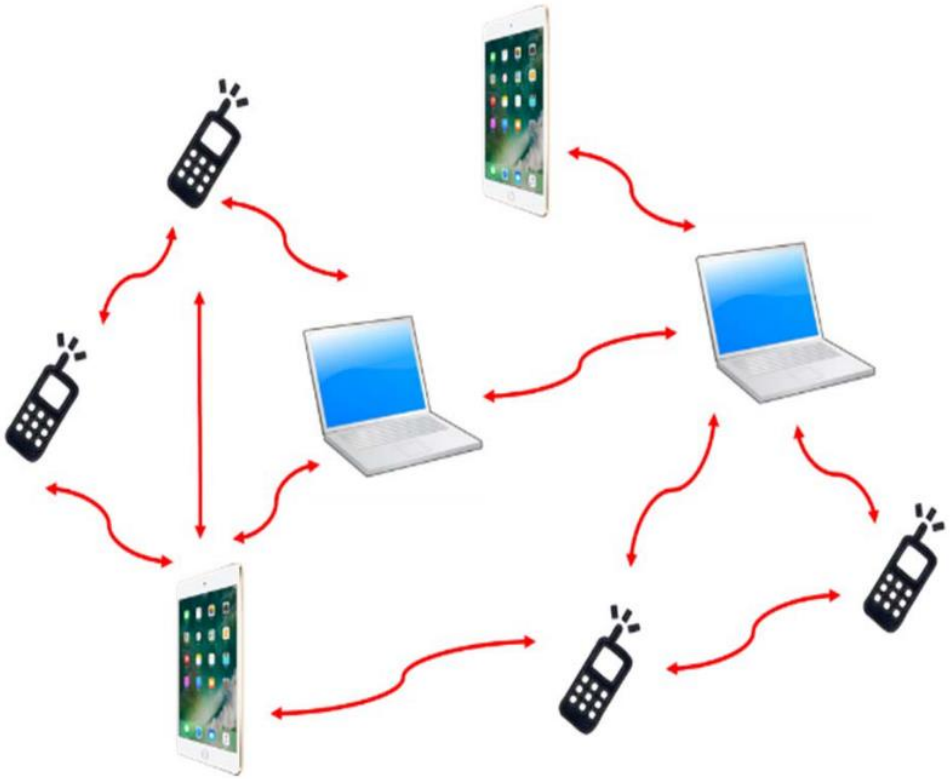
An Effective Multiple Paths Congestion Control AODV



Kawshik Kumar Paul
1705043
Undergrad Student
Dept of CSE, BUET

Conference Paper

CC-ADOV: An effective multiple paths congestion control AODV

January 2018

DOI:10.1109/CCWC.2018.8301758

Conference: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)

Authors:

