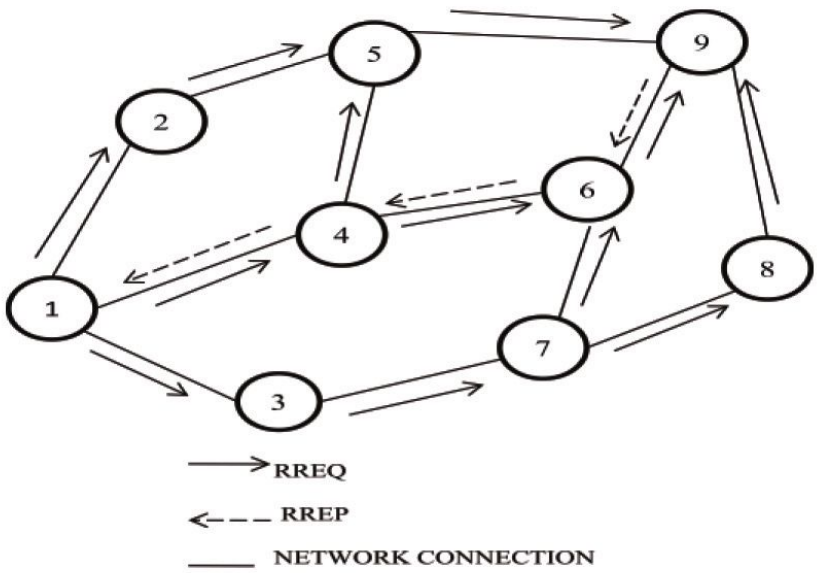
Workflow of IPv4 layer and Routing in NS3
AODV

Presented By:
1705043
1705053
1705060

Presentation 8: (Workflow of IPv4 layer and Routing in NS3 P2)

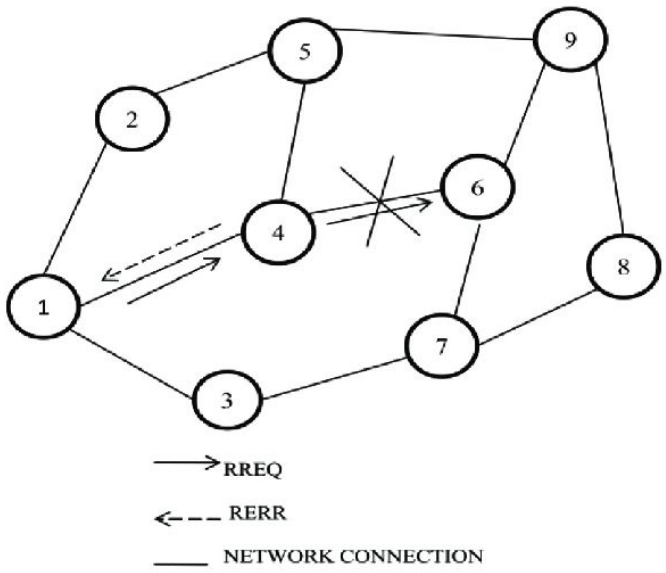
- Overview of a routing protocol (A1 & A2: AODV, B1 & B2: DSR)
    - Overview of packet header for this routing and where to update if we want to store additional information
    - Functions that handle data/control packet (Hello, Route Request etc) receive
    - Functions that decide forwarding and how they specify forwarding address
    - Functions that handle routing table update
    - Where data flows from/to TCP Layer and MAC Layer

AODV



ROUTE Discovery

- Discover all the routes using RREQ (Broadcast until destination reach)
- Destination will unicast RREP to source (Unicast)



ROUTE Maintenance

Overview of packet header for this routing (AODV)
and
where to update if we want to store additional information

AODV Packet Headers

- RreqHeader
- RrepHeader
- RerrHeader
- RrepAckHeader
- TypeHeader

RreqHeader

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |J|R|G|D|U|   Reserved          |   Hop Count   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            RREQ ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination IP Address                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Destination Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Originator IP Address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Originator Sequence Number                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RreqHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RreqHeader

```
130  */
131  class RreqHeader : public Header
132  {
133  public:
134    /**
135     * constructor
136     *
137     * \param flags the message flags (0)
138     * \param reserved the reserved bits (0)
139     * \param hopCount the hop count
140     * \param requestID the request ID
141     * \param dst the destination IP address
142     * \param dstSeqNo the destination sequence number
143     * \param origin the origin IP address
144     * \param originSeqNo the origin sequence number
145     */
146     RreqHeader (uint8_t flags = 0, uint8_t reserved = 0, uint8_t hopCount = 0,
147                 uint32_t requestID = 0, Ipv4Address dst = Ipv4Address (),
148                 uint32_t dstSeqNo = 0, Ipv4Address origin = Ipv4Address (),
149                 uint32_t originSeqNo = 0);
150
```

RreqHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RreqHeader

```
private:
  uint8_t        m_flags;          ///< |J|R|G|D|U| bit flags, see RFC
  uint8_t        m_reserved;       ///< Not used (must be 0)
  uint8_t        m_hopCount;       ///< Hop Count
  uint32_t       m_requestID;      ///< RREQ ID
  Ipv4Address    m_dst;            ///< Destination IP Address
  uint32_t       m_dstSeqNo;       ///< Destination Sequence Number
  Ipv4Address    m_origin;         ///< Originator IP Address
  uint32_t       m_originSeqNo;    ///< Source Sequence Number
};
```

RreqHeader

src > aodv > model > aodv-packet.cc > {} ns3 > {} aodv

```
//-----------------------------------------------------------------------------
// RREQ
//-----------------------------------------------------------------------------
RreqHeader::RreqHeader (uint8_t flags, uint8_t reserved, uint8_t hopCount, uint32_t requestID, Ipv4Address dst,
                        uint32_t dstSeqNo, Ipv4Address origin, uint32_t originSeqNo)
  : m_flags (flags),
    m_reserved (reserved),
    m_hopCount (hopCount),
    m_requestID (requestID),
    m_dst (dst),
    m_dstSeqNo (dstSeqNo),
    m_origin (origin),
    m_originSeqNo (originSeqNo)
{
}
```

RrepHeader

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |R|A|    Reserved     |Prefix Sz|   Hop Count   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Destination IP address                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Destination Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Originator IP address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Lifetime                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RrepHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RrepHeader

```
335  class RrepHeader : public Header
336  {
337  public:
338    /**
339     * constructor
340     *
341     * \param prefixSize the prefix size (0)
342     * \param hopCount the hop count (0)
343     * \param dst the destination IP address
344     * \param dstSeqNo the destination sequence number
345     * \param origin the origin IP address
346     * \param lifetime the lifetime
347     */
348    RrepHeader (uint8_t prefixSize = 0, uint8_t hopCount = 0, Ipv4Address dst =
349                Ipv4Address (), uint32_t dstSeqNo = 0, Ipv4Address origin =
350                Ipv4Address (), Time lifetime = MilliSeconds (0));
351    /**
```

RrepHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv

```
private:
  uint8_t       m_flags;                ///< A - acknowledgment required flag
  uint8_t       m_prefixSize;         ///< Prefix Size
  uint8_t             m_hopCount;         ///< Hop Count
  Ipv4Address   m_dst;              ///< Destination IP Address
  uint32_t      m_dstSeqNo;         ///< Destination Sequence Number
  Ipv4Address     m_origin;           ///< Source IP Address
  uint32_t      m_lifeTime;         ///< Lifetime (in milliseconds)
};
```

RrepHeader

src > aodv > model > aodv-packet.cc > {} ns3 > {} aodv

```
//-----------------------------------------------------------------------------
// RREP
//-----------------------------------------------------------------------------

RrepHeader::RrepHeader (uint8_t prefixSize, uint8_t hopCount, Ipv4Address dst,
                        uint32_t dstSeqNo, Ipv4Address origin, Time lifeTime)
  : m_flags (0),
    m_prefixSize (prefixSize),
    m_hopCount (hopCount),
    m_dst (dst),
    m_dstSeqNo (dstSeqNo),
    m_origin (origin)
{
  m_lifeTime = uint32_t (lifeTime.GetMilliSeconds ());
}

```

RerrHeader

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |N|          Reserved           |   DestCount   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Unreachable Destination IP Address (1)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Unreachable Destination Sequence Number (1)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|
|  Additional Unreachable Destination IP Addresses (if needed)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Additional Unreachable Destination Sequence Numbers (if needed)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RerrHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RerrHeader

```
557  class RerrHeader : public Header
558  {
559  public:
560    /// constructor
561    RerrHeader ();
562
563    /**
564     * \brief Get the type ID.
565     * \return the object TypeId
566     */
```

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RerrHeader

```
615  private:
616    uint8_t m_flag;            ///< No delete flag
617    uint8_t m_reserved;        ///< Not used (must be 0)
618
```

src > aodv > model > aodv-packet.cc > {} ns3 > {} aodv

```
517  //-----------------------------------------------------------------
518  // RERR
519  //-----------------------------------------------------------------
520  RerrHeader::RerrHeader ()
521    : m_flag (0),
522      m_reserved (0)
523  {
524  }
525
```

RrepAckHeader

```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Reserved   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RrepAckHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RrepAckHeader

```
503  class RrepAckHeader : public Header
504  {
505  public:
506    /// constructor
507    RrepAckHeader ();
508
509    /**
510     * \brief Get the type ID.
511     * \return the object TypeId
512     */
```

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > RrepAckHeader

```
526  private:
527    uint8_t       m_reserved; ///< Not used (must be 0)
528  };
```

src > aodv > model > aodv-packet.cc > {} ns3 > {} aodv

