addPar("source") = (int)parentModule()->id()؛

            send(reply,"toMac");
            //send Hello only when helloEvent is extracted from the FES (event queue)
            //or the message(like data) do not make the route table to be refreshed
            if((reply->kind() != HELLO) && (reply->kind() != DATA))
                if (helloEvent->isScheduled())
                    cancelEvent( helloEvent )؛
            //only control packets make the other nodes refresh their route
            if( reply->kind() != DATA)
            scheduleAt(simTime()+HELLO_INTERVAL,helloEvent )؛
            if( (reply->kind() == RREQ) || (reply->kind() == RREP)||
            (reply->kind() == RERR) || (reply->kind() == RREP_ACK))
                    statistics.sentCtrlPkt؛++
        }
        else
        {
            d("ttl espired! the msg will not be sent":)؛
            delete reply؛
        }
    }
}
void AODV::waitForAck(cMessage* msg)
{
        d("waitForAck")؛
        //schedule a trigger to simulate an ACK failure
        WaitingRREP* e = new WaitingRREP()؛
        e->destId = (int) msg->par("originator")؛
        e->nextHopId = (int) msg->par("mac")؛
        e->trials = 1؛
        //pointer to the rreq message
        e->rreqMsg = new cMessage(*msg)؛
        //trigger
        e->espireEvent = new cMessage("rrep ack espired",MK_ESP_ACK,0,P_ESP_ACK)؛
        //pointer to the RREP  entry in that has failed to arrive
        e->espireEvent->addPar("element") = (WaitingRREP*) e؛
        waitingRrep.insert( (WaitingRREP*) e)؛
        scheduleAt(simTime()+ NEXT_HOP_WAIT, e->espireEvent)؛
```

```
}

cMessage* AODV::handleESP_ACK(cMessage* msg)
{
    bool done = false؛
    WaitingRREP*e = NULL؛
    d("handle MK_ESP_ACK")؛
    //ugly but it is the only way...
    e =  (WaitingRREP*) (cObject*) msg->par("element")؛
    d("RREP ACK timed out (the ack message is not arrived) check out what's to be done")؛
    e->trials؛++
    if(e->trials > RREP_RETRIES)
    {
        d("no more trials left...put the neig. in the black list")؛
        //flush the RREP buffer!
        waitingRrep.remove(e)؛
        //add the node to the black list
        BlackListElement* b = new BlackListElement()؛
        b->id = e->nextHopId؛
        b->removeEvent = new cMessage("remove from B.L.",MK_BLK_LIST,0,P_BLK_LIST)؛
        b->removeEvent->addPar("node") = (cObject*) b؛
        blackList.insert( (BlackListElement*) b )؛
        //scehdule the node removal from the blacklist
        scheduleAt(simTime()+BLACKLIST_TIMEOUT,b->removeEvent)؛
        //delete the message here because it has to be deleted only in this case
        delete msg؛
        //msg is stored in e so I have to delete it here rather than before
        delete e؛
        return NULL؛
    }
    else
    {
        d("there are more chance left")؛
        //retrasmit the stored rrep
        cMessage* rrep = new cMessage(*e->rreqMsg)؛
        //schedule the next ack time out event
        scheduleAt(simTime()+ NEXT_HOP_WAIT, e->espireEvent)؛
        return rrep؛
    }
}
void AODV::handleBLK_LIST(cMessage* msg)
{
    d("hanldle black list")؛
    BlackListElement* e = (BlackListElement*)(cObject*) msg->par("node")؛
    blackList.remove(e)؛
    delete e؛
}
bool AODV::isInBlackList(int node)
{
    cQueue :: Iterator iter(blackList,1)؛
```

```
        bool found = false؛
        BlackListElement* e = NULL؛
        d("isInBlackList")؛
        while( ( !iter.end() ) && ( !found))
        {
                e = (BlackListElement*) iter()؛
                if(e->id == node)
                {
                        found = true؛
                }
                else iter؛++
        }
        return found؛
}
cMessage* AODV::sendData(cMessage* msg)
{
        RouteTableElement *e = NULL؛
        d("sendData")؛
        //check for a route
        e = findNode(msg->par("dest"))؛
        if( (e == NULL) || (e->active==false))
        {
                cMessage* reply؛
                reply = bufferize(msg->par("dest"),msg->length())؛
                if(reply !=NULL)
                {
                        //schedule the RREQ failure
                        scheduleAt(simTime()+ 2 * TTL_START * NODE_TRAVERSAL_TIME ,reply)؛
                        reply = generateRREQmsg(e,msg->par("dest"),TTL_START)؛
                        addNewReq(reply)؛ //remember the rreq
                        return reply؛ //return the RREQ message that wil be sent out
                }
                else
                {       d("RREQ not generated")؛
                        return reply؛
                }
        }
        else
        {
                statistics.sentDataPkt؛++
                cMessage* m = generateDATAmsg(e,msg->length؛(()
                d("want to send data to a known destination "<<msg->par("dest"))؛
                return m؛
        }؛
}
cMessage* AODV::bufferize(int dest,int pktSize)
{
        bool found = false؛
        WaitingPkt* p = NULL؛
        d("bufferize")؛
```

