7
Referências Bibliográficas

[1] Andersson, L., Feldman, N., Fredette, A., Thomas, B. **LDP Specification**. RFC 3036, 2008

[2] Arkko, J., Vogt, C., Haddad, W. **Enhanced Route Optimization for Mobile IPv6**. RFC 4866, 2007

[3] Awduche, D., Berger, L., Li, T., Srinivasan, V., Swallow, G. **RSVP-TE: Extensions to RSVP for LSP Tunnels**. RFC 3209, 2001

[4] Fall, K., Varadhan, K. **The ns Manual (formerly ns Notes and Documentation)**. The VINT Project, 2007

[5] Henderson, T. **End-Host Mobility and Multihoming with the Host Identity Protocol**. draft-ietf-hip-mm-05.txt, 2007

[6] **IPv6SuiteWithINET**. Disponível em <http://ctieware.eng.monash.edu.au/twiki/bin/view/Simulation/IPv6Suite>. Acesso em: dez/2007

[7] **INET Framework**. Disponível em <http://www.omnetpp.org/doc/INET/neddoc/index.html> Acesso em: mar/2008

[8] Janeiro, J., Roenick, M., Silva, A., Colcher, S. **Comparação entre os simuladores NS-2 e OMNeT++ - Relatório Técnico no. XXXX**. Departamento de Informática PUC-Rio, 2007

[9] Johnson, D., Deering, S. **Reserved IPv6 Subnet Anycast Addresses.** RFC 2526, 1999

[10] Johnson, D., Perkins, C., Arkko, J. **Mobility Support in IPv6**. RFC 3755, 2004

[11] Jokela, P., Moskowitz, R., Nikander, P. **Using ESP transport format with HIP**. draft-ietfu-hip-esp-05.txt, 2007

[12] Koodli, R. **Fast Handovers for Mobile IPv6**. RFC 4068, 2005

[13] Laganier, J., Eggert, L. **Host Identity Protocol (HIP) Rendezvous Extension**. draft-ietf-hip-rvs-05.txt, 2006

[14] Laganier, J., Koponen, T., Eggert, L. **Host Identity Protocol (HIP) Registration Extension**. draft-ietf-hip-registration-02.txt, 2006

[15] **Mobility Framework**. Disponível em <http://mobility-fw.sourceforge.net/manual/index.html>. Acesso em: jul/2008

[16] Mockapetris, P. **Domain names - concepts and facilities, STD 13**. RFC 1034, 1987

[17] Moskowitz, R., Nikander, P. **Host Identity Protocol (HIP) Architecture**. RFC 4423, 2006

[18] Moskowitz, R., Nikander, P., Jokela, P., Henderson, T. **Host Identity Protocol**. draft-ietf-hip-base-07.txt, 2007

[19] Narten, T., Nordmark, E., Simpson, W. **Neighbor Discovery for IP Version 6 (IPv6)**. IETF, 1998

[20] Nikander, P. **Host Identity Protocol (HIP) Domain Name System (DNS) Extensions**. draft-ietf-hip-dns-09.txt, 2007

[21] **OMNeT++ - Discrete Event Simulation System**. Disponível em <http://www.omnetpp.org/doc/INET/neddoc/> Acessado em: dez/2007

[22] Rescorla, E. **Diffie-Hellman Key Agreement Method**. RFC 2631, 1999

[23] Rosen, E., Viswanathan, A., Callon, R. **Multiprotocol Label Switching Architecture**. RFC 3021, 2001

[24] Soliman, H., Castelluccia, C., El Malki, K., Bellier, L. **Hierarchical Mobile IPv6 Mobility Management (HMIPv6).** RFC 4140, 2005

[25] Wilterdink, R.J.W. **Host Identity Protocol: A state of the art research**. 4th Twente Student Conference on IT, 2006