```
448
449  //-----------------------------------------------------------------------
450  // RREP-ACK
451  //-----------------------------------------------------------------------
452
453  RrepAckHeader::RrepAckHeader ()
454    : m_reserved (0)
455  {
456  }
457
```

TypeHeader

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > TypeHeader

```
class TypeHeader : public Header
{
public:
  /**
   * constructor
   * \param t the AODV RREQ type
   */
  TypeHeader (MessageType t = AODVTYPE_RREQ);

```

src > aodv > model > aodv-packet.h > {} ns3 > {} aodv > TypeHeader

```
  private:
    MessageType m_type; ///< type of the message
    bool m_valid; ///< Indicates if the message is valid
};
```

src > aodv > model > aodv-packet.cc > {} ns3 > {} aodv

```

TypeHeader::TypeHeader (MessageType t)
  : m_type (t),
    m_valid (true)
{
}
```

Functions That handle Data/Control Packet Receive

The following functions involve packet receive,

- void RecvAodv (Ptr< Socket > socket)
- void RecvRequest (Ptr< Packet > p, Ipv4Address receiver, Ipv4Address src)
- void RecvReply (Ptr< Packet > p, Ipv4Address my, Ipv4Address src)
- void RecvReplyAck (Ipv4Address neighbor)
- void RecvError (Ptr< Packet > p, Ipv4Address src)

Source : https://www.nsnam.org/docs/release/3.35/doxygen/aodv-routing-protocol_8cc_source.html

RecvAodv() Function

- This function receives and processes control packets.
- It has a socket parameter which is used to extract the packet.
- It checks the validity of the packet.
- Then calls the corresponding function of that packet type.
- It also updates the route to neighbor.

RecvAodv() Function

aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RecvAodv(Ptr<Socket>)

```
1119
1120  void
1121  RoutingProtocol::RecvAodv (Ptr<Socket> socket)
1122  {
1123    NS_LOG_FUNCTION (this << socket);
1124    Address sourceAddress;
1125
1126    Ptr<Packet> packet = socket-> RecvFrom(sourceAddress);
1127    InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom (sourceAddress);
1128    Ipv4Address sender = inetSourceAddr.GetIpv4 ();
1129    Ipv4Address receiver;
1130
1131    if (m_socketAddresses.find (socket) != m_socketAddresses.end ())
1132      {
1133        receiver = m_socketAddresses[socket].GetLocal ();
1134      }
1135    else if (m_socketSubnetBroadcastAddresses.find (socket) != m_socketSubnetBroadcastAddresses.end ())
1136      {
1137        receiver = m_socketSubnetBroadcastAddresses[socket].GetLocal ();
1138      }
1139    else
1140      {
1141        NS_ASSERT_MSG (false, "Received a packet from an unknown socket");
1142      }
1143    NS_LOG_DEBUG ("AODV node " << this << " received a AODV packet from " << sender << " to " << receiver);
1144
1145    UpdateRouteToNeighbor (sender, receiver);
```

RecvAodv() Function

aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RecvAodv(Ptr<Socket>)

```
1148
1149    TypeHeader tHeader (AODVTYPE_RREQ);
1150    packet->RemoveHeader (tHeader);
1151    if (!tHeader.IsValid ())
1152      {
1153        NS_LOG_DEBUG ("AODV message " << packet->GetUid () << " with unknown type received: " << tHeader.Get () << ". Drop");
1154        return; // drop
1155      }
1156    switch (tHeader.Get ())
1157      {
1158      case AODVTYPE_RREQ:
1159        {
1160          RecvRequest (packet, receiver, sender);
1161          break;
1162        }
1163      case AODVTYPE_RREP:
1164        {
1165          RecvReply (packet, receiver, sender);
1166          break;
1167        }
1168      case AODVTYPE_RERR:
1169        {
1170          RecvError (packet, sender);
1171          break;
1172        }
1173      case AODVTYPE_RREP_ACK:
1174        {
1175          RecvReplyAck (sender);
1176          break;
1177        }
1178      }
1179  }
```

RecvRequest() Function

- This function is used to receive RREQ packet.
- It ignores duplicate packets and packets from blacklisted nodes.
- Then it increments RREQ hop count and creates or updates reverse route.
- If the neighbor is not found in routing table, then it creates a new entry.
- Now if the node itself is the destination or has an active route to the destination, then a Reply packet(RREP) is generated and function is returned.
- It also checks if TTL is exceeded.
- And at last, it is broadcasted.

RecvRequest() Function

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

```
void
RoutingProtocol::RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
{
  NS_LOG_FUNCTION (this);
  RreqHeader rreqHeader;
  p->RemoveHeader (rreqHeader);

  // A node ignores all RREQs received from any node in its blacklist
  RoutingTableEntry toPrev;
  if (m_routingTable.LookupRoute (src, toPrev))
    {
      if (toPrev.IsUnidirectional ())
        {
          NS_LOG_DEBUG ("Ignoring RREQ from node in blacklist");
          return;
        }
    }

  uint32_t id = rreqHeader.GetId ();
  Ipv4Address origin = rreqHeader.GetOrigin ();