```
    for(cQueue:: Iterator iter(pktBuffer,1); !iter.end(); iter++)
    {
        p = (WaitingPkt*) iter()؛
        if(p->dest == dest)
        {
            //if there is a RREQ at work just add a new pkt
            p->pktNum؛++
            return NULL؛
        }
    }
    //this is a new paket : create the message that make the RREQ msg to be timed-out and reseded
    p = new WaitingPkt؛
    p->dest = dest؛
    p->trial = 1؛
    p->pktNum = 1؛
    p->pktSize = pktSize؛
    //RREQ time out trigger
    p->deleteEvent = new cMessage("RREQ time out",MK_FLUSH,P_FLUSH)؛
    p->deleteEvent->addPar("dest") = dest؛
    p->deleteEvent->addPar("ttl") = TTL_START؛
    pktBuffer.insert(p)؛
    return p->deleteEvent؛
{
cMessage* AODV::handleFlush(cMessage* msg)
{
    WaitingPkt* p = NULL؛
    cMessage* reply = NULL؛
    RouteTableElement *e = NULL؛
    d("handleFlush")؛
    for(cQueue::Iterator iter(pktBuffer,1); !iter.end(); iter++)
    {
        p = (WaitingPkt*) iter()؛
        if(p->dest == (int) msg->par("dest"))
        {
            d("RREQ timed out (RREP is not arrived in time)")؛
            //Perkins.... Each attempt increments the RREQ ID
            //field in the RREQ packet.  The RREQ can be broadcast with
            //TTL =   NET_DIAMETER up to a maximum of RREQ_RETRIES times.
            if ((int)msg->par("ttl") == NET_DIAMETER)
                    p->trial؛++
            if(p->trial > RREQ_RETRIES)
            {
                d("data trasmissiond aborted: deleting the out data buffer")؛
                pktBuffer.remove(p)؛
                delete p؛
                delete msg؛
                return NULL؛
        }
            else
            {
                //try a new RREQ
                d("RREQ timed out: retrasmit")؛
                int ttl = (int)msg->par("ttl") + TTL_INCREMENT؛
                ttl = ttl < TTL_THRESHOLD ? ttl : NET_DIAMETER؛
                msg->par("ttl") = ttl؛
                scheduleAt(simTime()+ ( 2 * ttl * NODE_TRAVERSAL_TIME) ,
                p->deleteEvent)؛
                reply = generateRREQmsg(e,msg->par("dest"),ttl)؛
                addNewReq(reply)؛
                reply->setLength(1024)؛
                return reply؛
            }
        }
    }
}
cMessage* AODV::handleDelete(cMessage* msg)
{
    RouteTableElement *e = NULL؛
    cMessage* reply = NULL؛
    int  err = 0; //checkRouteRep needs a int& as the 3rd par
    d("hndle Delete")؛
    e =(RouteTableElement*)(cObject*) msg->par("node")؛
    d("the route to "<<e->destId<<" is timed out".)؛
    if(e->active)
    {
        d("SET the route as inagible")؛
        //route has exired, set it invalid and schedule the final Delete event
        e->active = false؛
        e->seqNum؛++
        scheduleAt(simTime()+ DELETE_PERIOD, msg)؛
        reply=checkRouteTable(e,reply,err)؛
    }
    else
    {
        d(" delete ROUTE! to "<<e->destId)؛
        //unlink the route from the table
        routeTab.remove(e)؛
        delete e؛
    }
    return reply؛
    //the precursor list should be deleted by the destructor
}
cMessage* AODV::generateHELLOmsg()
{
    int signt؛
    cMessage* reply = new cMessage("Hello",HELLO,CTRL_PKT_SIZE,P_HELLO)؛
    d("genHello")؛
    reply->addPar("seqNumS") = sequenceNumber؛
    reply->addPar("hopCount") = 0؛
    //ttl is not needed due to the nature of the message that is never retrasmitted
```

```
        reply->addPar("ttl") = 1؛
        reply->addPar("mac") = BROADCAST؛
        // Add Signatue to Hello message
        printf("\nGENERATE HELLO NO: %d \n",sequenceNumber)؛
        bubble("Add Signatue to Hello message"!) ؛
        signt = (int) handleEncrypt(reply)؛
                        printf("\n HELLO SIGN:%d" , signt)؛
                        reply->addPar("Sign") = signt؛
                        printf("\n size of SIGN:%d" , sizeof(signt))؛