8
Apêndice A – Códigos fontes

8.1.
Arquivo HIP.cc

```
#include <omnetpp.h>
#include <vector>
#include "HIP.h"
#include "IPv6Datagram.h"
#include "IPv6ControlInfo.h"
#include "IPAddressResolver.h"
#include "HIP_m.h"
#include "NotifierConsts.h"

#define HIP_REGISTER           0
#define HIP_TIMEOUT      1
#define ACK_DELAYED      0.2

Define_Module(HIP);

HIP::HIP()
{
    lastSPI = 1024;
    handover_num = 0;
}

HIP::~HIP()
{
    commHip.clear();
}

void HIP::initialize(int stage=1)
{
   //std::cout <<"HIP Parameters initialized" << endl;
   nb = NotificationBoardAccess().get();
   //for L3 Changes
   nb->subscribe(this, NF_IPv6_INTERFACECONFIG_CHANGED);
   nb->subscribe(this, NF_L2_BEACON_LOST);

   //parameters
   handoffVector.setName("Handover");
```

```
   delay = par("procDelay");
   myHip = par("hipAddress");
   myRVS = par("rvsHITAddress");
   startHandoff, endHandoff = 0;
   timeout=NULL;

   // create a timer for register this node at Rendezvous
Server
   timerRegister = new cMessage("timerRegister",
HIP_REGISTER);
   scheduleAt(uniform(5,8), timerRegister);

   //insert all addresses at HIPCache
   insertAllHIT();
}

void HIP::handleMessage(cMessage *msg)
{
    if (msg->isSelfMessage())
    {
 //std::cout << "Receiving self-message" << endl;
 if (msg->kind()==HIP_TIMEOUT) {
    //std::cout << "Size of reSendHIPList: " <<
reSendHIPList.size() << endl;
    if (findRetransmissionEntry((HIPComm*) msg-
>contextPointer())!=NULL) {
       SendAgain* temp = findRetransmissionEntry((HIPComm*)
msg->contextPointer());
       //std::cout << "Temp = " << temp << endl;
       HIPComm* comm = (HIPComm*) msg->contextPointer();
       //std::cout << "Comm = " << comm << endl;
       int estado = comm->state();
       //std::cout << "Comm State = " << estado << endl;
       int type = temp->typeMsg;
       //std::cout << "Type of message at temp = " << type <<
endl;
       cMessage* pointer = temp->Mensagem;
       //std::cout << "Pointer of Msg at temp = " << pointer
<< endl;
       cMessage* copy = (cMessage *) pointer->dup();
       //std::cout << "Pointer of copy = " << copy << endl;
       IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
       if (copy!=NULL) {
             if ((IPv6ControlInfo *)copy-
>removeControlInfo()!=NULL) {
                  IPv6ControlInfo* teste = (IPv6ControlInfo
*)copy->removeControlInfo();
```

```
                    delete teste;
                }
                      controlInfo-
>setSrcAddr(findIPv6(myHip));
                    controlInfo->setDestAddr(findIPv6(comm-
>remoteHIT()));
                    controlInfo->setProtocol(253);
                copy->setControlInfo(controlInfo);
        }
        //Test if message was received and status changed
before resend msg
        if (estado==1 && type == 1 && copy!=NULL) {
              //std::cout << "Re-sending I1 Message" << endl;
              send(copy,"hipIPOut");
              scheduleAt(simTime()+ACK_DELAYED, msg);
        } else if ((estado==2 && type == 2 &&
copy!=NULL)||(estado==2 && type == 5 && copy!=NULL)) {
              //std::cout << "Re-sending I2 / I2_Rendezvous
Message" << endl;
              send(copy,"hipIPOut");
              scheduleAt(simTime()+ACK_DELAYED, msg);
        } else if ((estado==3 && type == 3 &&
copy!=NULL)||(estado==3 && type == 6 && copy!=NULL)) {
              //std::cout << "Re-sending R1 / R1_Rendezvous
Message" << endl;
              send(copy,"hipIPOut");
              scheduleAt(simTime()+ACK_DELAYED, msg);
        } else if ((estado==5 && type == 4 &&
copy!=NULL)||(estado==5 && type == 7 && copy!=NULL)) {
              //std::cout << "Re-sending R2 / R2_Rendezvous
Message" << endl;
              send(copy,"hipIPOut");
              scheduleAt(simTime()+ACK_DELAYED, msg);
        } else if (estado==5 && type == 14 && copy!=NULL &&
comm->addressStatus()==0) {
              //std::cout << "Re-sending Update1 Message" <<
endl;
              send(copy,"hipIPOut");
              scheduleAt(simTime()+ACK_DELAYED, msg);
        } else {
          //std::cout << "Descarting timeout message... State
of Comm already changed." << endl;
          delete copy;
          delete temp->Mensagem;
          reSendHIPList.erase(temp);
                delete temp;
        }
```

```
    } else {
       IPv6Address teste = findHIT("0:0:0:0:0:0:0:0");
       if (teste!="0:0:0:0:0:0:0:0") askRVSIPv6(teste);
    }
 } else if (msg->kind()==HIP_REGISTER)
 {
    if (findCommHIT(myHip, myRVS)==NULL && findIPv6(myHip) !=
"0:0:0:0:0:0:0:0")
    {  RVSRegister();
    } else { scheduleAt(simTime()+uniform(0,0.1), msg); }
 }
    } else {
 if (msg->arrivalGate()->isName("hipTcpIn")) {
             // packet from upper layers or ND: encapsulate
and send out
       //std::cout <<"Mensagem entregue para HIP - TCP para
IPv6" << endl;
             processTranspData(msg);
       } else if (msg->arrivalGate()->isName("hipUdpIn")) {
             // packet from upper layers or ND: encapsulate
and send out
       //std::cout <<"Mensagem entregue para HIP - UDP para
IPv6"<< endl;
             processTranspData(msg);
       } else if (msg->arrivalGate()->isName("hipIPTcpIn")) {
       // datagram from network or from ND: localDeliver
and/or route
       //std::cout <<"Mensagem entregue para HIP - IPv6 para
TCP"<< endl;
             processIPv6Datagram(msg);
       } else if (msg->arrivalGate()->isName("hipIPUdpIn")) {
       // datagram from network or from ND: localDeliver
and/or route
       //std::cout <<"Mensagem entregue para HIP - IPv6 para
UDP"<< endl;
             processIPv6Datagram(msg);
       } else if (msg->arrivalGate()->isName("hipIPIn")) {
       // datagram from network or from ND: localDeliver
and/or route
       //std::cout <<"Mensagem HIP vindo da camada IPv6 para
HIP. /n"<< endl;
             processIPv6Datagram(msg);
       }
    }
}

//Lookup of timeout entry
```

```
HIP::SendAgain* HIP::findRetransmissionEntry(HIPComm *comm)
{
   for (retransmissionList::iterator
it=reSendHIPList.begin(); it!=reSendHIPList.end(); it++)
   {
       SendAgain *entry = (*it);
       if (entry->communication == comm)
       {
           return entry;
    break;
       }
   }
   return NULL;
}

void HIP::printCommBrief(HIPComm comm)
{
    std::cout <<"communication (HIT Address / IPv6 Address)
";
    std::cout <<comm.localHIT() << "/" << comm.localAddr() <<
" to " << comm.remoteHIT() << "/" << comm.remoteAddr();
    std::cout <<",  on SPI " << comm.SPI();
    std::cout <<",  status " << comm.state();
    std::cout <<",  addressStatus " << comm.addressStatus();
    if (comm.state() == 5) { std::cout <<", Entry LifeTime "
<< comm.expireTime(); }
    std::cout <<"." <<endl;
}

// control of communication timeout - not implemented
void HIP::processExpireTime(HIPComm *comm)
{
    //std::cout <<comm << ": HIPCommunication expired" <<
endl;
    comm->setState(8);
    comm->setAddressStatus(2);
}

// process packets of Higher Layers
void HIP::processTranspData(cMessage *msg)
{
    //std::cout <<"Calling processTranspData()." << endl;
    IPv6ControlInfo *controlInfo = (IPv6ControlInfo *)msg-
>removeControlInfo();
    // Find important informations of packet received
    srcHIT = controlInfo->srcAddr();
    destHIT = controlInfo->destAddr();
```

```
    int protocolo = controlInfo->protocol();
    // usually, srcAddress is wrinting by IPv6 Layer.
Implementing a substitute...
    if (srcHIT=="0:0:0:0:0:0:0:0") srcHIT=myHip;
    //std::cout <<"processTranspData(): Source Address: "<<
srcHIT <<" and Destination Address: "<< destHIT << endl;

    //Find correspondents IPv6 Addresses of HIT Addresses
    srcIPv6 = findIPv6(srcHIT);
    destIPv6 = findIPv6(destHIT);
    //Initialize local parameters
    HIPComm* entry = NULL;
    bool entryFound = false;
    //Lookup comm at commHip Vector
    //std::cout <<"commHip Size: " << commHip.size() << endl;
    for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
        if (it->remoteHIT()==destHIT && it-
>localHIT()==srcHIT)
        {
          entryFound = true;
          destIPv6 = it->remoteAddr();
          srcIPv6 = it->localAddr();
          entry = &(*it);
   //std::cout <<"processTranspData(): HIP Communication
found: " << endl;
   //printCommBrief(*it);
          break;
        }
    };
    //Testing srcIPv6 and destIPv6 before initiating
communication HIP
    if (entryFound==false && ((srcIPv6 == "0:0:0:0:0:0:0:0")
|| (destIPv6 == "0:0:0:0:0:0:0:0")) ) {
 ////std::cout <<"An unspecified IPv6Address was found.
Waiting response of Rendezvous Server..." << endl;
 delete controlInfo;
 addHIPBuffer(srcHIT, destHIT, srcIPv6, destIPv6, protocolo,
msg);
    } else if (entryFound==false && ((srcIPv6 !=
"0:0:0:0:0:0:0:0") && (destIPv6 != "0:0:0:0:0:0:0:0"))) {
        //Test communication (existence and state) and type
of packet
 //std::cout <<"Communication HIP between src and dest not
found. Start a new one..." << endl;
        delete controlInfo;
```

```
        addHIPBuffer(srcHIT, destHIT, srcIPv6, destIPv6,
protocolo, msg);
 //std::cout <<"Sending a HIP I1 message..." << endl;
        sendI1(srcHIT, destHIT, srcIPv6, destIPv6);
    } else if (entryFound==true) {
 //std::cout <<"Communication HIP between src and dest found.
Checking state..." << endl;
 int estado = entry->state();
 //std::cout <<"State of communication HIP: " << estado <<
endl;
 if (estado==5)  //"HIP_S_ESTABLISHED"
        {
       SendAgain* teste = findRetransmissionEntry(entry);
       if (teste!=NULL && teste->timeoutHIP!=NULL) {
             cancelEvent(teste->timeoutHIP);
             reSendHIPList.erase(teste);
       }
       controlInfo->setSrcAddr(srcIPv6);
             controlInfo->setDestAddr(destIPv6);
       msg->setControlInfo(controlInfo);
       if (protocolo == 6) { send(msg, "hipIPTcpOut"); }
       else if (protocolo == 17) { send(msg, "hipIPUdpOut");
}
 } else {
       delete controlInfo;
       addHIPBuffer(srcHIT, destHIT, srcIPv6, destIPv6,
protocolo, msg);
 }
    }
}

void HIP::processIPv6Datagram(cMessage *datagram)
{
    //std::cout <<"Calling processIPv6Datagram()." << endl;
    // Find important informations of datagrtam received
    IPv6ControlInfo *controlInfo = (IPv6ControlInfo
*)datagram->removeControlInfo();
    srcIPv6 = controlInfo->srcAddr();
    destIPv6 = controlInfo->destAddr();
    int protocolo = controlInfo->protocol();
    //std::cout <<"processIPv6Datagram(): Source Address: "<<
srcIPv6 <<" and Destination Address: "<< destIPv6 << endl;
    //Find correspondents HIT Addresses of IPv6 Addresses
    srcHIT = findHIT(srcIPv6);
    destHIT = findHIT(destIPv6);
    //Lookup comm at commHip Vector
    HIPComm* comm = findCommIPv6(srcIPv6, destIPv6);
```

```
    //Processing Datagram
    if (comm) {
 //std::cout <<"processIPv6Datagram(): HIP Communication
found: " << endl;
 //printCommBrief(*comm);
    }

    //Identification of type of datagram
    if (protocolo == IP_PROT_IPv6_HIP)
    {
       HIPHeader *temp = check_and_cast<HIPHeader
*>(datagram);

 //To replace use of SPI at HIPCommunication - Update1
Message
 HIPComm* testUpdate = findCommHIT(temp->getHit_sender(),
temp->getHit_receiver());
 if (testUpdate && temp->getPacket_type()==14) {
       srcHIT=temp->getHit_sender();
       destHIT=temp->getHit_receiver();
       testUpdate->setLocalAddr(srcIPv6);
       testUpdate->setRemoteAddr(destIPv6);
       comm = &(*testUpdate);
       HIPComm* test2=findCommHIT(destHIT, srcHIT);
       test2->setLocalAddr(destIPv6);
       test2->setRemoteAddr(srcIPv6);
 }

       switch (temp->getPacket_type())
       {
          case(1): //HIP I1 Message
          {
       //std::cout <<"Receiving HIP I1 message" << endl;
             if (!comm && srcHIT=="0:0:0:0:0:0:0:0") {
             addHIPCache(srcIPv6,temp->getHit_sender());
             srcHIT = temp->getHit_sender();
             addCommHIP(srcHIT, destHIT, srcIPv6, destIPv6,
0);
             addCommHIP(destHIT, srcHIT, destIPv6, srcIPv6,
0);
                   comm = findCommHIT(srcHIT, destHIT);
       } else if (!comm && destHIT=="0:0:0:0:0:0:0:0") {
             addHIPCache(destIPv6,temp->getHit_receiver());
             destHIT = temp->getHit_receiver();
             addCommHIP(srcHIT, destHIT, srcIPv6, destIPv6,
0);
```

```
            addCommHIP(destHIT, srcHIT, destIPv6, srcIPv6,
0);
                  comm = findCommHIT(srcHIT, destHIT);

       } else if ((comm->state()==8 || comm->state()==0)) {
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setState(2);
            comm2->setState(2);
            sendR1(destHIT, srcHIT, destIPv6, srcIPv6);
            } else {
            //std::cout <<"Receiving a HIP I1 message but
state of communication is diferent of 8 and 0. Dropping
datagram..." << endl;
       }
       delete datagram;
       break;
   }
   case(3): //HIP I2 Message
   {
       if (comm && comm->state()==2)
       {
                        //std::cout <<"Receiving HIP_I2
message" << endl;
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setState(5);
            comm2->setState(5);
            SendAgain* teste =
findRetransmissionEntry(comm2);
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
            sendR2(destHIT, srcHIT, destIPv6, srcIPv6);
       } else if (comm && comm->state()==5) { sendR2(destHIT,
srcHIT, destIPv6, srcIPv6); }
            delete datagram;
       break;