  /*
   *  Node checks to determine whether it has received a RREQ with the same Originator IP Address and RREQ ID.
   *  If such a RREQ has been received, the node silently discards the newly received RREQ.
   */
  if (m_rreqIdCache.IsDuplicate (origin, id))
    {
      NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
      return;
    }
```

RecvReply() Function

- It is used to receive RREP packet.
- If the reply is a hello message, the node should make sure that it has an active route and otherwise, it will create one using ProcessHello() function.
- The forward route for this destination is created if it does not already exist. Otherwise, the node compares the destination sequence no with its own stored destination sequence no and upon comparison, the routing table entry is updated.
- If current node is not the destination node, the RREP packet is forwarded to the node determined by its routing table entry.
- Then it is acknowledged by sending RREP-ACK packet back.
- It also checks if TTL is exceeded.

RecvReply() Function

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

```
void
RoutingProtocol::RecvReply (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
{
  NS_LOG_FUNCTION (this << " src " << sender);
  RrepHeader rrepHeader;
  p->RemoveHeader (rrepHeader);
  Ipv4Address dst = rrepHeader.GetDst ();
  NS_LOG_LOGIC ("RREP destination " << dst << " RREP origin " << rrepHeader.GetOrigin ());

  uint8_t hop = rrepHeader.GetHopCount () + 1;
  rrepHeader.SetHopCount (hop);

  // If RREP is Hello message
  if (dst == rrepHeader.GetOrigin ())
    {
      ProcessHello (rrepHeader, receiver);
      return;
    }
```

aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > ProcessHello(RrepHeader const &, Ipv4Address)

```
void
RoutingProtocol::ProcessHello (RrepHeader const & rrepHeader, Ipv4Address receiver )
```

RecvError() Function

- It is used to receive RERR packet.
- It is sent when a node breaks or a route error happens.
- It lists unreachable destinations consisting unreachable neighbor and destinations that use the unreachable neighbor as next hop.
- It creates the precursor list with affected neighbors.
- Then it again send RERR packets to the precursors.
- It then invalidates routes with unreachable destinations.

RecvError() Function

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

```
void
RoutingProtocol::RecvError (Ptr<Packet> p, Ipv4Address src )
{
  NS_LOG_FUNCTION (this << " from " << src);
  RerrHeader rerrHeader;
  p->RemoveHeader (rerrHeader);
  std::map<Ipv4Address, uint32_t> dstWithNextHopSrc;
  std::map<Ipv4Address, uint32_t> unreachable;
  m_routingTable.GetListOfDestinationWithNextHop (src, dstWithNextHopSrc);
  std::pair<Ipv4Address, uint32_t> un;
  while (rerrHeader.RemoveUnDestination(un))
  {
    for (std::map<Ipv4Address, uint32_t>::const_iterator i =
            dstWithNextHopSrc.begin();
         i != dstWithNextHopSrc.end(); ++i)
    {
      if (i->first == un.first)
      {
        unreachable.insert(un);
      }
    }
    }
```

RecvReplyAck() Function

- It is used for RREP_ACK packet.
- It just sets the Route Flags and update routing table.

RecvReplyAck() Function

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

```
void
RoutingProtocol::RecvReplyAck (Ipv4Address neighbor)
{
  NS_LOG_FUNCTION (this);
  RoutingTableEntry rt;
  if (m_routingTable.LookupRoute (neighbor, rt))
    {
      rt.m_ackTimer.Cancel ();
      rt.SetFlag (VALID);
      m_routingTable.Update (rt);
    }
}
```

FORWARDING

Two functions are mainly used while forwarding

- RouteInput()

- Forwarding()

https://www.nsnam.org/doxygen/classns3_1_1aodv_1_1_routing_protocol.html

FORWARDING - RouteInput()

◆ RouteInput()

```
bool ns3::aodv::RoutingProtocol::RouteInput ( Ptr< const Packet >          p,
                                              const Ipv4Header &           header,
                                              Ptr< const NetDevice >       idev,
                                              UnicastForwardCallback       ucb,
                                              MulticastForwardCallback     mcb,
                                              LocalDeliverCallback         lcb,
                                              ErrorCallback                ecb
                                            )
```

Routes an input packet - to be forwarded or locally delivered

FORWARDING - RouteInput()

Parameters

p	received packet
header	input parameter used to form a search key for a route
idev	Pointer to ingress network device
ucb	**Callback** for the case in which the packet is to be forwarded as unicast
mcb	**Callback** for the case in which the packet is to be forwarded as multicast
lcb	**Callback** for the case in which the packet is to be locally delivered
ecb	**Callback** to call if there is an error in forwarding

Returns

true if the **Ipv4RoutingProtocol** takes responsibility for forwarding or delivering the packet, false otherwise

FORWARDING - RouteInput()

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