                        dd("************* Add Signatue to Hello message ************* \n")؛

        return reply؛
}
void AODV::handleHELLO(cMessage* msg)
{
        d("hndleHello")؛
        printf("HELLO Packet Size in bits is : %ld \n",msg->length())؛
        printf("HELLO Packet Size in Bytes is : %ld \n",msg->length()/8)؛
        for(int f=0; f<=5; f++)
        printf("\n par [%d] HELLO:%d \n" , f, (int) msg->par(f))؛
        if(msg->hasBitError())
        printf("HELLO has Bit error" )؛
        else printf("NO HELLO Bit error" )؛
        RouteTableElement *e = NULL؛
        e = findNode( (int) msg->par("source"))؛
        if( e == NULL)
                //add a new destination
                addNewDestination((int)msg->par("source")،
                                (int) msg->par("source")،
                                (int) msg->par("seqNumS")،
                                (int) msg->par("hopCount")،
                                simTime()+ACTIVE_ROUTE_TIMEOUT)؛
        else
        {
                //check whether there is the need of a refresh in the table data
                updateRouteTable(e،
                                (int)msg->par("seqNumS")،
                                (int)msg->par("hopCount")،
                                (int)msg->par("source")،
                                simTime()+ACTIVE_ROUTE_TIMEOUT
                )؛
        }
        d("fine handleHello")؛
}
cMessage* AODV::checkRouteTable(RouteTableElement*b,cMessage* reply,int& errors)
{
        RouteTableElement* e = NULL؛
        d("check RouteTable integrity")؛
        //if the brand new invalid destination has hosts in the precursor list...
```

```
if(!b->precList.empty(()
{
        char s[10]؛
        errors؛++
        d("the prec list of "<<b->destId<<"is not empty؛("
        //it can be not NULL!
        if(reply==NULL)
        {
                reply = new cMessage("RERR",RERR,CTRL_PKT_SIZE,P_RERR)؛
                reply->addPar("errCount") = 0؛
                reply->addPar("ttl") = 1؛
                reply->addPar("hopCount") = 0؛
                reply->addPar("seqNumS") = sequenceNumber؛
                reply->addPar("cc") ="fatto da check 1؛"
                reply->addPar("mac") = BROADCAST؛
        }
        sprintf(s,"%d",errors)؛
        reply->addPar(s) = b->destId؛
        //add the seq number of the known route
        sprintf(s,"seqNumD%d",errors)؛
        reply->addPar(s) = b->seqNum؛
        reply->par("errCount")= 1+(int)reply->par("errCount")؛
}
d("check other routes")؛
//if it is a neighbour that is no more reachable then there might be more ureachable desinations
if(b->hopCount == 1)
{
        d("it is a neighbour!")؛
        for( cQueue::Iterator it(routeTab,1) ; !it.end(); it++)
        {
                //check if there are destination that use the broken link as next hop
                e = (RouteTableElement*) (cObject*) it()؛
                if( (e->active) && (e->nextHop == b->nextHop))
                {
                        //the route is no more available
                        cancelEvent(e->deleteMessage)؛
                        scheduleAt(simTime()+DELETE_PERIOD, e->deleteMessage)؛
                        e->active = false؛
                        //there might be more invalid routes)-: ...
                        if(!e->precList.empty())
                        {
                                char s[20]؛
                                errors؛++
                                if(reply==NULL)
                                {
                                  reply = new cMessage("RERR",RERR,CTRL_PKT_SIZE,P_RERR)؛
                                  reply->addPar("errCount") = 0؛
                                  reply->addPar("ttl") = 1؛
                                  reply->addPar("hopCount") = 0؛
                                 // reply->addPar("cc") = "fatto da ck 2؛"
```

```
                         reply->addPar("seqNumS") = sequenceNumber؛
                         reply->addPar("mac") = BROADCAST؛
                    }
                    //add the unreachable nodes to the RERR message
                    sprintf(s,"%d",errors)؛
                    reply->addPar(s) = e->destId؛
                    reply->par("errCount") = errors؛
                    sprintf(s,"seqNumD%d",errors)؛
                    d(s)؛
                    reply->addPar(s) = e->seqNum؛
                }
                //there is not the need to check the precList of the new invalid routes
                //beacause all of these have as next hop the broken node "b" and so these
                //will be added to the RERR msg when checked by the iterator
            }
        }
    }
    return reply؛
}
cMessage* AODV::handleData(cMessage* msg)
{
    cMessage* reply = NULL؛
    RouteTableElement* e = NULL؛
    int docheck = FALSE؛
    d("hndleData")؛
    //check if the message has been precessed due to the promiscue mode
    if(parentModule()->id() != (int)msg->par("mac"))
    {
        d("received a message not for me...discarding"!)؛
        return NULL؛
    }
    d("arrived data msg for "<<msg->par("dest"))؛
    //check if the message is for this host
    if(parentModule()->id() == (int)msg->par("dest"))
    {
        d("DATA MESSAGE ARRIVED AT DESTINATION sent by "<<msg->par("source"))؛
        statistics.collect(msg, simTime())؛
        hopsHistogram.collect((int)msg->par("hopCount"))؛
        return NULL؛
    }
    else
    {
    //this host in not the final destination but the message states that this node is used to reach it.
        RouteTableElement* e = NULL؛
        d("Data message not for me but i am on the route, forwarding")؛
        e = findNode( (int)msg->par("dest"))؛
        if((e == NULL) || (!e->active))
        {
            //the route is unknown or expired do nothing...
            d("ERROR! The route to the destination is unknown"!)؛
```