   }
   case(2): //HIP R1 Message
   {
       if (comm && comm->state()==1)
       {
                  //std::cout <<"Receiving HIP_R1 message"
<< endl;
```

```
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setState(3);
            comm2->setState(3);
            SendAgain* teste =
findRetransmissionEntry(comm2);
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
            sendI2(destHIT, srcHIT, destIPv6, srcIPv6);

      } else //std::cout <<"Receiving a HIP R1 message but
state of communication is diferent of 1. Dropping
datagram..." << endl;
            delete datagram;
      break;
         }
   case(4): //HIP R2 Message
   {
      if (comm && comm->state()==3)
      {
            //std::cout <<"Receiving HIP_R2 message" <<
endl;
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setState(5);
            comm2->setState(5);
            SendAgain* teste =
findRetransmissionEntry(comm2);
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
                  printCommBrief(*comm);
                  sendHIPBuffer(destIPv6, srcIPv6);
      } else //std::cout <<"Receiving a HIP R2 message but
state of communication is diferent of 3. Dropping
datagram..." << endl;
            delete datagram;
      break;
  }
  case(14): //HIP Update1 Message
  {
      if (comm && comm->state()==5)
      {
            std::cout <<"HIP Update1 Message" << endl;
```

```
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setAddressStatus(0);
            comm2->setAddressStatus(0);
            HIPUpdate1 *data = check_and_cast<HIPUpdate1
*>(datagram);
            HIPUpdate2 *msg = new HIPUpdate2();
            msg->setHit_sender(destHIT);
            msg->setHit_receiver(srcHIT);
            msg->setPacket_type(15);
            msg->setSpi(comm->SPI());
            msg->setSeq(data->getSeq());
            msg->setAck(data->getSeq()+1);
            msg->setHmac("MyMACAddress");
            msg->setHip_sig("000111");
            msg->setEcho_req("simulator");
            controlInfo->setSrcAddr(destIPv6);
            controlInfo->setDestAddr(srcIPv6);
                        controlInfo-
>setProtocol(IP_PROT_IPv6_HIP);
                  msg->setControlInfo(controlInfo);
                        std::cout <<"Sending HIP Update2
Message" << endl;
            send(msg,"hipIPOut");
            updateHIPCache(data->getLocator(), srcHIT);
      } else std::cout <<"Receiving a HIP Update1 message
but state of communication is diferent of 5. Dropping
datagram..." << endl;
            delete datagram;
      break;
   }
   case(15): //HIP Update2 Message
   {
      if (comm && (comm->state()==5 && comm-
>addressStatus()==0))
      {
            std::cout <<"HIP Update2 Message" << endl;
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setAddressStatus(1);
            comm2->setAddressStatus(1);
            SendAgain* teste =
findRetransmissionEntry(comm);
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
```

```
                    HIPUpdate2 *data =
check_and_cast<HIPUpdate2 *>(datagram);
              HIPUpdateAck *msg = new HIPUpdateAck();
              msg->setHit_sender(destHIT);
              msg->setHit_receiver(srcHIT);
              msg->setPacket_type(16);
              msg->setAck(data->getAck());
              msg->setHmac("MyMACAddress");
              msg->setHip_sig("000111");
              msg->setEcho_resp("omnetpp");
              controlInfo->setSrcAddr(destIPv6);
              controlInfo->setDestAddr(srcIPv6);
                        controlInfo-
>setProtocol(IP_PROT_IPv6_HIP);
                    msg->setControlInfo(controlInfo);
                        std::cout <<"Sending a HIP UpdateAck
Message" << endl;
       } else std::cout <<"Receiving a HIP Update2 message
but state of communication is diferent of 5. Dropping
datagram..." << endl;
              delete datagram;
       break;
          }
   case(16): //HIP Update Ack Message
   {
       if (comm && (comm->state()==5 && comm-
>addressStatus()==0))
       {
              std::cout <<"HIP Update Ack Message" << endl;
              HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
              comm->setAddressStatus(1);
              comm2->setAddressStatus(1);
              comm->setExpireTime(simTime()+120);
              comm2->setExpireTime(simTime()+120);
       } else std::cout <<"Receiving a HIP UpdateAck message
but state of communication is diferent of 5. Dropping
datagram..." << endl;
              delete datagram;
       break;
   }
   case(5): //HIP_R1_Rendezvous Message
   {
       if (comm && comm->state()==1)
       {
              HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
```

```
            comm->setState(3);
            comm2->setState(3);
            SendAgain* teste =
findRetransmissionEntry(comm2);
            //std::cout << " Pointer of teste: " << teste <<
endl;
            //std::cout << " Timeout msg at teste: " <<
teste->timeoutHIP << endl;
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
            //std::cout <<"Receiving HIP_R1_Rendezvous
message" << endl;
            sendI2Rendezvous(destHIT, srcHIT, destIPv6,
srcIPv6);
      } else //std::cout <<"Receiving a HIP R1_RVS message
but state of communication is diferent of 1. Dropping
datagram..." << endl;
            delete datagram;
      break;
   }
   case(7): //HIP_R2_Rendezvous Message
   {
      if (comm && comm->state()==3)
      {
            //std::cout <<"Receiving HIP_R2_Rendezvous
message" << endl;
            HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
            comm->setState(5);
            comm2->setState(5);
            SendAgain* teste =
findRetransmissionEntry(comm2);
            //std::cout << " Pointer of teste: " << teste <<
endl;
            //std::cout << " Timeout msg at teste: " <<
teste->timeoutHIP << endl;
            if (teste!=NULL && teste->timeoutHIP!=NULL) {
                  cancelEvent(teste->timeoutHIP);
                  reSendHIPList.erase(teste);
            }
                  printCommBrief(*comm);
      } else //std::cout <<"Receiving a HIP R2_RVS message
but state of communication is diferent of 3. Dropping
datagram..." << endl;
            delete datagram;
```

```
        break;
    }
    case(27): //RespRVSIPv6 message
    {
        //std::cout <<"Receiving RespRVSIPv6 message" << endl;
        HIPResp *data = check_and_cast<HIPResp *>(datagram);
        addHIPCache(data->getAnswer(), data->getTarget());
        srcIPv6 = findIPv6(myHip);
        destIPv6 = data->getAnswer();
        sendHIPBuffer(srcIPv6, destIPv6);
              delete datagram;
        break;
    }
    case(28): //RespRVSHip message
    {
        //std::cout <<"Receiving RespRVSHip message" << endl;
        HIPResp *data = check_and_cast<HIPResp *>(datagram);
        addHIPCache(data->getTarget(), data->getAnswer());
        srcIPv6 = findIPv6(myHip);
        destIPv6 = data->getTarget();
        sendHIPBuffer(srcIPv6, destIPv6);
              delete datagram;
        break;
    }
        };
         //If datagram is TCP or UDP Datagram
     } else if (protocolo == 6 || protocolo == 17)
     {
  if (comm && (comm->state()==5))
  {
     srcHIT = findHIT(srcIPv6);
     destHIT = findHIT(destIPv6);
     controlInfo->setSrcAddr(srcHIT);
     controlInfo->setDestAddr(destHIT);
     if (protocolo==6) {
              controlInfo->setProtocol(6);
        datagram->setControlInfo(controlInfo);
        send(datagram,"hipTcpOut");
            } else if (protocolo==17) {
        controlInfo->setProtocol(17);
        datagram->setControlInfo(controlInfo);
        send(datagram,"hipUdpOut");
            }
   } else {
                //std::cout <<"TCP or UDP Datagram received
but state of HIP Communication is diferent of 5. Dropping
datagram..." << endl;
```

```
                delete datagram;
            }
        }
    }

void HIP::sendI1(IPv6Address srcHIT, IPv6Address destHIT,
IPv6Address srcAddr, IPv6Address destAddr)
{
   //If that isn't a HIP Communication between this nodes,
add a new one
   HIPComm* comm = findCommHIT(srcHIT,destHIT);
   //std::cout << "Pointer of Comm: " << comm << endl;
   if (!comm)
   {
        addCommHIP(srcHIT, destHIT, srcAddr, destAddr, 1);
 addCommHIP(destHIT, srcHIT, destAddr, srcAddr, 1);
       comm = findCommHIT(srcHIT,destHIT);
       //std::cout << "Pointer of Comm: " << comm << endl;
   }
   //Preparing message for initialize HIP Communication
   HIPHeader *msg = new HIPHeader();
   msg->setHit_sender(srcHIT);
   msg->setHit_receiver(destHIT);
   msg->setPacket_type(1);
   IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
   controlInfo->setSrcAddr(srcAddr);
   controlInfo->setDestAddr(destAddr);
   controlInfo->setProtocol(IP_PROT_IPv6_HIP);
   msg->setControlInfo(controlInfo);
   //std::cout <<"Sending HIP I1 message" << endl;
   //std::cout <<"HIP I1 source and destination addresses: "
<< controlInfo->srcAddr() << ", " << controlInfo->destAddr()
<< endl;
   send(msg,"hipIPOut");
   timeout = new cMessage("I1Timeout", HIP_TIMEOUT);
   timeout->setContextPointer(comm);
   scheduleAt(simTime()+ACK_DELAYED, timeout);
   SendAgain *entry = new SendAgain();
   entry->communication = comm;
   entry->typeMsg = 1;
   entry->timeoutHIP= timeout;
   entry->Mensagem= (cMessage *) msg->dup();
   reSendHIPList.insert(entry);
}

void HIP::sendI2(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr)
```

```
{
 HIP_I2 *msg = new HIP_I2();
 msg->setHit_sender(destHIT);
 msg->setHit_receiver(srcHIT);
 msg->setPacket_type(3);
 msg->setSolution("HIP");
 msg->setDiffieHellman("group 1");
 msg->setHip_transf("AES with Hmac_SHA-1-96");
 msg->setHostId(destHIT);
 msg->setHmac("MyMACAddress");
 msg->setHip_sig("000111");
        IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
 controlInfo->setSrcAddr(destIPv6);
 controlInfo->setDestAddr(srcIPv6);
        controlInfo->setProtocol(IP_PROT_IPv6_HIP);
        msg->setControlInfo(controlInfo);
        HIPComm* comm2 = findCommHIT(destHIT,srcHIT);
       //std::cout << "Pointer of Comm: " << comm2 << endl;
       timeout = new cMessage("I2Timeout", HIP_TIMEOUT);
       timeout->setContextPointer(comm2);
       SendAgain *entry = new SendAgain();
       entry->communication = comm2;
 entry->timeoutHIP= timeout;
 entry->typeMsg= 3;
       entry->Mensagem= (cMessage *) msg->dup();
       reSendHIPList.insert(entry);
 //std::cout <<"Sending HIP_I2 message" << endl;
 send(msg,"hipIPOut");
}

void HIP::sendI2Rendezvous(IPv6Address destHIT, IPv6Address
srcHIT, IPv6Address destAddr, IPv6Address srcAddr)
{
 HIP_I2_Rendezvous *msg = new HIP_I2_Rendezvous();
 msg->setHit_sender(destHIT);
 msg->setHit_receiver(srcHIT);
 msg->setPacket_type(6);
 msg->setSolution("HIP");
 msg->setDiffieHellman("group 1");
 msg->setHip_transf("AES with Hmac_SHA-1-96");
 msg->setHostId(destHIT);
 msg->setHmac("MyMACAddress");
 msg->setHip_sig("000111");
 msg->setService("Register");
 msg->setValidade((short)simTime()+120);
        IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
 controlInfo->setSrcAddr(destIPv6);
```

```
 controlInfo->setDestAddr(srcIPv6);
        controlInfo->setProtocol(IP_PROT_IPv6_HIP);
        msg->setControlInfo(controlInfo);
        HIPComm* comm2 = findCommHIT(destHIT,srcHIT);
       //std::cout << "Pointer of Comm: " << comm2 << endl;
       timeout = new cMessage("I2RVSTimeout", HIP_TIMEOUT);
       timeout->setContextPointer(comm2);
       scheduleAt(simTime()+ACK_DELAYED, timeout);
       SendAgain *entry = new SendAgain();
       entry->communication = comm2;
 entry->timeoutHIP= timeout;
       entry->Mensagem= (cMessage *) msg->dup();
 entry->typeMsg= 6;
       reSendHIPList.insert(entry);
 //std::cout <<"Sending HIP_I2_Rendezvous Message" << endl;
 send(msg,"hipIPOut");
}

void HIP::sendR1(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr)
{
 HIP_R1 *msg = new HIP_R1();
 msg->setHit_sender(destHIT);
 msg->setHit_receiver(srcHIT);
 msg->setPacket_type(2);
 msg->setPuzzle("New Protocol");
 msg->setDiffieHellman("group 1");
 msg->setHip_transf("AES with Hmac_SHA-1-96");
 msg->setHostId(destHIT);
 msg->setHip_sig2("000000");
        IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
 controlInfo->setSrcAddr(destIPv6);
 controlInfo->setDestAddr(srcIPv6);
        controlInfo->setProtocol(IP_PROT_IPv6_HIP);
        msg->setControlInfo(controlInfo);
 //std::cout <<"Sending HIP_R1 message" << endl;
 send(msg,"hipIPOut");
        HIPComm* comm2 = findCommHIT(destHIT,srcHIT);
       timeout = new cMessage("R1Timeout", HIP_TIMEOUT);
       timeout->setContextPointer(comm2);
       SendAgain *entry = new SendAgain();
       entry->communication = findCommHIT(destHIT,srcHIT);
       entry->Mensagem= (cMessage *) msg->dup();
 entry->timeoutHIP= timeout;
 entry->typeMsg= 2;
       reSendHIPList.insert(entry);
}
```

```
void HIP::sendR2(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr)
{
 HIP_R2 *msg = new HIP_R2();
 msg->setHit_sender(destHIT);
 msg->setHit_receiver(srcHIT);
 msg->setPacket_type(4);
 msg->setHmac2("MyMACAddress");
 msg->setHip_sig("000111");
        IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
 controlInfo->setSrcAddr(destIPv6);
 controlInfo->setDestAddr(srcIPv6);
        controlInfo->setProtocol(IP_PROT_IPv6_HIP);
        msg->setControlInfo(controlInfo);
 //std::cout <<"Sending HIP_R2 message" << endl;