```
bool
RoutingProtocol::RouteInput (Ptr<const Packet> p, const Ipv4Header &header,
                             Ptr<const NetDevice> idev, UnicastForwardCallback ucb,
                             MulticastForwardCallback mcb, LocalDeliverCallback lcb, ErrorCallback ecb)
{
  NS_LOG_FUNCTION (this << p->GetUid () << header.GetDestination () << idev->GetAddress ());
  if (m_socketAddresses.empty ())
    {
      NS_LOG_LOGIC ("No aodv interfaces");
      return false;
    }
  NS_ASSERT (m_ipv4 != 0);
  NS_ASSERT (p != 0);
  // Check if input device supports IP
  NS_ASSERT (m_ipv4->GetInterfaceForDevice (idev) >= 0);
  int32_t iif = m_ipv4->GetInterfaceForDevice (idev);
```

FORWARDING - RouteInput()

FORWARDING - Forwarding()

◆ Forwarding()

```
bool ns3::aodv::RoutingProtocol::Forwarding ( Ptr< const Packet >          p,
                                              const Ipv4Header &           header,
                                              UnicastForwardCallback       ucb,
                                              ErrorCallback                ecb
                                            )
```

If route exists and is valid, forward the packet

FORWARDING - Forwarding()

Parameters

p	the packet to route
header	the IP header
ucb	the UnicastForwardCallback function
ecb	the ErrorCallback function

Returns

true if forwarded

FORWARDING - Forwarding()

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

```
592  bool
593  RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
594                               UnicastForwardCallback ucb, ErrorCallback ecb)
595  {
596    NS_LOG_FUNCTION (this);
597    Ipv4Address dst = header.GetDestination ();
598    Ipv4Address origin = header.GetSource ();
599    m_routingTable.Purge ();
600    RoutingTableEntry toDst;
601    if (m_routingTable.LookupRoute (dst, toDst))
602      {
```

ROUTING TABLE - Classes

src > aodv > model > C aodv-rtable.h > {} ns3 > {} aodv > RoutingTable

```
389  class RoutingTable
390  {
391  public:
392    /**
393     * constructor
394     * \param t the routing table entry lifetime
395     */
396    RoutingTable (Time t);
397    ///\name Handle lifetime of invalid route
398    //\{
399    /**
400     * Get the lifetime of a bad link
401     *
402     * \return the lifetime of a bad link
```

src > aodv > model > C aodv-rtable.h > {} ns3 > {} aodv > RoutingTable >

```
59  class RoutingTableEntry
60  {
61  public:
62    /**
63     * constructor
64     *
65     * \param dev the device
66     * \param dst the destination IP address
67     * \param vSeqNo verify sequence number flag
68     * \param seqNo the sequence number
69     * \param iface the interface
70     * \param hops the number of hops
71     * \param nextHop the IP address of the next hop
72     * \param lifetime the lifetime of the entry
73     */
```

https://www.nsnam.org/doxygen/classns3_1_1aodv_1_1_routing_table.html

ROUTING TABLE - Basic Functions

src > aodv > model > C aodv-rtable.h > {} ns3 > {} aodv > RoutingTable

```
418     /**
419      * Add routing table entry if it doesn't yet exist in routing table
420      * \param r routing table entry
421      * \return true in success
422      */
423     bool AddRoute (RoutingTableEntry & r);
424     /**
425      * Delete routing table entry with destination address dst, if it exists.
426      * \param dst destination address
427      * \return true on success
428      */
429     bool DeleteRoute (Ipv4Address dst);
430     /**
431      * Lookup routing table entry with destination address dst
432      * \param dst destination address
433      * \param rt entry with destination address dst, if exists
434      * \return true on success
435      */
436     bool LookupRoute (Ipv4Address dst, RoutingTableEntry & rt);
437     /**
438      * Lookup route in VALID state
439      * \param dst destination address
440      * \param rt entry with destination address dst, if exists
441      * \return true on success
442      */
443     bool LookupValidRoute (Ipv4Address dst, RoutingTableEntry & rt);
444     /**
445      * Update routing table
446      * \param rt entry with destination address dst, if exists
447      * \return true on success
448      */
449     bool Update (RoutingTableEntry & rt);
```

ROUTING TABLE - Basic Functions

src > aodv > model > C aodv-rtable.h > {} ns3 > {} aodv > RoutingTable

```
/**
 * Delete all route from interface with address iface
 * \param iface the interface IP address
 */
void DeleteAllRoutesFromInterface (Ipv4InterfaceAddress iface);
/// Delete all entries from routing table
void Clear ()
{
  m_ipv4AddressEntry.clear ();
}
/// Delete all outdated entries and invalidate valid entry if Lifetime is expired
void Purge ();
```

ROUTING TABLE - Adding Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > SetIpv4(Ptr<Ipv4>)