```
                        return NULL؛
                }
                else
                {
                        d("Data message updated, forwarding"!)؛
                        reply = copyMessage(msg)؛
                        reply->par("mac") = e->nextHop؛
                        e->expiration = max(e->expiration,simTime() +ACTIVE_ROUTE_TIMEOUT)؛
                        //shift the invalidation of the route
                        cancelEvent(e->deleteMessage)؛
                        scheduleAt( e->expiration, e->deleteMessage)؛
                        e = findNode(e->nextHop)؛
                        e->expiration = max(e->expiration,simTime() +ACTIVE_ROUTE_TIMEOUT)؛
                        //shift the invalidation of the route
                        cancelEvent(e->deleteMessage)؛
                        scheduleAt(e->expiration, e->deleteMessage)؛
                        return reply؛
                }
        }
}
cMessage* AODV::generateDATAmsg(RouteTableElement*e,int size)
{
        int signt؛
        cMessage* m = new cMessage("Data",DATA,size,P_DATA)؛
        m->addPar("dest") = e->destId؛
        m->addPar("originator") = parentModule()->id()؛
        m->addPar("hopCount") = 0؛
        m->addPar("ttl") = e->hopCount؛
        m->addPar("mac") = e->nextHop؛
        m->addPar("sendingTime") = simTime()؛
        // Add Signatue to DATA message
        signt = (int) handleEncrypt(m)؛
        printf("\n DATA SIGN:%d" , signt)؛
        m->addPar("Sign") = signt؛
        printf("\n size of SIGN:%d" , sizeof(signt))؛
        dd("************* Add Signatue to DATA message ************* \n؛("
        bubble("Add Signatue to Data message"!) ؛
        return m؛
}
cMessage* AODV::handleRERR(cMessage *msg)
{
        cMessage* reply = NULL؛
        d("handle RERR")؛
        RouteTableElement * e؛
        int errors = 0؛
        //check if the trasmitting node is known
        e = findNode( (int)msg->par("source"))؛
        if( e == NULL)
                //add a new destination
                addNewDestination((int)msg->par("source")،
                (int)msg->par("source"),0,1،
                simTime()+ACTIVE_ROUTE_TIMEOUT)؛
        else
                updateRouteTable(e, e->seqNum،
                (int)msg->par("hopCount")،
                (int)msg->par("source")،
                simTime()+ACTIVE_ROUTE_TIMEOUT)؛
        if(msg->hasPar("errCount"))
        {
                char s[5],d[10]؛
                int k = (int) msg->par("errCount")؛
                for(int i = 1; i <= k; i++)
                {
                        //the parameter stores the nodes's id
                        d("number of errors: "<<k)؛
                        //extract the unreachable destination
                        sprintf(s,"%d",i)؛
                        //extract its sequence number
                        sprintf(d,"seqNumD%d",i)؛
                        //find the broken node and update the route table
                        e = findNode( (int) msg->par(s))؛
                        if((e != NULL)&&
                         (e->nextHop ==(int) msg->par("source"))&&
                         (e->active(     &&
                         (e->seqNum <= (int)msg->par(d)))
                        {
                                e->active = false؛
                                cancelEvent(e->deleteMessage)؛
                                scheduleAt(simTime()+ DELETE_PERIOD, e->deleteMessage)؛
                                //build an eventual new RERR message
                                reply = checkRouteTable(e,reply,errors)؛
                        }
                }       {
        return reply؛
}
cMessage* AODV::generateRERRmsg(RouteTableElement* e,int dest)
{
        int signt؛
        cMessage* msg = NULL؛
        d("genRERR")؛
        if((e == NULL) || !e->precList.empty())
        {
                msg = new cMessage("RERR",RERR,CTRL_PKT_SIZE,P_RERR)؛
                msg->addPar("1") = dest؛
                msg->addPar("errCount") = 1؛
                msg->addPar("seqNumD1") = (e != NULL? e->seqNum : 0)؛
                msg->addPar("seqNumS") = sequenceNumber؛
                msg->addPar("hopCount") = 0؛
                msg->addPar("ttl") = 1؛
                //msg->addPar("cc") ="fatto da genRERR؛"
                msg->addPar("mac") = BROADCAST؛
                // Add Signatue to RRER message
                signt = (int) handleEncrypt(msg)؛
                printf("\n RRER SIGN:%d" , signt)؛
                msg->addPar("Sign") = signt؛
                printf("\n size of SIGN:%d" , sizeof(signt))؛
                dd("************* Add Signatue to RRER message ************* \n")؛