Yefa Mai
California State University, Fresno

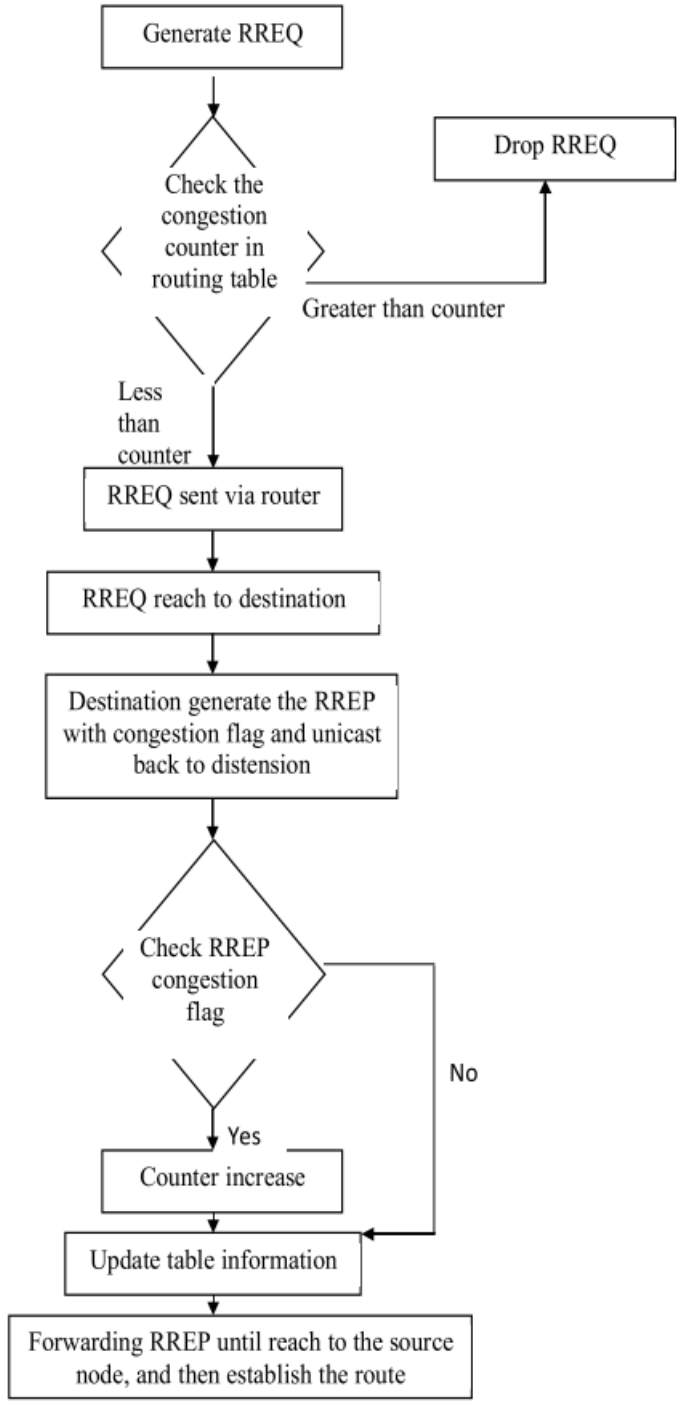
Fernando Molina Rodriguez

Nan Wang

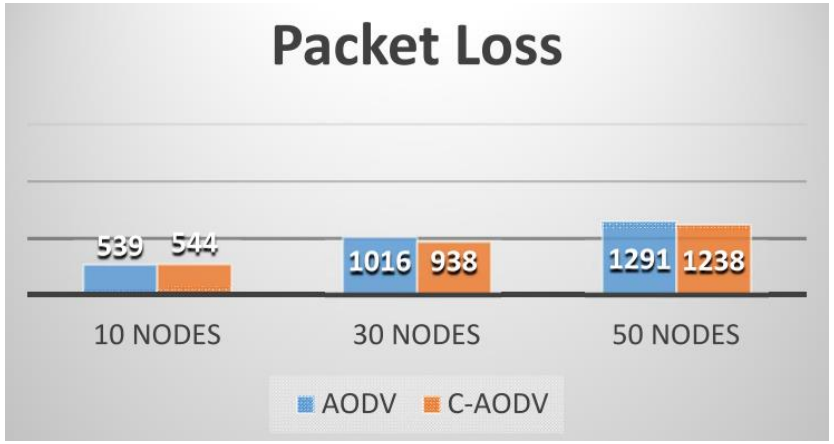
https://www.researchgate.net/publication/323562880_CC-ADOV_An_effective_multiple_paths_congestion_control_AODV

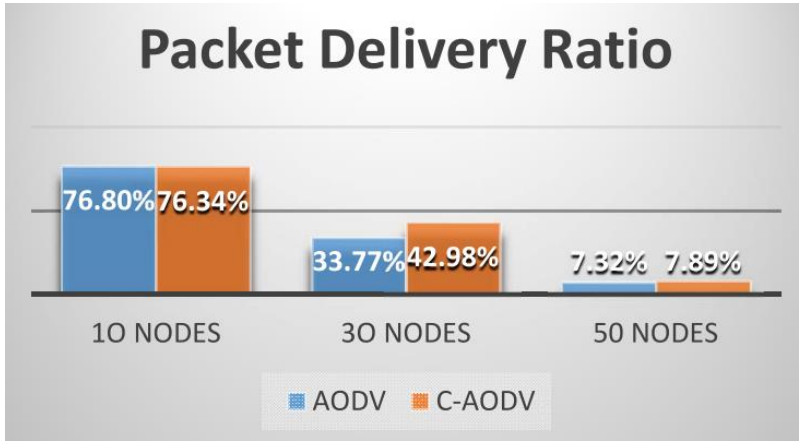
https://ieeexplore.ieee.org/document/8301758