```
648  void
649  RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
650  {
651    NS_ASSERT (ipv4 != 0);
652    NS_ASSERT (m_ipv4 == 0);
653
654    m_ipv4 = ipv4;
655
656    // Create lo route. It is asserted that the only one interface up
657    NS_ASSERT (m_ipv4->GetNInterfaces () == 1 && m_ipv4->GetAddress (0
658    m_lo = m_ipv4->GetNetDevice (0);
659    NS_ASSERT (m_lo != 0);
660    // Remember lo route
661    RoutingTableEntry rt (/*device=*/ m_lo, /*dst=*/ Ipv4Address::GetL
662                          /*iface=*/ Ipv4InterfaceAddress
663                          /*hops=*/ 1, /*next hop=*/ Ipv4A
664                          /*lifetime=*/ Simulator::GetMaxi
665    m_routingTable.AddRoute (rt);
666
667    Simulator::ScheduleNow (&RoutingProtocol::Start, this);
668  }
```

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > UpdateRouteToNeighbor(Ipv4Address, Ipv4Address)

```
1196 void
1197 RoutingProtocol::UpdateRouteToNeighbor (Ipv4Address sender, Ipv4Address receiver)
1198 {
1199   NS_LOG_FUNCTION (this << "sender " << sender << " receiver " << receiver);
1200   RoutingTableEntry toNeighbor;
1201   if (!m_routingTable.LookupRoute (sender, toNeighbor))
1202     {
1203       Ptr<NetDevice> dev = m_ipv4->GetNetDevice (m_ipv4->GetInterfaceForAddress (receive
1204       RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=*/ fals
1205                                   /*iface=*/ m_ipv4->GetAddress (m_ipv4->Get
1206                                   /*hops=*/ 1, /*next hop=*/ sender, /*lifet
1207       m_routingTable.AddRoute (newEntry);
```

ROUTING TABLE - Adding Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > SendRequest(Ipv4Address)

```
1028        if (ttl == m_netDiameter)
1029          {
1030            newEntry.IncrementRreqCnt ();
1031          }
1032        newEntry.SetFlag (IN_SEARCH);
1033        m_routingTable.AddRoute (newEntry);
1034      }
1035
```

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

```
1276        RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ origin, /*validSeno=*/ ti
1277                                    /*iface=*/ m_ipv4->GetAddress (m_ipv4-
1278                                    /*nextHop*/ src, /*timeLife=*/ Time ((
1279        m_routingTable.AddRoute (newEntry);
1280      }
1281    else
1282      {
1283        if (toOrigin.GetValidSeqNo ())
1284          {
```

ROUTING TABLE - Deleting Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > RouteRequestTimerExpire(Ipv4Address)

```
1774      */
1775     if (toDst.GetRreqCnt () == m_rreqRetries)
1776       {
1777         NS_LOG_LOGIC ("route discovery to " << dst << " has been attempted Rre
1778         m_addressReqTimer.erase (dst);
1779         m_routingTable.DeleteRoute (dst);
1780         NS_LOG_DEBUG ("Route not found. Drop all packets with dst " << dst);
1781         m_queue.DropPacketWithDst (dst);
1782         return;
```

ROUTING TABLE - Updating Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > SendRequest(Ipv4Address)

```
1013            rreqHeader.SetUnknownSeqno (true);
1014          }
1015        rt.SetHop (ttl);
1016        rt.SetFlag (IN_SEARCH);
1017        rt.SetLifeTime (m_pathDiscoveryTime);
1018        m_routingTable.Update (rt);
1019      }
```

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

```
1298        toOrigin.SetHop (hop);
1299        toOrigin.SetLifeTime (std::max (Time (2 * m_netTraversalTime - 2 * hop * m_node
1300                                        toOrigin.GetLifeTime ()));
1301        m_routingTable.Update (toOrigin);
1302        //m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
1303      }
1304
```

ROUTING TABLE - Updating Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > SendReplyByIntermediateNode(RoutingTableEntry &, RoutingTableEntry &, bool)

```
1457        toNextHop.m_ackTimer.SetDelay (m_nextHopWait);
1458      }
1459    toDst.InsertPrecursor (toOrigin.GetNextHop ());
1460    toOrigin.InsertPrecursor (toDst.GetNextHop ());
1461    m_routingTable.Update (toDst);
1462    m_routingTable.Update (toOrigin);
1463
```

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

```
1553        */
1554        if (!toDst.GetValidSeqNo ())
1555          {
1556            m_routingTable.Update (newEntry);
1557          }
1558        // (ii)the Destination Sequence Number in the RREP is greater than the node's c
1559        else if ((int32_t (rrepHeader.GetDstSeqno ()) - int32_t (toDst.GetSeqNo ())) >
1560          {
1561            m_routingTable.Update (newEntry);
1562          }
1563        else
1564          {
```

ROUTING TABLE - Updating Route

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > UpdateRouteLifeTime(Ipv4Address, Time)