        bubble("Add Signatue to RERR message"!) ؛
        }
        return msg؛
}
cMessage* AODV::handleRREP(cMessage *msg)
{
        cMessage* reply = NULL؛
        d("handle RREP")؛
        RouteTableElement *e, *f؛
        //check if the trasmitting neighbour node is known
        d("check the route to the neighbour node")؛
        e = findNode( (int)msg->par("source"))؛
        if( e == NULL)
                //add a new destination
                e = addNewDestination((int)msg->par("source")،
                                (int)msg->par("source"), 0,1،
                                simTime()+ACTIVE_ROUTE_TIMEOUT)؛
        else
                updateRouteTable(e, e->seqNum,1,(int)msg->par("source")،
                                simTime()+ACTIVE_ROUTE_TIMEOUT)؛
        if (parentModule()->id() ==  (int)msg->par("dest"))
        {
                d("received a rrep generated by me, deleting")؛
                return NULL؛
        }
        //check the node that generated the RREP
        d("check if the RREP originating node is known")؛
        //if false, dest is the neighbour that has alredy been checked
        if ((int) msg->par("dest") != (int) msg->par("source"))
        {
                e = findNode( (int)msg->par("dest"))؛
                d("rrep->Lifetime: "<<(int)msg->par("lifetime"))؛
                if( e == NULL)
                        //add a new destination
                        e = addNewDestination((int)msg->par("dest")،
                                        (int)msg->par("source")،
                                        (int)msg->par("seqNumD")،
                                        (int)msg->par("hopCount")،
                                        simTime()+(simtime_t)msg->par("lifetime"))؛
                else
                {
                        //check whether there is the need of a refresh in the table data
        updateRouteTable(e,(int)msg->par("seqNumD")،
        (int)msg->par("hopCount")،
        (int)msg->par("source")،
        simTime()+ (simtime_t) msg->par("lifetime"))؛
    }
}
//handle the RREP msg only if it is for this node
if( parentModule()->id() != (int) msg->par("mac"))
{
    d("received a RREP message that was not for me...siffing and discarding")؛
    return NULL؛
}
//check whether I am not the originator node of the RREQmessage, just forward it toward the right
if( parentModule()->id() != (int)msg->par("originator"))
{
    d("I am on the route back to the RREQ originator --> Forward RREP" )؛
    reply = copyMessage(msg)؛
    f = findNode( (int)msg->par("originator"))؛
    if(f == NULL)
    {
        d("ERROR! the route back to the RREQ originator is not known or espired؛("!
        return NULL؛
    }
    else
    {
        //add the RREP future next hop to the precursor list of the route toward
        //the RREP originating node (the RREQ target node)
        d("update the precursor list")؛
        f->updatePrecList((int)msg->par("source"))؛
        //e->updatePrecList((int)msg->par("source"))؛
        //send the ack message to the neighbour node
        broadcast( generateACKmsg(msg))؛
        //set the RREP future next hop
        reply->par("mac")= f->nextHop؛
         f->expiration = max(f->expiration,simTime() +ACTIVE_ROUTE_TIMEOUT)؛
        //shift the invalidation of the route
        if( f->deleteMessage->isScheduled())
            cancelEvent(f->deleteMessage)؛
        scheduleAt(f->expiration, f->deleteMessage)؛
        //setup the wait for the ack message
        waitForAck(reply)؛
        return reply؛
    }
}
else
{
    //I am the destination,now a new reoute is available and all data can be sent
    WaitingPkt* p = NULL؛
    bool done = false؛
    e = findNode( (int)msg->par("dest))؛
```