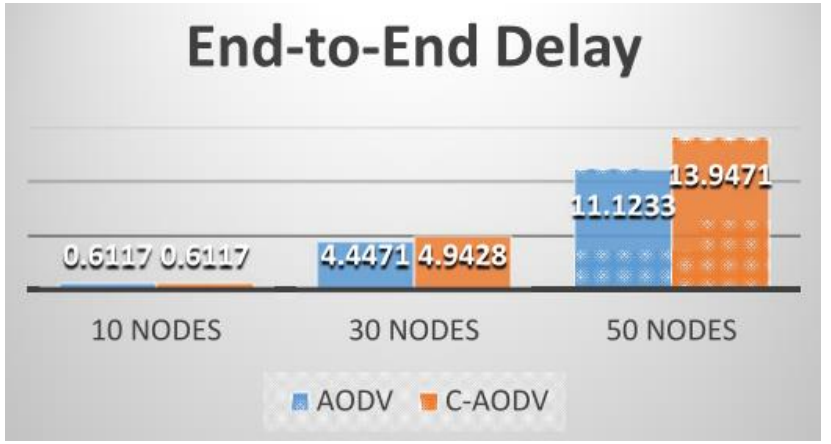
CC-AODV Flowchart

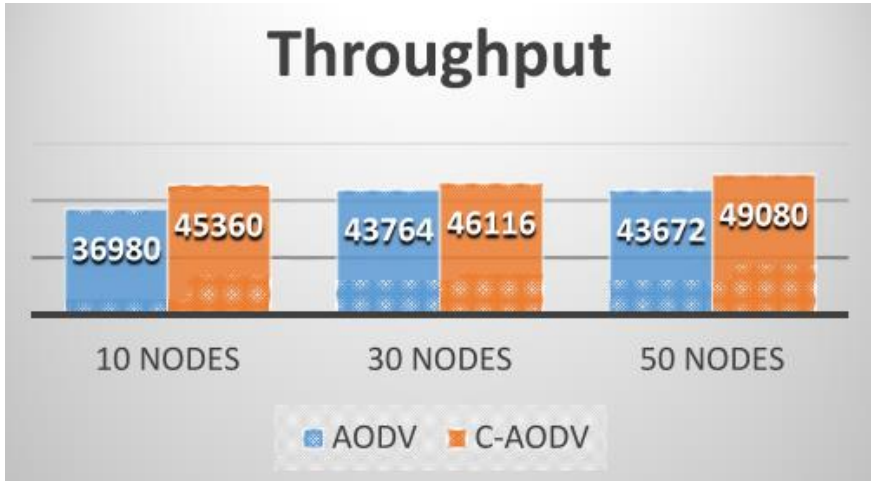


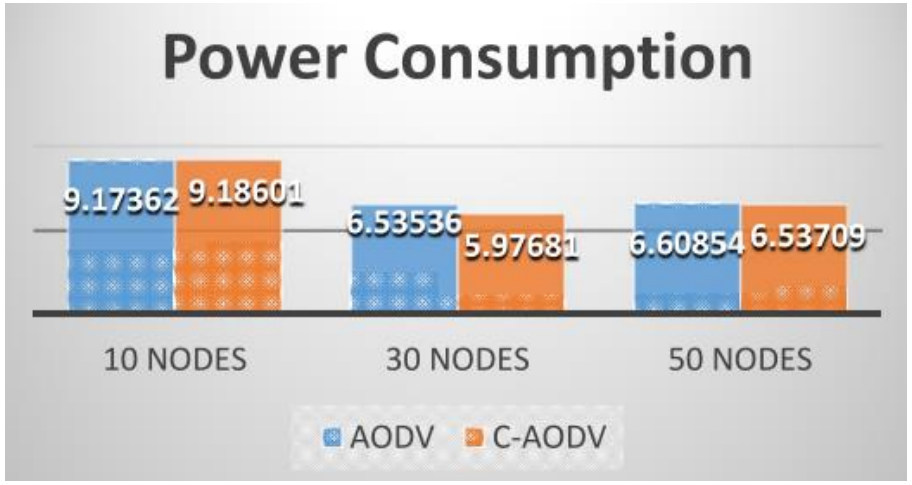
Expected Performance Measure



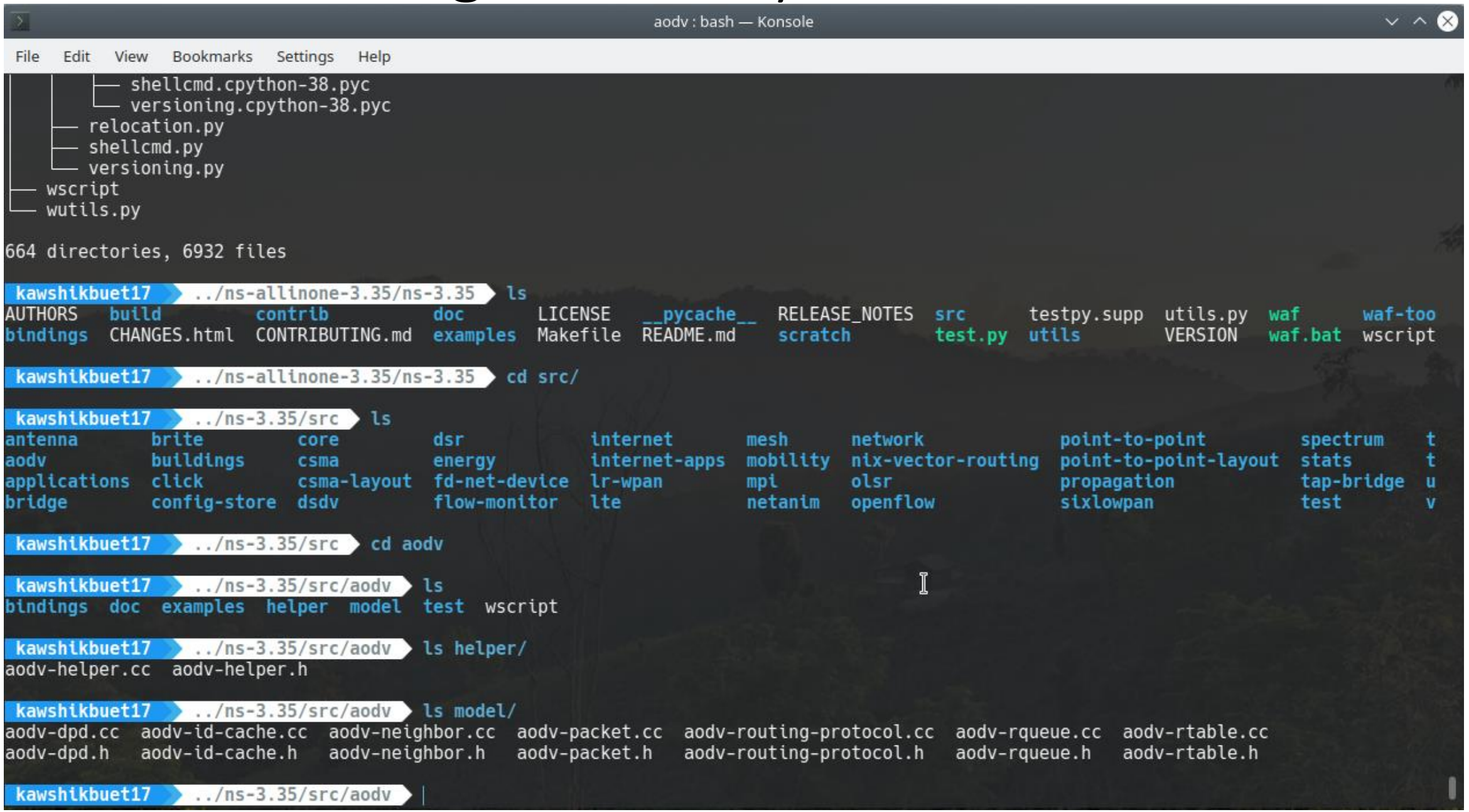








Finding Necessary Code Portion



Routing Path Change

```
kawshikbuet17 ../ns-3.35/src/aodv cd model/

kawshikbuet17 ../ns-3.35/src/aodv/model ls
aodv-dpd.cc  aodv-id-cache.cc  aodv-neighbor.cc  aodv-packet.cc  aodv-routing-protocol.cc  aodv-rqueue.cc  aodv-rtable.cc
aodv-dpd.h   aodv-id-cache.h   aodv-neighbor.h   aodv-packet.h   aodv-routing-protocol.h   aodv-rqueue.h   aodv-rtable.h

kawshikbuet17 ../ns-3.35/src/aodv/model grep SetNextHop *
aodv-routing-protocol.cc:       toOrigin.SetNextHop (src);
aodv-routing-protocol.cc:       toNeighbor.SetNextHop (src);
aodv-routing-protocol.cc:       toNeighbor.SetNextHop (rrepHeader.GetDst ());
aodv-rtable.h:  void SetNextHop (Ipv4Address nextHop)

kawshikbuet17 ../ns-3.35/src/aodv/model grep UpdateRoute *
aodv-routing-protocol.cc:       UpdateRouteLifeTime (dst, m_activeRouteTimeout);
aodv-routing-protocol.cc:       UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
aodv-routing-protocol.cc:               UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc:       UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc:           UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