 send(msg,"hipIPOut");
        HIPComm* comm2 = findCommHIT(destHIT,srcHIT);
       timeout = new cMessage("R2Timeout", HIP_TIMEOUT);
       timeout->setContextPointer(comm2);
       SendAgain *entry = new SendAgain();
       entry->communication= findCommHIT(destHIT,srcHIT);
       entry->Mensagem= (cMessage *) msg->dup();
 entry->timeoutHIP= timeout;
 entry->typeMsg= 4;
       reSendHIPList.insert(entry);
}

void HIP::sendUpdate1(HIPComm *comm, IPv6Address newAddr)
{
    //Preparing and sending message with changed IPv6address.
HITAddress don't changed
    HIPUpdate1 *msg = new HIPUpdate1();
    msg->setHit_sender(comm->localHIT());
    msg->setHit_receiver(comm->remoteHIT());
    msg->setPacket_type(14);
    msg->setSpi(comm->SPI());
    msg->setLocator(newAddr);
    msg->setSeq(0);
    msg->setHmac("MyMACAddress");
    msg->setHip_sig("000111");
    comm->setAddressStatus(1);
    comm->setLocalAddr(newAddr);
    IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
    controlInfo->setSrcAddr(comm->localAddr());
    controlInfo->setDestAddr(comm->remoteAddr());
    controlInfo->setProtocol(253);
```

```
    msg->setControlInfo(controlInfo);
    //std::cout <<"Sending message Update1 with src and
destination addresses: " << controlInfo->srcAddr() << ", " <<
controlInfo->destAddr() << endl;
    timeout = new cMessage("UP1Timeout", HIP_TIMEOUT);
    timeout->setContextPointer(comm);
    scheduleAt(simTime()+ACK_DELAYED, timeout);
    SendAgain *entry = new SendAgain();
    entry->communication = comm;
    entry->typeMsg= 14;
    entry->timeoutHIP= timeout;
    entry->Mensagem= (cMessage *) msg->dup();
    reSendHIPList.insert(entry);
    send(msg,"hipIPOut");
}

void HIP::UpdateComm(IPv6Address oldAddr, IPv6Address
newAddr)
{
    //Verify for who this node will send a HIP Update1
message
    IPv6Address novo = newAddr;
    //std::cout <<"Calling UpdateComm() with this arguments:
" << oldAddr << ", " << newAddr << endl;
    for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
           if (it->localAddr()==oldAddr) {
       //std::cout <<"Sending Update1/I1 to " << it-
>remoteAddr() << endl;
       it->setLocalAddr(novo);
       if (it->state()==5) { it->setAddressStatus(0);
sendUpdate1(&(*it), novo); }
       else sendI1(it->localHIT(), it->remoteHIT(), it-
>localAddr(), it->remoteAddr());
     } else if (it->remoteAddr()==oldAddr) {
       it->setRemoteAddr(novo);
       if (it->state()==5) it->setAddressStatus(0);
     }
    };
    return;
}

void HIP::sendClose(HIPComm *comm, IPv6Address srcHIT,
IPv6Address destHIT, IPv6Address srcAddr, IPv6Address
destAddr)
{
```

```
    //Informing peer that HIP Communication will be closed -
dont implemented
 HIPHeader *msg = new HIPHeader();
 msg->setHit_sender(destHIT);
 msg->setHit_receiver(srcHIT);
 msg->setPacket_type(18);
 comm->setState(6);
 IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
 controlInfo->setSrcAddr(destAddr);
 controlInfo->setDestAddr(srcAddr);
       controlInfo->setProtocol(253);
 msg->setControlInfo(controlInfo);
 send(msg,"hipIPOut");
}

void HIP::RVSRegister()
{
    //Register a node at Rendezvous Server
    //std::cout <<"Calling RVSRegister()" << endl;
    IPv6Address srcIPv6, destIPv6, srcHIT, destHIT;
    destHIT = myRVS;
    destIPv6 = findIPv6(myRVS);
    srcHIT = myHip;
    srcIPv6 = findIPv6(myHip);
    sendI1(srcHIT, destHIT, srcIPv6, destIPv6);
}

HIPComm* HIP::findCommIPv6(IPv6Address srcAddr, IPv6Address
destAddr)
{
    //Lookup at commHIP of comm between this two
IPv6Addresses
    HIPComm *entry = NULL;
    for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
        if (it->localAddr() == srcAddr && it-
>remoteAddr()==destAddr)
        {
            entry = &(*it);
            break;
        }
    };
    return entry;
}

HIPComm* HIP::findCommHIT(IPv6Address srcAddr, IPv6Address
destAddr)
```

```
{
    //Lookup at commHIP of comm between this two HITAddresses
    HIPComm *entry = NULL;
    for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
        if (it->localHIT()==srcAddr && it-
>remoteHIT()==destAddr)
        {
            entry = &(*it);
            break;
        }
    };
    return entry;
}

void HIP::addCommHIP(IPv6Address srcHIT, IPv6Address destHIT,
IPv6Address srcIPv6, IPv6Address destIPv6, int state)
{
    //Adding a new HIP Communication at commHip vector
    HIPComm *comm = new HIPComm;
    short spi = lastSPI;
    comm->setHIPComm(srcIPv6, destIPv6, srcHIT, destHIT,
state, spi);
    //std::cout <<"Adding new HIP Communication:" <<endl;
    commHip.push_back(*comm);
    printCommBrief(*comm);
    lastSPI = lastSPI+1;
}

void HIP::addHIPBuffer(IPv6Address srcHIT, IPv6Address
destHIT, IPv6Address srcIPv6, IPv6Address destIPv6, int
protocolo, cMessage* msg)
{
    //Adding a new message a HIPBuffer vector - Waiting for
finish Basic HIP Exchange or answer of Rendezvous Server
    //std::cout <<"Adding a new message at HIPBuffer" <<
endl;
    HIPQueue *entry = new HIPQueue;
    entry->setHIPQueue(srcHIT, destHIT, srcIPv6, destIPv6,
protocolo, msg);
    HIPBuffer.push_back(*entry);
    int pos = HIPBuffer.size()-1;
    //std::cout <<"srcHIT: " << HIPBuffer.at(pos).OrigemHIT()
<< ", destHIT: " << HIPBuffer.at(pos).DestinoHIT() << ",
srcIPv6: " << HIPBuffer.at(pos).OrigemIPv6() << ", destIPv6:
" << HIPBuffer.at(pos).DestinoIPv6() << ", protocolo: " <<
HIPBuffer.at(pos).Protocolo() << endl;
```

```
}

void HIP::sendHIPBuffer(IPv6Address srcIPv6, IPv6Address
destIPv6)
{
    //std::cout <<"sendHIPBUffer() => Checking HIPBuffer
Size: " << HIPBuffer.size() << endl;
    IPv6Address src = findHIT(srcIPv6);
    IPv6Address dest = findHIT(destIPv6);
    //std::cout <<"HIPBuffer: seaching this source and
destination addresses (IPv6 and HIT): "<< src << ", " << dest
<< ", " << srcIPv6 << ", " << destIPv6 << endl;
    HIPComm *comm = findCommHIT(src, dest);
    if (comm) //HIP Communication is already established -
Message will be send to destination peer
           for (BufferType::iterator it=HIPBuffer.begin();
it!=HIPBuffer.end(); it++)
           {
       //if (it->Mensagem() !=NULL) {
               if (it->OrigemIPv6() == srcIPv6 && it-
>DestinoIPv6() == destIPv6)
         {
             //std::cout <<"Found an entry of HIPBuffer with
addresses given" << endl;
             temporario.push_back(*it);             }
       }
           };
           for (BufferType::iterator it=HIPBuffer.begin();
it!=HIPBuffer.end(); it++)
           {
       IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
             controlInfo->setSrcAddr(srcIPv6);
            controlInfo->setDestAddr(destIPv6);
             controlInfo->setProtocol(it->Protocolo());
       cMessage* copy = (cMessage *) it->Mensagem()->dup();
       copy->setControlInfo(controlInfo);
       if (it->Protocolo() == 6) { send(copy, "hipIPTcpOut");
}
             else if (it->Protocolo() == 17) { send(copy,
"hipIPUdpOut"); }
           };
     do
     {
       temporario.pop_back();
     } while (temporario.size()>0);
    } else if (!comm) //Message was adding here while answer
of Rendezvous Server don't arrived
```

```
    {
            for (BufferType::iterator it=HIPBuffer.begin();
it!=HIPBuffer.end(); it++)
            {
       if (it->Mensagem() !=NULL) {
                 if (it->OrigemHIT() == src && it-
>DestinoHIT() == dest) {
             it->setOrigemIPv6(srcIPv6);
               it->setDestinoIPv6(destIPv6);
               IPv6ControlInfo *controlInfo = new
IPv6ControlInfo();
                     controlInfo->setSrcAddr(src);
                   controlInfo->setDestAddr(dest);
                   controlInfo->setProtocol(it->Protocolo());
               //std::cout <<"New ControlInfo: Src= " <<
controlInfo->srcAddr() << ", Dest= " << controlInfo-
>destAddr() << ", Protocol= " << controlInfo->protocol() <<
endl;
                   it->Mensagem()-
>setControlInfo(controlInfo);
               processTranspData(it->Mensagem());
                   it->setMsg(NULL);
                   break;
             }
       }
     };
    }
}

void HIP::addHIPCache(IPv6Address ipv6, IPv6Address hit)
{
    //Adding a new node at HIPCache
    BindingCacheEntry *comm = new BindingCacheEntry;
    comm->setBindingCacheEntry(ipv6, hit);
    HIPCache.push_back(*comm);
}

void HIP::updateHIPCache(IPv6Address ipv6, IPv6Address hit)
{
    //Update address of a node at HIPCache
    bool achou = false;
    //Lookup for node at HIPCache
    for (HIPBindingCache::iterator it=HIPCache.begin();
it!=HIPCache.end(); it++) {
        if (it->hitAddr()==hit) {
         it->setBindingCacheEntry(ipv6, hit);
```

```
          //std::cout <<"Update entry at HIPCache Vector
(IPv6,HIT): " << it->ipv6Addr() << " , " << it->hitAddr() <<
endl;
          achou = true;
          break;
        }
    };
    if (achou == false) addHIPCache(ipv6,hit);
}

IPv6Address HIP::findIPv6(IPv6Address enderecoHIP)
{
    //Retorn an IPv6 Address correspondent at HIT Address
given.
    bool achou = false;
    IPv6Address ipv6resp;
    //Lookup for HIT Address at HIPCache
    for (HIPBindingCache::iterator it=HIPCache.begin();
it!=HIPCache.end(); it++) {
        if (it->hitAddr()==enderecoHIP)
        {
           ipv6resp = it->ipv6Addr();
         achou = true;
           break;
        }
    };
    //If HIT Address don't exist at HIPCache, asking to
Rendezvous Server about it
    if (achou == false) askRVSIPv6(enderecoHIP);
    return ipv6resp;
}

IPv6Address HIP::findHIT(IPv6Address enderecoIpv6)
{
    //Retorn an HIT Address correspondent at IPv6 Address
given.
    bool achou = false;
    IPv6Address hitresp;
    //Lookup for IPv6 Address at HIPCache
    for (HIPBindingCache::iterator it=HIPCache.begin();
it!=HIPCache.end(); it++) {
        if (it->ipv6Addr()==enderecoIpv6)
        {
            hitresp = it->hitAddr();
            //achou = true;
            break;
        }
```

```
    };
    //If IPv6 Address don't exist at HIPCache, asking to
Rendezvous Server about it
    //if (achou == false) askRVSHip(enderecoIpv6);
    return hitresp;
}

/**void HIP::askRVSHip(IPv6Address enderecoIpv6)
{
    //Sending message with question about IPv6 Address to
Rendezvous Server
    HIPAsk *msg = new HIPAsk();
    msg->setHit_sender(myHip);
    msg->setHit_receiver(myRVS);
    msg->setPacket_type(26);
    msg->setTarget(enderecoIpv6);
    IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
    controlInfo->setSrcAddr(findIPv6(myHip));
    controlInfo->setDestAddr(findIPv6(myRVS));
    controlInfo->setProtocol(IP_PROT_IPv6_HIP);
    msg->setControlInfo(controlInfo);
    //std::cout <<"Sending AskRVSHip to Rendezvous Server" <<
endl;
    send(msg,"hipIPOut");
}**/

void HIP::askRVSIPv6(IPv6Address enderecoHIP)
{
    //Sending message with question about IPv6 Address to
Rendezvous Server
   HIPAsk *msg = new HIPAsk();
   msg->setHit_sender(myHip);
   msg->setHit_receiver(myRVS);
   msg->setPacket_type(25);
   msg->setTarget(enderecoHIP);
   IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
   controlInfo->setSrcAddr(findIPv6(myHip));
   controlInfo->setDestAddr(findIPv6(myRVS));
   controlInfo->setProtocol(IP_PROT_IPv6_HIP);
   msg->setControlInfo(controlInfo);
   timeout = new cMessage("ASKTimeout", HIP_TIMEOUT);
   timeout->setContextPointer(NULL);
   scheduleAt(simTime()+ACK_DELAYED, timeout);
   //std::cout <<"Sending AskRVSIPv6 to Rendezvous Server" <<
endl;
   send(msg,"hipIPOut");
}
```

```
void HIP::insertAllHIT()
{
    //Write IPv6 and HIT addresses of known nodes - XML Parse
    //std::cout <<"Calling insertAllHIT()" << endl;
    cXMLElement *dnsFile = par("DNSFile");
    IPv6Address ipv6Address, hitAddress;
    cXMLElementList addrList = dnsFile-
>getElementsByTagName("inetAddr");
    cXMLElementList hitList = dnsFile-
>getElementsByTagName("hipAddr");
    for (unsigned int k=0; k<addrList.size(); k++)
    {
       cXMLElement *ipv6node = addrList[k];
       cXMLElement *hitnode = hitList[k];
       IPv6Address ipv6Address(ipv6node->getNodeValue());
       IPv6Address hitAddress(hitnode->getNodeValue());
       addHIPCache(ipv6Address, hitAddress);
       //std::cout <<"Vector HIPCache Entry (IPv6,HIT): " <<
HIPCache.at(k).ipv6Addr() << " , " <<
HIPCache.at(k).hitAddr() << endl;
    };
}

void HIP::receiveChangeNotification(int category,
cPolymorphic *details)
{
    //Define procedures when special //std::coutents occurs
    //Define procedures when special //std::coutents occurs
    Enter_Method_Silent();
    printNotificationBanner(category, details);
    if (category == NF_IPv6_INTERFACECONFIG_CHANGED) {
        EV <<"IPv6 Interface Changed; update my list of
communications..." << endl;
        IPv6Address novo =
check_and_cast<IPv6InterfaceData*>(details)-
>preferredAddress();
        updateHIPCache(novo, myHip);
 EV <<"HIPCache updated. Calling RegisterRVS() or
UpdateComm()" << endl;
        handover_num += handover_num;
 endHandoff = simTime();
        handoffVector.record(endHandoff - startHandoff);
        HIPComm* testeRVS = findCommHIT(myHip, myRVS);
 if (testeRVS) {
       UpdateComm(controlAddr, novo);
        } else { RVSRegister(); }
```

```
    } else if (category == NF_L2_BEACON_LOST) {
 startHandoff = simTime();
 controlAddr = findIPv6(myHip);
 //std::cout << "Valor de controlAddr: " << controlAddr <<
endl;
 updateHIPCache("0:0:0:0:0:0:0:0", myHip);
    }
}
```