```
1176
1177  bool
1178  RoutingProtocol::UpdateRouteLifeTime (Ipv4Address addr, Time lifetime)
1179  {
1180    NS_LOG_FUNCTION (this << addr << lifetime);
1181    RoutingTableEntry rt;
1182    if (m_routingTable.LookupRoute (addr, rt))
1183      {
1184        if (rt.GetFlag () == VALID)
1185          {
1186            NS_LOG_DEBUG ("Updating VALID route");
1187            rt.SetRreqCnt (0);
1188            rt.SetLifeTime (std::max (lifetime, rt.GetLifeTime ()));
1189            m_routingTable.Update (rt);
```

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > UpdateRouteToNeighbor(Ipv4Address, Ipv4Address)

```
1213          {
1214            toNeighbor.SetLifeTime (std::max (m_activeRouteTimeout, toNeighbor.GetLifeT
1215          }
1216        else
1217          {
1218            RoutingTableEntry newEntry (/*device=*/ dev, /*dst=*/ sender, /*know seqno=
1219                                        /*iface=*/ m_ipv4->GetAddress (m_ip
1220                                        /*hops=*/ 1, /*next hop=*/ sender,
1221            m_routingTable.Update (newEntry);
1222          }
1223      }
```

ROUTING TABLE - Purge

src > aodv > model > aodv-routing-protocol.cc > {} ns3 > {} aodv > Forwarding(Ptr<const Packet>, const Ipv4He

```
591  // TODO
592  bool
593  RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header & header,
594                               UnicastForwardCallback ucb, ErrorCallback ecb)
595  {
596    NS_LOG_FUNCTION (this);
597    Ipv4Address dst = header.GetDestination ();
598    Ipv4Address origin = header.GetSource ();
599    m_routingTable.Purge ();
600    RoutingTableEntry toDst;
601    if (m_routingTable.LookupRoute (dst, toDst))
602      {
603        if (toDst.GetFlag () == VALID)
604          {
```

Data flows from/to
TCP Layer and MAC Layer

Run the AODV

src > aodv > examples > aodv.cc > ...

```
void
AodvExample::Run ()
{
//  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", UintegerVal
  CreateNodes ();
  CreateDevices ();
  InstallInternetStack ();
  InstallApplications ();

  std::cout << "Starting simulation for " << totalTime << " s ...\n";

  Simulator::Stop (Seconds (totalTime));
  Simulator::Run ();
  Simulator::Destroy ();
}
```

Create Devices

src > aodv > examples > aodv.cc

```
void
AodvExample::CreateDevices ()
{
  WifiMacHelper wifiMac;
  wifiMac.SetType ("ns3::AdhocWifiMac");
  YansWifiPhyHelper wifiPhy;
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiPhy.SetChannel (wifiChannel.Create ());
  WifiHelper wifi;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode", StringVal
  );
  devices = wifi.Install (wifiPhy, wifiMac, nodes);

  if (pcap)
    {
      wifiPhy.EnablePcapAll (std::string ("aodv"));
    }
}
```

MAC Layer

build > ns3 > wifi-mac-helper.h

```
protected:
  ObjectFactory m_mac;                  ///< MAC object factory
  ObjectFactory m_protectionManager;    ///< Factory to create a protection manager
  ObjectFactory m_ackManager;           ///< Factory to create an acknowledgment manager
  ObjectFactory m_muScheduler;          ///< Multi-user Scheduler object factory
};
```

build > ns3 > wifi-mac-helper.h > {} ns3

```
class WifiMac;
class NetDevice;
```

Send Receive

build > ns3 > net-device.h > {} ns3 > NetDevice

```
248   virtual bool Send (Ptr<Packet> packet, const Address& dest, uint16_t protocolNumber) = 0;
249   /**
250    * \param packet packet sent from above down to Network Device
251    * \param source source mac address (so called "MAC spoofing")
252    * \param dest mac address of the destination (already resolved)
253    * \param protocolNumber identifies the type of payload contained in
254    *        this packet. Used to call the right L3Protocol when the packet
255    *        is received.
256    *
257    *  Called from higher layer to send packet into Network Device
258    *  with the specified source and destination Addresses.
259    *
260    * \return whether the Send operation succeeded
261    */
```

build > ns3 > net-device.h > {} ns3

```
35   class Node;
36   class Channel;
37
```

build > ns3 > node.h > {} ns3 > Node