aodv-routing-protocol.cc:           UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc:           UpdateRouteLifeTime (dst, m_activeRouteTimeout);
aodv-routing-protocol.cc:           UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
aodv-routing-protocol.cc:           UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
aodv-routing-protocol.cc:   UpdateRouteToNeighbor (sender, receiver);
aodv-routing-protocol.cc:RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
aodv-routing-protocol.cc:RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
aodv-routing-protocol.h:    bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);
aodv-routing-protocol.h:    void UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver);

kawshikbuet17 ../ns-3.35/src/aodv/model
```

aodv-routing-protocol.cc

aodv-routing-protocol.cc

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RecvRequest(Ptr<Packet>, Ipv4Address, Ipv4Address)

```
1193  }
1194
1195  void
1196  RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
1197  {
1198    NS_LOG_FUNCTION (this << "sender " << sender << " receiver " << receiver);
1199    RoutingTableEntry toNeighbor;
1200    if (!m_routingTable.LookupRoute (sender, toNeighbor))
1201      {
1202        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1203        RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1204                                    /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
1205                                    /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ m_activeRouteTimeout);
1206        m_routingTable.AddRoute (newEntry);
1207      }
1208    else
1209      {
1210        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1211        if (toNeighbor.GetValidSeqNo () && (toNeighbor.GetHop () == 1) && (toNeighbor.GetOutputDevice () == dev))
1212          {
1213            toNeighbor.SetLifeTime (std::max (m_activeRouteTimeout, toNeighbor.GetLifeTime ()));
1214          }
1215        else
1216          {
1217            RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ false, /*seqno=*/ 0,
1218                                        /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
1219                                        /*hops=*/ 1, /*next hop=*/ sender, /*lifetime=*/ std::max (m_activeRouteTimeout,
                                        toNeighbor.GetLifeTime ()));
1220            m_routingTable.Update (newEntry);
1221          }
1222      }
1223
1224  }
```