8.2. Arquivo HIP.h

```
#ifndef __HIP_H__
#define __HIP_H__

#include <omnetpp.h>
#include <cmodule.h>
#include <cmessage.h>
#include <vector>
#include "INETDefs.h"
#include "IPv6Address.h"
#include "RoutingTable6.h"
#include "NotificationBoard.h"
#include "IPv6InterfaceData.h"
#include "InterfaceTable.h"

class IPv6Datagram;
class IPv6InterfaceData;

enum HipState
{
    HIP_S_INIT        = 0,
    HIP_S_I1_SENT     = 1,
    HIP_S_R1_SENT     = 2,
    HIP_S_I2_SENT     = 3,
    HIP_S_R2_SENT     = 4,
    HIP_S_ESTABLISHED = 5,
    HIP_S_CLOSE_SENT  = 6,
    HIP_S_CLOSE_RCVD  = 7,
    HIP_S_CLOSED      = 8
};

enum status_addr {
    Unverified = 0,
    Active = 1,
    Deprecated = 2
```

```
};

class HIPComm: public cPolymorphic
{
  protected:
      IPv6Address _localAddr;
      IPv6Address _remoteAddr;
      IPv6Address _localHIT;
      IPv6Address _remoteHIT;
      short _SPI;
      int _state;
      simtime_t _expireTime;
      int _addressStatus;

  public:
    HIPComm() {}
    void setHIPComm(const IPv6Address& localAddr, const
IPv6Address& remoteAddr, const IPv6Address& localHIT, const
IPv6Address& remoteHIT, const int state, const short spi)
    {
     _localAddr = localAddr;
     _remoteAddr = remoteAddr;
     _localHIT = localHIT;
     _remoteHIT = remoteHIT;
     _state = state;
     _SPI = spi;
     _expireTime = 120;
     _addressStatus = 1;
    }
    void setLocalAddr(const IPv6Address& localAddr)
{_localAddr = localAddr;};
    void setRemoteAddr(const IPv6Address& remoteAddr)
{_remoteAddr = remoteAddr;};
    void setLocalHIT(const IPv6Address& localHIT)  {_localHIT
= localHIT;};
    void setRemoteHIT(const IPv6Address& remoteHIT)
{_remoteHIT = remoteHIT;};
    void setSPI(const int spi) {_SPI = spi;};
    void setState(const int state) {_state = state;};
    void setExpireTime(simtime_t expireTime) {_expireTime =
expireTime;};
    void setAddressStatus(const int addressStatus)
{_addressStatus = addressStatus;};
    const IPv6Address& localAddr() const {return
_localAddr;};
    const IPv6Address& remoteAddr() const {return
_remoteAddr;};
```

```
    const IPv6Address& localHIT() const {return _localHIT;};
    const IPv6Address& remoteHIT() const {return
_remoteHIT;};
    int SPI() {return _SPI;};
    int state() {return _state;};
    simtime_t expireTime() const  {return _expireTime;};
    int addressStatus() {return _addressStatus;};
};

class BindingCacheEntry: public cPolymorphic
{
  protected:
      IPv6Address Ipv6Addr;
      IPv6Address HitAddr;
  public:
    BindingCacheEntry() {};
    void setBindingCacheEntry(const IPv6Address& ipv6, const
IPv6Address& hit)
    {
      Ipv6Addr = ipv6;
      HitAddr = hit;
    };
    const IPv6Address& ipv6Addr() const {return Ipv6Addr;};
    const IPv6Address& hitAddr() const {return HitAddr;};
};

class HIPQueue: public cPolymorphic
{
  protected:
      IPv6Address origemHIT;
      IPv6Address destinoHIT;
      IPv6Address origemIPv6;
      IPv6Address destinoIPv6;
      int protocolo;
      cMessage* msg;

  public:
    HIPQueue() {};
    void setHIPQueue(const IPv6Address& srcHIT, const
IPv6Address& destHIT, const IPv6Address& srcIPv6, const
IPv6Address& destIPv6, int protocol, cMessage* data)
    {
      origemHIT = srcHIT;
      destinoHIT = destHIT;
      origemIPv6 = srcIPv6;
      destinoIPv6 = destIPv6;
      protocolo = protocol;
```

```
      msg = data;
    };
    void setOrigemIPv6(IPv6Address src) { origemIPv6=src; };
    void setDestinoIPv6(IPv6Address dest) { destinoIPv6=dest;
};
    void setMsg(cMessage* data) { msg=data; };
    const IPv6Address& OrigemHIT() const {return origemHIT;};
    const IPv6Address& DestinoHIT() const {return
destinoHIT;};
    const IPv6Address& OrigemIPv6() const {return
origemIPv6;};
    const IPv6Address& DestinoIPv6() const {return
destinoIPv6;};
    int Protocolo() {return protocolo;};
    cMessage* Mensagem() {return msg;}
};

class HIP: public cSimpleModule, public INotifiable
{
  protected:
    struct SendAgain {
 HIPComm* communication;
 cMessage* timeoutHIP;
 int typeMsg;
 cMessage* Mensagem;
    };
    InterfaceTable *ift;
    RoutingTable6 *rt;
    const char* myHip;
    const char* myRVS;
    cMessage* timerRegister;
    cMessage* timeout;
    double delay;
    short lastSPI;
    int handover_num;
    simtime_t startHandoff, endHandoff;
    IPv6Address srcIPv6, destIPv6, srcHIT, destHIT,
controlAddr;
    typedef std::vector<HIPComm> HIPCommList;
    HIPCommList commHip;
    typedef std::vector<BindingCacheEntry> HIPBindingCache;
    HIPBindingCache HIPCache;
    typedef std::vector<HIPQueue > BufferType;
    BufferType HIPBuffer;
    BufferType temporario;
    typedef std::set<SendAgain*> retransmissionList;
    retransmissionList reSendHIPList;
```

```
    void processExpireTime(HIPComm *comm);
    void insertAllHIT();
    void RVSRegister();
    // INotifiable
    virtual void receiveChangeNotification(int category,
cPolymorphic *details);

  private:
    NotificationBoard *nb;
    cOutVector handoffVector;

  public:
    HIP();
    ~HIP();
    virtual void initialize(int stage);
    virtual void handleMessage(cMessage*);
    void sendI1(IPv6Address srcHIT, IPv6Address destHIT,
IPv6Address srcAddr, IPv6Address destAddr);
    void sendI2(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr);
    void sendR1(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr);
    void sendR2(IPv6Address destHIT, IPv6Address srcHIT,
IPv6Address destAddr, IPv6Address srcAddr);
    void sendI2Rendezvous(IPv6Address destHIT, IPv6Address
srcHIT, IPv6Address destAddr, IPv6Address srcAddr);
    void sendUpdate1(HIPComm *comm, IPv6Address newAddr);
    virtual void UpdateComm(IPv6Address oldAddr, IPv6Address
newAddr);
    void sendClose(HIPComm *comm, IPv6Address srcHIT,
IPv6Address destHIT, IPv6Address srcAddr, IPv6Address
destAddr);
    void printCommBrief(HIPComm comm);
    void addHIPBuffer(IPv6Address srcHIT, IPv6Address
destHIT, IPv6Address srcIPv6, IPv6Address destIPv6, int
protocol, cMessage* msg);
    void sendHIPBuffer(IPv6Address srcIPv6, IPv6Address
destIPv6);
    void addHIPCache(IPv6Address ipv6, IPv6Address hit);
    void updateHIPCache(IPv6Address ipv6, IPv6Address hit);
    void addCommHIP(IPv6Address srcHIT, IPv6Address destHIT,
IPv6Address srcIPv6, IPv6Address destIPv6, int state);
    IPv6Address findIPv6(IPv6Address enderecoHIP);
    IPv6Address findHIT(IPv6Address enderecoIpv6);
    //void askRVSHip(IPv6Address enderecoIpv6);
    void askRVSIPv6(IPv6Address enderecoHIP);
```

```
    HIPComm* findCommIPv6(IPv6Address srcAddr, IPv6Address
destAddr);
    HIPComm* findCommHIT(IPv6Address srcAddr, IPv6Address
destAddr);
    SendAgain* findRetransmissionEntry(HIPComm *comm);
    void processTranspData(cMessage *msg);
    virtual void processIPv6Datagram(cMessage *datagram);
};

#endif
```