```
        d("I received the RREP that I needed")؛
        if(e == NULL)
        {
            d("error: newly aquired route unaviable"!)؛
            exit(1)؛
        }
        //send the ack message to the neighbour node
        broadcast( generateACKmsg(msg))؛
        d("....sending data".)؛
        cQueue::Iterator iter(pktBuffer,1)؛
        while( ( !iter.end() )  && ( !done))
        {
            p = (WaitingPkt*) iter()؛
            if( ( p->dest ==(int) msg->par("dest")))
            {
                //now it is possible to send data, cancel the RREQ failre trigger
                if(p->deleteEvent->isScheduled())
                    cancelEvent(p->deleteEvent)؛
                //send all the pakets
                for(int i=0 ;  i < p->pktNum ; i++)
                {
                    d("sending pkt"...)؛
                    reply = generateDATAmsg(e,p->pktSize؛(
                    statistics.sentDataPkt؛++
                    broadcast(reply)؛
                }
                pktBuffer.remove(p)؛
                delete (p)؛
                done = true؛
            }
            else iter++;
        }
        return NULL؛
    }
}
cMessage* AODV::generateRREPmsg(cMessage* msg, int seqNumD,int hops)
{
    int signt؛
    cMessage* rrep = new cMessage("RREP",RREP,CTRL_PKT_SIZE,P_RREP)؛
    d("genRREP")؛
    //spcify the node addtess for wich a route is supplyed
    rrep->addPar("dest") = msg->par("dest")؛
    //the destination seqNum associated to the route
    rrep->addPar("seqNumD") = seqNumD؛
    //rrep.originator is the address of the node which originated the RREQ
    rrep->addPar("originator") =(int)  msg->par("originator")؛
    //the time for wich nodes receiving the RREP consider the route to be valid
    rrep->addPar("lifetime") = MY_ROUTE_TIMEOUT؛
    //if the node is the destinatary of the rreq then hopcount is 0 otherwise it is the distance to the destination
    rrep->addPar("hopCount")=0؛
    //ask for a RREP-ACK. used for unidir.links
    rrep->addPar("flagA") = 1؛
    rrep->addPar("seqNumS") = sequenceNumber؛
    rrep->addPar("ttl") = hops؛
    rrep->addPar("mac") = msg->par("source")؛
    // Add Signatue to RREP message
    signt = (int) handleEncrypt(rrep)؛
    printf("\n RREP SIGN:%d" , signt)؛
    rrep->addPar("Sign") = signt؛
    printf("\n size of SIGN:%d" , sizeof(signt))؛
    dd("************* Add Signatue to RREP message ************* \n")؛
    return rrep؛
}
cMessage* AODV::generateRREQmsg(RouteTableElement* e,int dest,int ttl)
{
    int signt؛
    cMessage* reply = new cMessage("RREQ",RREQ,CTRL_PKT_SIZE,P_RREQ)؛
    d("genRREQ")؛
    reply->addPar("originator") = parentModule()->id()؛
    reply->addPar("dest") = dest؛
    reply->addPar("seqNumS") = sequenceNumber؛++
    reply->addPar("seqNumD") = (e == NULL? 0 : e->seqNum)؛
    reply->addPar("reqId") = reqId؛++
    reply->addPar("hopCount") = 0؛
    reply->addPar("ttl") = ttl؛
    reply->addPar("mac") = BROADCAST؛
    // Add Signatue to RREQ message
    signt = (int) handleEncrypt(reply)؛
    printf("\n RREQ SIGN:%d" , signt)؛
    reply->addPar("Sign") = signt؛
    printf("\n size of SIGN:%d" , sizeof(signt))؛
    bubble("Add Signatue to RREQ message"!) ؛
    dd("************* Add Signatue to RREQ message ************* \n")؛
    return reply؛
}
void AODV::handleACK(cMessage* msg)
{
    d("handle ACK")؛
    //if it is not for this node, discard
    if((int) msg->par("mac") != parentModule()->id())
    {
        d("received an ACK message not for me, discarding")؛
    }
    else
    {
        bool done = false؛
        WaitingRREP* e = NULL؛
        cQueue::Iterator iter(waitingRrep,1)؛
        while( ( !iter.end() )  && ( !done))
        {
            e = (WaitingRREP*) iter()؛
            if( ( e->destId ==(int) msg->par("originator")))
            {
                //it is the right rrep
                d("buffered RREP found and acked")؛
                if(e->espireEvent->isScheduled())
                    cancelEvent(e->espireEvent)؛
                waitingRrep.remove(e)؛
                //delete all the triggers
                delete e->espireEvent؛
                delete (e)؛
                done = true؛
            }
            else iter؛++
        }
    }
}
cMessage* AODV::generateACKmsg(cMessage* msg)
{
    cMessage* reply = new cMessage("RREP_ACK",RREP_ACK,CTRL_PKT_SIZE,P_RREP_ACK)؛
    d("generateACK")؛
    reply->addPar("mac") = msg->par("source")؛
    reply->addPar("originator") = msg->par("originator")؛
    reply->addPar("ttl") = 1؛
    reply->addPar("hopCount") = 0؛
    return reply؛
}
cMessage* AODV::handleRREQ(cMessage *msg)
{
    cMessage* reply؛
    RouteTableElement * e؛
    d("hndRREQ")؛
    //check if the message has been alredy received and processed
    if (! isNewReq(msg))
        return NULL؛
    else
        addNewReq(msg)؛
    //avoid the RREQ messages form black list's node
    if( isInBlackList(msg->par("source")))
    {
        d("received a RREQ msg from a node in the black list. DISCARDING")؛
        return NULL؛
    }
    //check the neighbour node that sent the message
    d("check the neighbour node that sent the message")؛
    e =  findNode( (int)msg->par("source"))؛
    if(e == NULL)
        //add a new neighbour but I don't know the seqNumber -->0 the hopCount is 1
        addNewDestination((int)msg->par("source")،
                (int)msg->par("source"),0,1,simTime()+ACTIVE_ROUTE_TIMEOUT)؛
```