aodv-routing-protocol.cc

aodv-routing-protocol.cc

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv

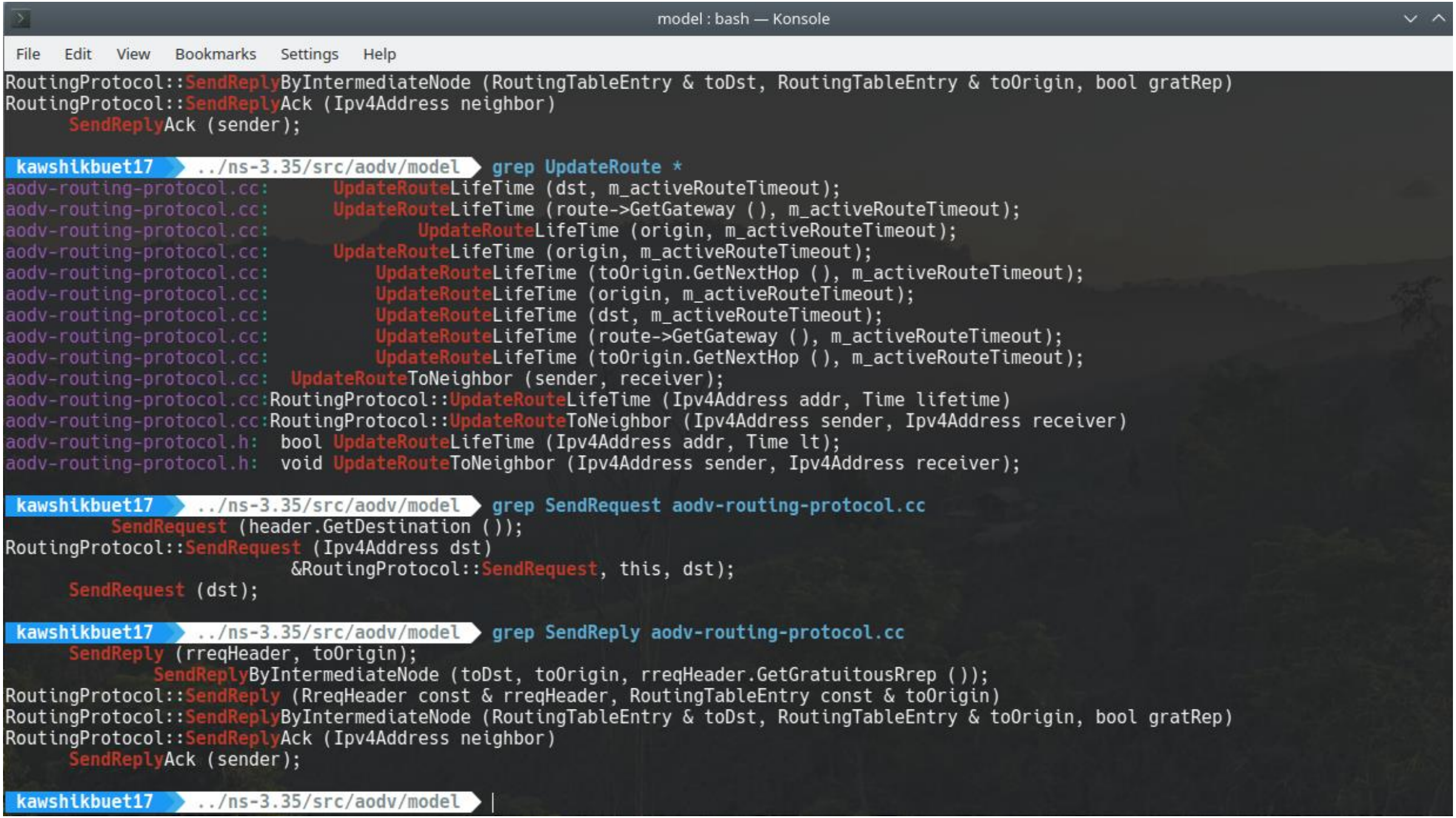
```
1261    /*
1262     *  When the reverse route is created or updated, the following actions on the route are also carried out:
1263     *  1. the Originator Sequence Number from the RREQ is compared to the corresponding destination sequence number
1264     *     in the route table entry and copied if greater than the existing value there
1265     *  2. the valid sequence number field is set to true;
1266     *  3. the next hop in the routing table becomes the node from which the  RREQ was received
1267     *  4. the hop count is copied from the Hop Count in the RREQ message;
1268     *  5. the Lifetime is set to be the maximum of (ExistingLifetime, MinimalLifetime), where
1269     *     MinimalLifetime = current time + 2*NetTraversalTime - 2*HopCount*NodeTraversalTime
1270     */
1271    RoutingTableEntry toOrigin;
1272    if (!m_routingTable.LookupRoute (origin, toOrigin))
1273      {
1274        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1275        RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ origin, /*validSeno=*/ true, /*seqNo=*/ rreqHeader.GetOriginSeqno (),
1276                                    /*iface=*/ m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0), /*hops=*/ hop,
1277                                    /*nextHop*/ src, /*timeLife=*/ Time ((2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime)));
1278        m_routingTable.AddRoute (newEntry);
1279      }
1280    else
1281      {
1282        if (toOrigin.GetValidSeqNo ())
1283          {
1284            if (int32_t (rreqHeader.GetOriginSeqno ()) - int32_t (toOrigin.GetSeqNo ()) > 0)
1285              {
1286                toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1287              }
1288          }
1289        else
1290          {
1291            toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
1292          }
1293        toOrigin.SetValidSeqNo (true);
1294        toOrigin.SetNextHop (src);
1295        toOrigin.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1296        toOrigin.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1297        toOrigin.SetHop (hop);
1298        toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_nodeTraversalTime),
1299                                        toOrigin.GetLifeTime ()));
1300        m_routingTable.Update (toOrigin);
1301        //m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
1302      }
1303
```