8.3. Arquivo HIP.ned

```
simple HIP
    parameters:
      procDelay: numeric const,
      DNSFile: xml,
      rvsHITAddress: string,
      hipAddress: string;
    gates:
 in: hipTcpIn;
 out: hipTcpOut;
 in: hipUdpIn;
 out: hipUdpOut;
 in: hipIPTcpIn;
 out: hipIPTcpOut;
 in: hipIPUdpIn;
 out: hipIPUdpOut;
 in: hipIPIn;
 out: hipIPOut;
endsimple
```

8.4. Arquivo HIP.msg

```
// inserir includes
cplusplus {{
# include <iostream>
# include "HIP_m.h"
# include "../Contract/IPv6Address.h"
}};
```

```
class noncobject IPv6Address;

  // Definição de tipos de pacotes:
  enum hip_packet_type {
    I1 = 1;
    R1 = 2;
    I2 = 3;
    R2 = 4;
    R1_Rendezvous = 5;
    I2_Rendezvous = 6;
    R2_Rendezvous = 7;
    Update1 = 14;
    Update2 = 15;
    UpdateAck = 16;
    Notify = 17;
    Close = 18;
    Close_Ack = 19;
    AskRVSIPv6 = 25;
    AskRVSHip = 26;
    RespRVSIPv6 = 27;
    RespRVSHip = 28;
  };


message HIPHeader {
 fields:
     int packet_type enum(hip_packet_type);
     int hip_version = 1; //0001 = 4 bits
     //double checksum;
     //int controls;
     IPv6Address hit_sender; // tamanho fixo = 128 bits
     IPv6Address hit_receiver; // tamanho fixo = 128 bits
};

message HIPUpdate1 extends HIPHeader {
 fields:
     short spi;
     IPv6Address locator;
     int seq;
     string hmac;
```

```
        string hip_sig;
};

message HIPUpdate2 extends HIPHeader {
 fields:
        short spi;
        int seq;
        int ack;
        string hmac;
        string hip_sig;
        string echo_req;
};

message HIPUpdateAck extends HIPHeader {
 fields:
        int ack;
        string hmac;
        string hip_sig;
        string echo_resp;
};

message HIP_I2 extends HIPHeader {
 fields:
        string solution;
        string DiffieHellman;
        string hip_transf;
        IPv6Address hostId;
        string hip_sig;
        string hmac;
        string echo_req;
};

message HIP_R1 extends HIPHeader {
 fields:
        string puzzle;
        string DiffieHellman;
        string hip_transf;
        IPv6Address hostId;
        string hip_sig2;
};
```

```
    message HIP_R2 extends HIPHeader {
     fields:
         string hmac2;
         string hip_sig;
    };

    message HIPAsk extends HIPHeader {
     fields:
         IPv6Address target;
    };

    message HIPResp extends HIPHeader {
     fields:
         IPv6Address target;
         IPv6Address answer;
    };

    //Rendez-vous Server I2 packet extends HIP_I2 with time of
validade of registry.
    message HIP_I2_Rendezvous extends HIP_I2 {
     fields:
         string service;
         short validade;
    };

    //Rendez-vous Server R1 packet extends HIP_R1 with
solicitation of registration.
    message HIP_R1_Rendezvous extends HIP_R1 {
     fields:
         string typeService; //name of services
         short validade; //time of validade defined
    };

    //Rendez-vous Server R2 packet extends HIP_R2 with response
of registration's solicitation.
    message HIP_R2_Rendezvous extends HIP_R2 {
     fields:
         string solicitation; //name of service successfull
registried
         short validade; //time of validade defined
    };
```

8.5. Arquivo HIPRendezvous.h

```
#ifndef __HIPRENDEZVOUS_H__
#define __HIPRENDEZVOUS_H__

#include <omnetpp.h>
#include <cmodule.h>
#include <cmessage.h>
#include <vector>
#include "INETDefs.h"
#include "HIP.h"

class HIPRendezvous: public HIP
{
  protected:
    cMessage* timeout;
    bool isRegistered(IPv6Address hit);
    void insertAllHIT();
    void answerRVSIPv6(IPv6Address srcAddr, IPv6Address
srcHIT, IPv6Address hit);
    void answerRVSHip(IPv6Address srcAddr, IPv6Address
srcHIT, IPv6Address ipv6);
    void sendR1Rendezvous(IPv6Address destHIT, IPv6Address
srcHIT, IPv6Address destAddr, IPv6Address srcAddr);
    void sendR2Rendezvous(IPv6Address destHIT, IPv6Address
srcHIT, IPv6Address destAddr, IPv6Address srcAddr);

  public:
    HIPRendezvous() {};
    virtual void initialize(int stage);
    virtual void handleMessage(cMessage*);
    virtual void UpdateComm(IPv6Address oldAddr, IPv6Address
newAddr);
    virtual void processIPv6Datagram(cMessage *datagram);
};

#endif
```