```
else
        updateRouteTable(e,e->seqNum,1،
                        (int)msg->par("source")،
                        simTime()+ACTIVE_ROUTE_TIMEOUT)؛
//check if the originator node is known
d("check if the originator node is known")؛
if( (int)msg->par("originator") != (int) msg->par("source"))
{
        e = findNode( (int)msg->par("originator"))؛
        if( e == NULL)
                //add a new destination
                addNewDestination((int)msg->par("originator")،
                                (int)msg->par("source")،
                                (int)msg->par("seqNumS")،
                                (int)msg->par("hopCount")،
                                simTime()+REV_ROUTE_LIFE)؛
        else
                //check whether there is the need of a refresh in the table data
                updateRouteTable(e, (int)msg->par("seqNumS")،
                                (int)msg->par("hopCount")،
                                (int)msg->par("source")،
                                max(e->expiration,simTime()+ REV_ROUTE_LIFE))؛
}
//now check the destination
e = findNode((int) msg->par("dest"))؛
d("now check the RREQ destination")؛
if( parentModule()->id() == (int)msg->par("dest"))
{
         //I am the destination
        d("---- I am the RREQ destination generate RREP" )؛
        //a host must increment his seq.num before genereting a new RREP mesage
        sequenceNumber = max(sequenceNumber,(int)msg->par("seqNumD"))؛
        reply = generateRREPmsg(msg, sequenceNumber,(int) msg->par("hopCount"))؛
        //setup the wait for the ack message
        waitForAck(reply)؛
        return reply؛
}
else
if (e == NULL)
{
         //the destination is unknown copy the RREQ message,increment hopCount،
         //decrement TTL and rebroadcast it
        d("RREQ destination unknown, forwarding")؛
        reply = new cMessage(*msg)؛
        reply->par("hopCount") = (int) reply->par("hopCount")+1؛
        return reply؛
}
else
if( (e->seqNum < (int) msg->par("seqNumD") ) || (!e->active))
{
        //I am an intermediary node but
        //the informations in the routeTable are old.Do nothing
        d("the informations in the routeTable are old, do nothing؛("...
        return NULL؛
    }
    else
    {
        //I am an intermediary node.
        d("I am an intermediary node: I've got a route to the destination")؛
        //uses the last known sequence number as seqNumberD
        //rrep ttl is the sum of the rreq made hops and the hops remaining toward the destination
        reply = generateRREPmsg(msg, e->seqNum,(int) msg->par("hopCount"))؛
        reply->par("hopCount") =  e->hopCount؛
        reply->par("lifetime") = e->expiration – simTime()؛
        //add the source node into the precursor list of the destination
        e->updatePrecList( (int)msg->par("source"))؛
        //setup the wait for the ack message
        waitForAck(reply)؛
        return reply؛
    }
}
cMessage* AODV::copyMessage(cMessage* msg)
{
    //copy the data within the msg oject
    cMessage* newMsg = new cMessage(*msg)؛
    return newMsg؛
    d("cpy")؛
}
RouteTableElement* AODV::addNewDestination(int dest,int source,int seqN,int hopCount,simtime_t
expire)
{
    RouteTableElement* e = new RouteTableElement()؛
    d("addNewDest")؛
    char d[20]؛
    d("aggiungo :"<<dest)؛
    e->destId = dest؛
    //the neighbour node that sent the message
    e->nextHop = source؛
    e->seqNum = seqN؛
    e->hopCount = hopCount؛
    d("hops:"<<hopCount)؛
    e->expiration = expire؛
    d("add new dest : espire = "<<expire)؛
    e->active = true؛
    sprintf(d,"r.time out to %d",dest)؛
    e->deleteMessage = new cMessage(d,MK_DELETE,0,P_DELETE)؛
    e->deleteMessage->addPar("node") = (cObject*) e؛
    //if whithin a preconfigured period the route
    //will not be refreshed it will be cancelled
    scheduleAt(expire ,e->deleteMessage)؛
```