```
259    * \returns true if the packet has been delivered to a protocol handler.
260    */
261   bool ReceiveFromDevice (Ptr<NetDevice> device, Ptr<const Packet>, uint16_t protocol,
262                           const Address &from, const Address &to, NetDevice::PacketType packetType, bool promisc);
263
264   /**
265    * \brief Finish node's construction by setting the correct node ID.
266    */
```

Routing

src > aodv > examples > aodv.cc

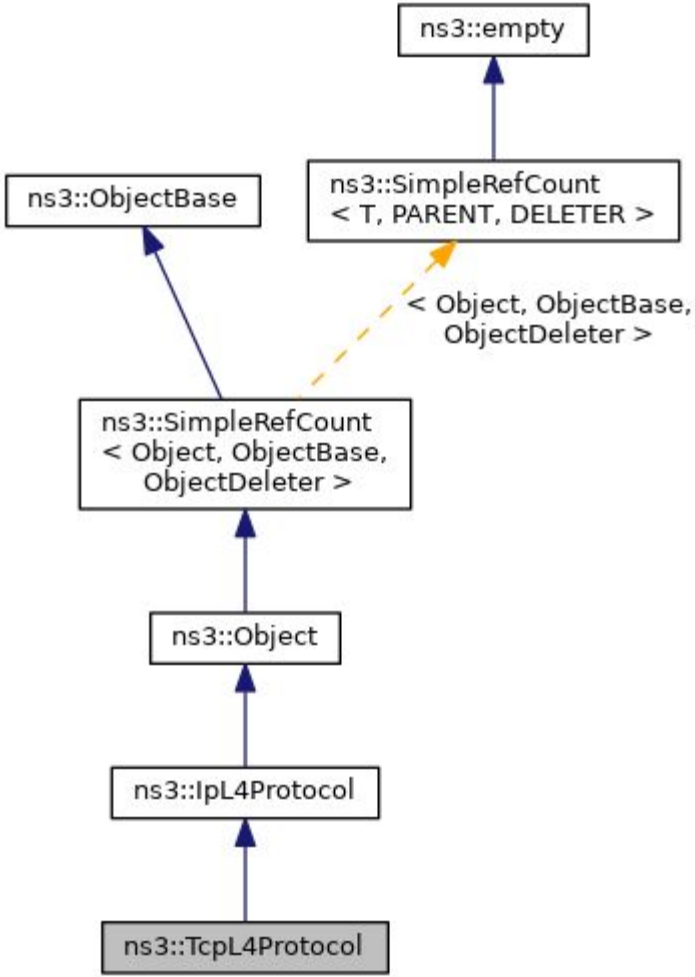
```
void
AodvExample::InstallInternetStack ()
{
  AodvHelper aodv;
  // you can configure AODV attributes here using aodv.Set(name, value)
  InternetStackHelper stack;
  stack.SetRoutingHelper (aodv); // has effect on the next Install ()
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);

  if (printRoutes)
    {
      Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("aodv.routes", std::ios::out);
      aodv.PrintRoutingTableAllAt (Seconds (8), routingStream);
    }
}
```

TCP Protocol

src > internet > helper > internet-stack-helper.cc

```
void
InternetStackHelper::Initialize ()
{
  SetTcp ("ns3::TcpL4Protocol");
  Ipv4StaticRoutingHelper staticRouting;
  Ipv4GlobalRoutingHelper globalRouting;
  Ipv4ListRoutingHelper listRouting;
  Ipv6StaticRoutingHelper staticRoutingv6;
  listRouting.Add (staticRouting, 0);
  listRouting.Add (globalRouting, -10);
  SetRoutingHelper (listRouting);
  SetRoutingHelper (staticRoutingv6);
}
```

TcpL4Protocol Class



TcpL4Protocol Class

	TcpL4Protocol ()
	TcpL4Protocol (const TcpL4Protocol &)=delete
virtual	~TcpL4Protocol ()
void	AddSocket (Ptr< TcpSocketBase > socket) Make a socket fully operational. More...
Ipv4EndPoint *	Allocate (Ipv4Address address) Allocate an IPv4 Endpoint. More...
Ipv4EndPoint *	Allocate (Ptr< NetDevice > boundNetDevice, Ipv4Address address, uint16_t port) Allocate an IPv4 Endpoint. More...
Ipv4EndPoint *	Allocate (Ptr< NetDevice > boundNetDevice, Ipv4Address localAddress, uint16_t localPort, Ipv4Address peerAddress, uint16_t peerPort) Allocate an IPv4 Endpoint. More...
Ipv4EndPoint *	Allocate (Ptr< NetDevice > boundNetDevice, uint16_t port) Allocate an IPv4 Endpoint. More...

Sending Packet

◆ SendPacketV4()

```
void ns3::TcpL4Protocol::SendPacketV4 ( Ptr< Packet >          pkt,
                                        const TcpHeader &      outgoing,
                                        const Ipv4Address &    saddr,
                                        const Ipv4Address &    daddr,
                                        Ptr< NetDevice >       oif = 0
                                      )                        const
```

Send a packet via TCP (IPv4)

Receiving Packet

◆ Receive() [1/2]

```
enum IpL4Protocol::RxStatus ns3::TcpL4Protocol::Receive ( Ptr< Packet >          p,
                                                          Ipv4Header const &     header,
                                                          Ptr< Ipv4Interface >   incomingInterface
                                                        )
```

Thank You