8.6.
Arquivo HIPRendezvous.cc

```
#include <omnetpp.h>
#include <vector>
#include "HIPRendezvous.h"
#include "IPv6Datagram.h"
#include "IPv6ControlInfo.h"
#include "IPAddressResolver.h"
#include "HIP_m.h"

Define_Module(HIPRendezvous);

void HIPRendezvous::initialize(int stage)
{
   std::cout << "Iniciando parâmetros do módulo
HIPRendezvous..." << endl;
   //parameters
   delay = par("procDelay");
   myHip = par("hipAddress");
   timeout=NULL;
   //initiate HIPBindingCache
   insertAllHIT();
}

void HIPRendezvous::handleMessage(cMessage *msg)
{
    if (msg->isSelfMessage())
    {
  HIPComm* comm = (HIPComm*) msg->contextPointer();
        switch (comm->state())
        {
              case(2):
        {
              sendR1Rendezvous(comm->localHIT(), comm-
>remoteHIT(), comm->localAddr(), comm->remoteAddr());
              break;
        }
  };
  delete msg;
   } else if (msg->arrivalGate()->isName("hipIPIn"))
```

```
        {
      // datagram from network or from ND: localDeliver and/or
route
      std::cout << "Mensagem HIP entregue - origem: camada IPv6."
<< endl;
            processIPv6Datagram(msg);
        }
    }

    // Tratamento de mensagens
    void HIPRendezvous::processIPv6Datagram(cMessage *datagram)
    {
        std::cout << "Mensagem entregue para
processIPv6Datagram." << endl;
        //Informação do datagrama recebido
        IPv6ControlInfo *controlInfo = (IPv6ControlInfo
*)datagram->removeControlInfo();
        srcIPv6 = controlInfo->srcAddr();
        destIPv6 = controlInfo->destAddr();
        int protocolo = controlInfo->protocol();
        HIPComm* comm = findCommIPv6(srcIPv6, destIPv6);
        HIPHeader *temp = check_and_cast<HIPHeader *>(datagram);

        //To replace use of SPI at HIPCommunication - Update1
Message
        HIPComm* testUpdate = findCommHIT(temp->getHit_sender(),
temp->getHit_receiver());
        if (testUpdate && temp->getPacket_type()==14) {
            std::cout << "HIPRendezvous: Update message
received... Update communication and HIPCache..." << endl;
      srcHIT=temp->getHit_sender();
      updateHIPCache(srcIPv6,srcHIT);
      destHIT=temp->getHit_receiver();
      testUpdate->setLocalAddr(srcIPv6);
      testUpdate->setRemoteAddr(destIPv6);
      comm = &(*testUpdate);
      HIPComm* test2=findCommHIT(destHIT, srcHIT);
      test2->setLocalAddr(destIPv6);
      test2->setRemoteAddr(srcIPv6);
        } else if (temp->getPacket_type()!=14) {
      srcHIT = temp->getHit_sender();
```

```
            destHIT = temp->getHit_receiver();
        }

         std::cout << "HIPRendezvous: HIT Addresses (src, dest)
and HIP message type: " << srcHIT << ", " << destHIT << " e " <<
temp->getPacket_type() << endl;

         if (findIPv6(srcHIT)!=srcIPv6 && !comm)
      updateHIPCache(srcIPv6,srcHIT);

         //Test if is an initiate HIP Exchange Basic
         if (!comm) {
      if (temp->getPacket_type()==1 && destHIT==myHip) {
            std::cout << "Mensagem I1 para Rendezvous sem conexão"
<< endl;
            addCommHIP(srcHIT, destHIT, srcIPv6, destIPv6, 0);
            addCommHIP(destHIT, srcHIT, destIPv6, srcIPv6, 0);
                  comm = findCommIPv6(srcIPv6, destIPv6);
            std::cout << "Solicitando conexão com servidor
Rendezvous - Mensagem tipo HIP_I1  ";
            comm->setState(2);
            HIPComm* comm2 = findCommIPv6(destIPv6, srcIPv6);
            comm2->setState(2);
            sendR1Rendezvous(destHIT, srcHIT, destIPv6, srcIPv6);
                  timeout = new cMessage("timeout");
                  timeout->setContextPointer(comm2);
                  scheduleAt(simTime()+0.2, timeout);
                  delete datagram;
            } else if (temp->getPacket_type()>1 && destHIT==myHip)
            {
            switch (temp->getPacket_type())
            {
               case(25): //AskRVSIPv6 Packet
               {
                  std::cout << "Mensagem AskRVSIPv6 para
Rendezvous" << endl;
                  HIPAsk *data = check_and_cast<HIPAsk
*>(datagram);
                  answerRVSIPv6(srcIPv6, srcHIT, data-
>getTarget());
                        delete datagram;
```

```
                  break;
               }
               case(26): //AskRVSHip Packet
               {
                  std::cout << "Mensagem AskRVSHip para
Rendezvous" << endl;
                  HIPAsk *data = check_and_cast<HIPAsk
*>(datagram);
                  answerRVSHip(srcIPv6, srcHIT, data-
>getTarget());
                          delete datagram;
                  break;
               }
            };
      } else if (temp->getPacket_type()==1 && destHIT!=myHip)
            {
            std::cout << "Mensagem I1 para host registrado em
Rendezvous" << endl;
            //Test if src is sending HIP I1 Packet to an registred
hosts
            if (isRegistered(destHIT))
            {
               HIPHeader *msg = new HIPHeader();
               msg->setHit_sender(srcHIT);
               msg->setHit_receiver(destHIT);
               msg->setPacket_type(1);
               controlInfo->setSrcAddr(srcIPv6);
               controlInfo->setDestAddr(findIPv6(destHIT));
               controlInfo->setProtocol(IP_PROT_IPv6_HIP);
               msg->setControlInfo(controlInfo);
               std::cout << "Servidor Rendezvous encaminhando
mensagem HIP I1" << endl;
               send(msg,"hipIPOut");
                     delete datagram;
               }
      }
         } else if (comm) {
            if (temp->getPacket_type()>1 && destHIT==myHip) {
            std::cout << "Mensagem diferente de I1 para Rendezvous
com conexão" << endl;
            switch (temp->getPacket_type())
```

```
            {
               case(6): //HIP_I2_Rendezvous Packet
               {
                  if (comm->state()==2)
                  {
                        HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
                        comm->setState(5);
                        comm2->setState(5);
                        if (timeout && ((HIPComm*)timeout-
>contextPointer()==comm ||(HIPComm*)timeout-
>contextPointer()==comm2)) cancelEvent(timeout);
                        sendR2Rendezvous(destHIT, srcHIT,
destIPv6, srcIPv6);
                  } else if (comm->state()==5) {
sendR2Rendezvous(destHIT, srcHIT, destIPv6, srcIPv6); }
                          delete datagram;
                  break;
               }
               case(14): //HIP Update1 Packet
               {
                  if (comm->state()==5)
                  {
                        HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
                        comm->setAddressStatus(0);
                        comm2->setAddressStatus(0);
                        if (timeout && ((HIPComm*)timeout-
>contextPointer()==comm ||(HIPComm*)timeout-
>contextPointer()==comm2)) cancelEvent(timeout);
                        HIPUpdate1 *data =
check_and_cast<HIPUpdate1 *>(datagram);
                        HIPUpdate2 *msg = new HIPUpdate2();
                        msg->setHit_sender(destHIT);
                        msg->setHit_receiver(srcHIT);
                        msg->setPacket_type(15);
                        msg->setSpi(comm->SPI());
                        msg->setSeq(data->getSeq());
                        msg->setAck(data->getSeq()+1);
                        msg->setHmac("MyMACAddress");
                        msg->setHip_sig("000111");
```

```
                        msg->setEcho_req("simulator");
                        controlInfo->setSrcAddr(destIPv6);
                        controlInfo->setDestAddr(data-
>getLocator());
                              controlInfo-
>setProtocol(IP_PROT_IPv6_HIP);
                              msg->setControlInfo(controlInfo);
                        send(msg,"hipIPOut");
                  }
                          delete datagram;
                  break;
               }
               case(16): //HIP Update Ack Packet
              {
                  if ((comm->state()==5 && comm-
>addressStatus()==0))
                  {
                        HIPComm* comm2 = findCommIPv6(destIPv6,
srcIPv6);
                        if (timeout && ((HIPComm*)timeout-
>contextPointer()==comm ||(HIPComm*)timeout-
>contextPointer()==comm2)) cancelEvent(timeout);
                        comm->setAddressStatus(1);
                        comm2->setAddressStatus(1);
                        comm->setExpireTime(120);
                        comm2->setExpireTime(120);
                  }
                          delete datagram;
                  break;
               }
               case(25): //AskRVSIPv6 Packet
               {
                  std::cout << "Mensagem AskRVSIPv6 para
Rendezvous" << endl;
                  HIPAsk *data = check_and_cast<HIPAsk
*>(datagram);
                  answerRVSIPv6(srcIPv6, srcHIT, data-
>getTarget());
                          delete datagram;
                  break;
               }
```

```
                case(26): //AskRVSHip Packet
                {
                   std::cout << "Mensagem AskRVSHip para
Rendezvous" << endl;
                   HIPAsk *data = check_and_cast<HIPAsk
*>(datagram);
                   answerRVSHip(srcIPv6, srcHIT, data-
>getTarget());
                           delete datagram;
                   break;
                }
                   };
            } else if (temp->getPacket_type()==1 &&
destHIT!=myHip)
            {
            std::cout << "Mensagem I1 para host registrado em
Rendezvous" << endl;
            //Test if src is sending HIP I1 Packet to an registred
hosts
            if (isRegistered(destHIT))
            {
               HIPHeader *msg = new HIPHeader();
               msg->setHit_sender(srcHIT);
               msg->setHit_receiver(destHIT);
               msg->setPacket_type(1);
               controlInfo->setSrcAddr(srcIPv6);
               controlInfo->setDestAddr(findIPv6(destHIT));
               controlInfo->setProtocol(IP_PROT_IPv6_HIP);
               msg->setControlInfo(controlInfo);
               std::cout << "Servidor Rendezvous encaminhando
mensagem HIP I1" << endl;
               send(msg,"hipIPOut");
                     delete datagram;
                }
            }
         }
     }

     void HIPRendezvous::UpdateComm(IPv6Address oldAddr,
IPv6Address newAddr)
     {
```

```
        IPv6Address novo = newAddr;
        for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
            if (it->localAddr()==oldAddr) { it-
>setLocalAddr(novo); }
            else if (it->remoteAddr()==oldAddr) { it-
>setRemoteAddr(novo); }
        };
        return;
     }

     bool HIPRendezvous::isRegistered(IPv6Address hit)
     {
        bool result = false;
        for (HIPCommList::iterator it=commHip.begin();
it!=commHip.end(); it++) {
           if (it->localHIT()==hit && it->state()==5)
           {
              result = true;
              break;
           }
        };
        return result;
     }

     void HIPRendezvous::insertAllHIT()
     {
        std::cout << "Entrando em insertAllHIT()" << endl;
        cXMLElement *dnsFile = par("DNSFile");
        IPv6Address ipv6Address, hitAddress;
        cXMLElementList addrList = dnsFile-
>getElementsByTagName("inetAddr");
        cXMLElementList hitList = dnsFile-
>getElementsByTagName("hipAddr");
        for (unsigned int k=0; k<addrList.size(); k++)
        {
            cXMLElement *ipv6node = addrList[k];
            cXMLElement *hitnode = hitList[k];
            IPv6Address ipv6Address(ipv6node->getNodeValue());
            IPv6Address hitAddress(hitnode->getNodeValue());
            addHIPCache(ipv6Address, hitAddress);
```

```
            std::cout << "Entrada de Vetor HIPCache (IPv6,HIT): "
<< HIPCache.at(k).ipv6Addr() << " , " << HIPCache.at(k).hitAddr()
<< endl;
        };
     }

     void HIPRendezvous::answerRVSIPv6(IPv6Address srcAddr,
IPv6Address srcHIT, IPv6Address hit)
     {
        IPv6Address ipv6resp;
        for (HIPBindingCache::iterator it=HIPCache.begin();
it!=HIPCache.end(); it++) {
          if (it->hitAddr()==hit)
          {
            ipv6resp = it->ipv6Addr();
            break;
          }
        };
        if (srcAddr != "0:0:0:0:0:0:0:0") {
       HIPResp *msg = new HIPResp();
       msg->setHit_sender(myHip);
       msg->setHit_receiver(srcHIT);
       msg->setPacket_type(27);
       msg->setTarget(hit);
       msg->setAnswer(ipv6resp);
       IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
       controlInfo->setSrcAddr(findIPv6(myHip));
       controlInfo->setDestAddr(srcAddr);
       controlInfo->setProtocol(IP_PROT_IPv6_HIP);
       msg->setControlInfo(controlInfo);
       std::cout << "Enviando mensagem RespRVSHip" << endl;
       send(msg,"hipIPOut");
          }
     }

     void HIPRendezvous::answerRVSHip(IPv6Address srcAddr,
IPv6Address srcHIT, IPv6Address ipv6)
     {
        IPv6Address hitresp;
        for (HIPBindingCache::iterator it=HIPCache.begin();
it!=HIPCache.end(); it++) {
```

```
            if (it->ipv6Addr()==ipv6)
            {
              hitresp = it->hitAddr();
              break;
            }
          };
          if (srcAddr != "0:0:0:0:0:0:0:0") {
         HIPResp *msg = new HIPResp();
         msg->setHit_sender(myHip);
         msg->setHit_receiver(srcHIT);
         msg->setPacket_type(28);
         msg->setTarget(ipv6);
         msg->setAnswer(hitresp);
         IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
         controlInfo->setSrcAddr(findIPv6(myHip));
         controlInfo->setDestAddr(srcAddr);
         controlInfo->setProtocol(IP_PROT_IPv6_HIP);
         msg->setControlInfo(controlInfo);
         std::cout << "Enviando mensagem RespRVSHip" << endl;
         send(msg,"hipIPOut");
           }
        }

    void HIPRendezvous::sendR1Rendezvous(IPv6Address destHIT,
IPv6Address srcHIT, IPv6Address destAddr, IPv6Address srcAddr)
    {
     HIP_R1_Rendezvous *msg = new HIP_R1_Rendezvous();
     msg->setHit_sender(destHIT);
     msg->setHit_receiver(srcHIT);
     msg->setPacket_type(5);
     msg->setPuzzle("New Protocol");
     msg->setDiffieHellman("group 1");
     msg->setHip_transf("AES with Hmac_SHA-1-96");
     msg->setHostId(destHIT);
     msg->setHip_sig2("000000");
     msg->setTypeService("Register");
     msg->setValidade(120);
     IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
     controlInfo->setSrcAddr(destIPv6);
     controlInfo->setDestAddr(srcIPv6);
           controlInfo->setProtocol(IP_PROT_IPv6_HIP);
```

```
            msg->setControlInfo(controlInfo);
       std::cout << "Enviando mensagem HIP_R1_Rendezvous" << endl;
       send(msg,"hipIPOut");
      }

     void HIPRendezvous::sendR2Rendezvous(IPv6Address destHIT,
IPv6Address srcHIT, IPv6Address destAddr, IPv6Address srcAddr)
     {
       HIP_R2_Rendezvous *msg = new HIP_R2_Rendezvous();
       msg->setHit_sender(destHIT);
       msg->setHit_receiver(srcHIT);
       msg->setPacket_type(7);
       msg->setHmac2("MyMACAddress");
       msg->setHip_sig("000111");
       msg->setSolicitation("Register");
       msg->setValidade(120);
       IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
       controlInfo->setSrcAddr(destIPv6);
       controlInfo->setDestAddr(srcIPv6);
              controlInfo->setProtocol(IP_PROT_IPv6_HIP);
              msg->setControlInfo(controlInfo);
       std::cout << "Enviando mensagem HIP_R2_Rendezvous" << endl;
       send(msg,"hipIPOut");
      }
```