```
        routeTab.insert( (RouteTableElement*) e)؛
        return e؛
}
void AODV::updateRouteTable(RouteTableElement* e,int seqNum,int hopCount,int nextHop,
simtime_t time)
{
        d("updRoute per :"<<e->destId)؛
        if( (seqNum > e->seqNum|| (
        (( seqNum==e->seqNum) && (hopCount < e->hopCount ))(( seqNum==e->seqNum) &&
 (hopCount == e->hopCount) && (e->expiration < time)))
        {
                d("updating")؛
                //update the entry
                e->hopCount = hopCount؛
                e->nextHop = nextHop؛
                e->seqNum = seqNum؛
                e->active = true؛
                e->expiration = time؛
                //shift the invalidation of the route
                cancelEvent(e->deleteMessage)؛
                scheduleAt(e->expiration, e->deleteMessage)؛
        }
        else d("table not upadated")؛
}
RouteTableElement* AODV::findNode(int n)
{
        RouteTableElement * e = NULL؛
        d("find :"<<n)؛
        for( cQueue::Iterator iter(routeTab,1) ; !iter.end(); iter++)
        {
                e = (RouteTableElement*) iter()؛
                if(e->destId  == n )
                        return e؛
        }
        return NULL؛
}
void AODV::addNewReq(cMessage* msg)
{
        OldReqs* r = new OldReqs()؛
        d("addNewReq")؛
        r->originator = msg->par("originator")؛
        r->reqId = msg->par("reqId")؛
        r->time = simTime()؛
        oldReqs.insert( (OldReqs*) r)؛
}
bool AODV::isNewReq(cMessage *msg)
{
        int origin،     reqId؛
        d("isNewReq msg:"<<msg->name())؛
        origin = msg->par("originator")؛
	reqId = msg->par("reqId")؛
	OldReqs* r = NULL؛
	for(cQueue::Iterator iter(oldReqs,1); !iter.end(); iter++)
	{
		r =(OldReqs*) iter؛()
		if((r->originator == origin)&&(r->reqId == reqId))
		{
			//the same request can not be served twice
			//within a period of  PATH_TRAVERSAL_TIME
			if( (simTime()- r->time) >= PATH_TRAVERSAL_TIME)
			{
				//the request is processable remove just unlik r from the queue
				oldReqs.remove( (OldReqs*)r)؛
				delete r؛
				return true؛
			}
			else
				return false؛
		}
	}
	return true؛
}
bool RouteTableElement:: updatePrecList(int ip)
{
	PrecursorElement * e = NULL؛
	for( cQueue::Iterator iter(precList,1) ; !iter.end(); iter++)
	{
		e = (PrecursorElement*) iter()؛
		if(e->ip == ip)
		{
			 return false؛
		}
	}
	//ip is a new element so add it to the list
	e = new PrecursorElement()؛
	e->ip = ip؛
	precList.insert( (PrecursorElement*) e)؛
	return true؛
}
############################################################################ //
//Add encrypt method
int AODV::handleEncrypt(cMessage* msg)
{
	struct MD5Context md5c؛
	int cdata = TRUE؛
	int j=0؛
	int nn= NULL؛
	char *n=NULL, *test1=NULL؛
	unsigned char  signatures[16]؛
	char buffer [50]؛
```

```
        unsigned int ul؛
        printf("\n\n handleEncrypt \n")؛
        memset(buffer,0,sizeof(buffer))؛
        nn = (int)parentModule()->id()؛
        nn+=1000؛
        printf("\n\n int n:%d\n",nn)؛
        j = sprintf(buffer,"%d", nn) ؛
        printf("\n\n n :%s \n",buffer)؛
        test1= strcat(buffer ,":Sign ")؛
        n= buffer؛
        printf("\n\nENTERING ENCRYPT METHOD:\n")؛
        //******************ENCRYPT*************************
        MD5Init(&md5c)؛
        MD5Update(&md5c, (unsigned char *) n, strlen(n))؛
        MD5Final(signatures, &md5c)؛
        ul=0؛
        for(j=0; j<16; j++)
                ul += (int) signatures[j]؛
        printf ("\n ul= %lu\n" , ul)؛
        printf("\n\nEND ENCRYPT METHOD:\n")؛
        return (int) ul؛
}
//######################################################################
void Statistics::collect(cMessage* msg, double now)
{
        double latency = now - (double) msg->par("sendingTime")؛
        int i = (int)msg->par("hopCount")؛
        maxHop = max(maxHop, i)؛
        //if the vector cell is not empty
        PartialStat* cell = (PartialStat*)hopsV[i]؛
        if(cell)
        {
                cell->latencySum += latency؛
                cell->throughSum += msg->length() / latency؛
                cell->samples؛++
        }
        else
        {
                PartialStat* cell = new PartialStat(latency, msg->length() / latency)؛
                hopsV.addAt(i,cell)؛
        }
        hopsSum += i؛
        deliveredDataMsg؛++
}
```