aodv-routing-protocol.cc

aodv-routing-protocol.cc

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv

```
1305    RoutingTableEntry toNeighbor;
1306    if (!m_routingTable.LookupRoute (src, toNeighbor))
1307      {
1308        NS_LOG_DEBUG ("Neighbor:" << src << " not found in routing table. Creating an entry");
1309        Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver));
1310        RoutingTableEntry newEntry (dev, src, false, rreqHeader.GetOriginSeqno (),
1311                                    m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),
1312                                    1, src, m_activeRouteTimeout);
1313        m_routingTable.AddRoute (newEntry);
1314      }
1315    else
1316      {
1317        toNeighbor.SetLifeTime (m_activeRouteTimeout);
1318        toNeighbor.SetValidSeqNo (false);
1319        toNeighbor.SetSeqNo (rreqHeader.GetOriginSeqno ());
1320        toNeighbor.SetFlag (VALID);
1321        toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receiver)));
1322        toNeighbor.SetInterface (m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0));
1323        toNeighbor.SetHop (1);
1324        toNeighbor.SetNextHop (src);
1325        m_routingTable.Update (toNeighbor);
1326      }
1327    m_nb.Update (src, Time (m_allowedHelloLoss * m_helloInterval));
1328
1329    NS_LOG_LOGIC (receiver << " receive RREQ with hop count " << static_cast<uint32_t> (rreqHeader.GetHopCount ())
1330                           << " ID " << rreqHeader.GetId ()
1331                           << " to destination " << rreqHeader.GetDst ());
1332
1333    //  A node generates a RREP if either:
1334    //  (i)  it is itself the destination,
1335    if (IsMyOwnAddress (rreqHeader.GetDst ()))
1336      {
1337        m_routingTable.LookupRoute (origin, toOrigin);
1338        NS_LOG_DEBUG ("Send reply since I am the destination");
1339        SendReply (rreqHeader, toOrigin);
1340        return;
1341      }
```


model : bash — Konsole
File Edit View Bookmarks Settings Help
RoutingProtocol::SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool gratRep)
RoutingProtocol::SendReplyAck (Ipv4Address neighbor)
SendReplyAck (sender);
kawshikbuet17 ../ns-3.35/src/aodv/model grep UpdateRoute *
aodv-routing-protocol.cc: UpdateRouteLifeTime (dst, m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (origin, m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (dst, m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (route->GetGateway (), m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteLifeTime (toOrigin.GetNextHop (), m_activeRouteTimeout);
aodv-routing-protocol.cc: UpdateRouteToNeighbor (sender, receiver);
aodv-routing-protocol.cc:RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
aodv-routing-protocol.cc:RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
aodv-routing-protocol.h: bool UpdateRouteLifeTime (Ipv4Address addr, Time lt);
aodv-routing-protocol.h: void UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver);
kawshikbuet17 ../ns-3.35/src/aodv/model grep SendRequest aodv-routing-protocol.cc
SendRequest (header.GetDestination ());
RoutingProtocol::SendRequest (Ipv4Address dst)
&RoutingProtocol::SendRequest, this, dst);
SendRequest (dst);
kawshikbuet17 ../ns-3.35/src/aodv/model grep SendReply aodv-routing-protocol.cc
SendReply (rreqHeader, toOrigin);
SendReplyByIntermediateNode (toDst, toOrigin, rreqHeader.GetGratuitousRrep ());
RoutingProtocol::SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)
RoutingProtocol::SendReplyByIntermediateNode (RoutingTableEntry & toDst, RoutingTableEntry & toOrigin, bool gratRep)
RoutingProtocol::SendReplyAck (Ipv4Address neighbor)
SendReplyAck (sender);
kawshikbuet17 ../ns-3.35/src/aodv/model

aodv-routing-protocol.cc

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RouteInput(Ptr<const Packet>, const Ipv4Header &, Ptr<const NetDevice>, UnicastForward(

```
427   void
428   RoutingProtocol::DeferredRouteOutput (Ptr<const Packet> p, const Ipv4Header & header,
429                                         UnicastForwardCallback ucb, ErrorCallback ecb)
430   {
431     NS_LOG_FUNCTION (this << p << header);
432     NS_ASSERT (p != 0 && p != Ptr<Packet> ());
433
434     QueueEntry newEntry (p, header, ucb, ecb);
435     bool result = m_queue.Enqueue (newEntry);
436     if (result)
437       {
438         NS_LOG_LOGIC ("Add packet " << p->GetUid () << " to queue. Protocol " << (uint16_t) header.GetProtocol ());
439         RoutingTableEntry rt;
440         bool result = m_routingTable.LookupRoute (header.GetDestination (), rt);
441         if (!result || ((rt.GetFlag () != IN_SEARCH) && result))
442           {
443             NS_LOG_LOGIC ("Send new RREQ for outbound packet to " << header.GetDestination ());
444             SendRequest (header.GetDestination ());
445           }
446       }
447   }
448
```

aodv-routing-protocol.cc

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RecvRequest(Ptr<Packet>, Ipv4Address, Ipv4Address)

SendReply Aa

```
1330                    << " ID " << rreqHeader.GetId ()
1331                    << " to destination " << rreqHeader.GetDst ());
1332
1333  //  A node generates a RREP if either:
1334  //  (i)  it is itself the destination,
1335  if (IsMyOwnAddress (rreqHeader.GetDst ()))
1336    {
1337      m_routingTable.LookupRoute (origin, toOrigin);
1338      NS_LOG_DEBUG ("Send reply since I am the destination");
1339      SendReply (rreqHeader, toOrigin);
1340      return;
1341    }
1342  /*
1343   * (ii) or it has an active route to the destination, the destination sequence number in the node's existing route table
1344   *      is valid and greater than or equal to the Destination Sequence Number of the RREQ, and the "destination only" fla
1345   */
1346  RoutingTableEntry toDst;
1347  Ipv4Address dst = rreqHeader.GetDst ();
1348  if (m_routingTable.LookupRoute (dst, toDst))
1349    {
1350      /*
1351       * Drop RREQ, This node RREP will make a loop.
1352       */
1353      if (toDst.GetNextHop () == src)
1354        {
1355          NS_LOG_DEBUG ("Drop RREQ from " << src << ", dest next hop " << toDst.GetNextHop ());
1356          return;
1357        }
1358      /*
1359       * The Destination Sequence number for the requested destination is set to the maximum of the corresponding value
1360       * received in the RREQ message, and the destination sequence value currently maintained by the node for the requeste
1361       * However, the forwarding node MUST NOT modify its maintained value for the destination sequence number, even if the
1362       * received in the incoming RREQ is larger than the value currently maintained by the forwarding node.
```

Thank You