8.7. Arquivo IPAddressResolver.h

```
     #ifndef __IPADDRESSRESOLVER_H
     #define __IPADDRESSRESOLVER_H

     #include <omnetpp.h>
     #include "IPvXAddress.h"

     class InterfaceTable;
     class InterfaceEntry;
     class RoutingTable;
     class RoutingTable6;
     class NotificationBoard;

     /**
```

```
     * Utility class for finding IPv4 or IPv6 address of a host
or router.
     *
     * Syntax variations understood:
     *    - literal IPv4 address: "186.54.66.2"
     *    - literal IPv6 address:
"3011:7cd6:750b:5fd6:aba3:c231:e9f9:6a43"
     *    - module name: "server", "subnet.server[3]"
     *    - interface of a host or router: "server/eth0",
"subnet.server[3]/eth0"
     *    - IPv4 or IPv6 address of a host or router:
"server(ipv4)",
     *      "subnet.server[3](ipv6)"
     *    - IPv4 or IPv6 address of an interface of a host or
router:
     *      "server/eth0(ipv4)", "subnet.server[3]/eth0(ipv6)"
     *    - routerId: "router1/routerId", "R1/routerId"
     */
    class INET_API IPAddressResolver
    {
      private:
        // internal
        IPAddress getIPv4AddressFrom(InterfaceTable *ift);
        // internal
        IPv6Address getIPv6AddressFrom(InterfaceTable *ift);
        // internal
        IPv6Address getIPv6AddressFrom(InterfaceTable *ift, int
scope);
        // internal
        IPv6Address getInterfaceIPv6Address(InterfaceEntry *ie);

      public:
        enum {
            ADDR_PREFER_IPv4,
            ADDR_PREFER_IPv6,
            ADDR_IPv4,
            ADDR_IPv6
        };

      public:
        IPAddressResolver() {}
```

```
        ~IPAddressResolver() {}

        //insert by mroenick at 06/fev/08
        IPv6Address getIPv6AddressFrom(InterfaceEntry *ie, int
addrType);

        /**
         * Accepts dotted decimal notation ("127.0.0.1"), module
name of the host
         * or router ("host[2]"), and empty string (""). For the
latter, it returns
         * the null address. If module name is specified, the
module will be
         * looked up using <tt>simulation.moduleByPath()</tt>,
and then
         * addressOf() will be called to determine its IP
address.
         */
        IPvXAddress resolve(const char *str, int
addrType=ADDR_PREFER_IPv6);
        IPv6Address resolveIPv6(const char *str, int
addrType=ADDR_PREFER_IPv6);

        /**
         * Similar to resolve(), but returns false (instead of
throwing an error)
         * if the address cannot be resolved because the given
host (or interface)
         * doesn't have an address assigned yet. (It still throws
an error
         * on any other error condition).
         */
        bool tryResolve(const char *str, IPvXAddress& result, int
addrType=ADDR_PREFER_IPv6);
        bool tryResolveIPv6(const char *str, IPv6Address& result,
int addrType=ADDR_PREFER_IPv6);

        /** @name Utility functions supporting resolve() */
        //@{
        /**
```

```
         * Returns IPv4 or IPv6 address of the given host or
router.
         *
         * This function uses routingTableOf() to find the
RoutingTable module,
         * then invokes getAddressFrom() to extract the IP
address.
         */
        IPvXAddress addressOf(cModule *host, int
addrType=ADDR_PREFER_IPv6);
        IPv6Address IPv6AddressOf(cModule *host, int
addrType=ADDR_PREFER_IPv6);
        /**
         * Similar to addressOf(), but only looks at the given
interface
         */
        IPvXAddress addressOf(cModule *host, const char *ifname,
int addrType=ADDR_PREFER_IPv6);
        IPv6Address IPv6AddressOf(cModule *host, const char
*ifname, int addrType=ADDR_PREFER_IPv6);

        /**
         * Returns the router Id of the given router. Router Id
is obtained from
         * the routerId() method of the RoutingTable submodule.
         */
        IPAddress routerIdOf(cModule *host);

        /**
         * Returns the IPv4 or IPv6 address of the given host or
router, given its InterfaceTable
         * module. For IPv4, the first usable interface address
is chosen.
         */
        IPvXAddress getAddressFrom(InterfaceTable *ift, int
addrType=ADDR_PREFER_IPv6);

        /**
         * Returns the IPv4 or IPv6 address of the given
interface (of a host or router).
         */
```

```
        IPvXAddress getAddressFrom(InterfaceEntry *ie, int
addrType=ADDR_PREFER_IPv6);

        /**
         * The function tries to look up the InterfaceTable
module as submodule
         * <tt>"interfaceTable"</tt> or
<tt>"networkLayer.interfaceTable"</tt> within
         * the host/router module. Throws an error if not found.
         */
        InterfaceTable *interfaceTableOf(cModule *host);

        /**
         * The function tries to look up the RoutingTable module
as submodule
         * <tt>"routingTable"</tt> or
<tt>"networkLayer.routingTable"</tt> within
         * the host/router module. Throws an error if not found.
         */
        RoutingTable *routingTableOf(cModule *host);

    #ifdef WITH_IPv6
        /**
         * The function tries to look up the RoutingTable6 module
as submodule
         * <tt>"routingTable6"</tt> or
<tt>"networkLayer.routingTable6"</tt> within
         * the host/router module. Throws an error if not found.
         */
        RoutingTable6 *routingTable6Of(cModule *host);
    #endif

        /**
         * The function tries to look up the NotificationBoard
module as submodule
         * <tt>"notificationBoard"</tt> within the host/router
module. Throws an error
         * if not found.
         */
        NotificationBoard *notificationBoardOf(cModule *host);
```

```
        /**
         * Like interfaceTableOf(), but doesn't throw error if
not found.
         */
        InterfaceTable *findInterfaceTableOf(cModule *host);

        /**
         * Like routingTableOf(), but doesn't throw error if not
found.
         */
        RoutingTable *findRoutingTableOf(cModule *host);

    #ifdef WITH_IPv6
        /**
         * Like interfaceTableOf(), but doesn't throw error if
not found.
         */
        RoutingTable6 *findRoutingTable6Of(cModule *host);
    #endif

        /**
         * Like notificationBoardOf(), but doesn't throw error if
not found.
         */
        NotificationBoard *findNotificationBoardOf(cModule
*host);
        //@}
    };


    #endif
```

8.8. Arquivo IPAddressResolver.cc

```
    #include "IPAddressResolver.h"
    #include "InterfaceTable.h"
    #include "IPv4InterfaceData.h"
    #include "RoutingTable.h"
    #ifdef WITH_IPv6
    #include "IPv6InterfaceData.h"
```

```
    #include "RoutingTable6.h"
    #endif

    IPvXAddress IPAddressResolver::resolve(const char *s, int
addrType)
    {
        IPvXAddress addr;
        if (!tryResolve(s, addr, addrType))
            opp_error("IPAddressResolver: address `%s' not
configured (yet?)", s);
        return addr;
    }

    //insert by mroenick at 06/fev/08
    IPv6Address IPAddressResolver::resolveIPv6(const char *s, int
addrType)
    {
        IPv6Address addr;
        if (!tryResolveIPv6(s, addr, addrType))
            opp_error("IPAddressResolver: address `%s' not
configured (yet?)", s);
        return addr;
    }

    bool IPAddressResolver::tryResolveIPv6(const char *s,
IPv6Address& result, int addrType)
    {
        // empty address
        result = IPv6Address();
        if (!s || !*s)
            return true;

        // handle address literal
        if (result.tryParse(s))
            return true;

        // must be "modulename/interfacename(protocol)" syntax
then,
        // "/interfacename" and "(protocol)" being optional
        const char *slashp = strchr(s,'/');
        const char *leftparenp = strchr(s,'(');
```

```
        const char *rightparenp = strchr(s,')');
        const char *endp = s+strlen(s);

        // rudimentary syntax check
        if ((slashp && leftparenp && slashp>leftparenp) ||
            (leftparenp && !rightparenp) ||
            (!leftparenp && rightparenp) ||
            (rightparenp && rightparenp!=endp-1))
        {
            opp_error("IPAddressResolver: syntax error parsing
address spec `%s'", s);
        }

        // parse fields: modname, ifname, protocol
        std::string modname, ifname, protocol;
        modname.assign(s,
(slashp?slashp:leftparenp?leftparenp:endp)-s);
        if (slashp)
            ifname.assign(slashp+1, (leftparenp?leftparenp:endp)-
slashp-1);
        if (leftparenp)
            protocol.assign(leftparenp+1, rightparenp-leftparenp-
1);

        // find module and check protocol
        cModule *mod = simulation.moduleByPath(modname.c_str());
        if (!mod)
            opp_error("IPAddressResolver: module `%s' not found",
modname.c_str());
        if (!protocol.empty() && protocol!="ipv4" &&
protocol!="ipv6")
            opp_error("IPAddressResolver: error parsing address
spec `%s': address type must be `(ipv4)' or `(ipv6)'", s);
        if (!protocol.empty())
            addrType = protocol=="ipv4" ? ADDR_IPv4 : ADDR_IPv6;

        // get address from the given module/interface
        if (ifname.empty())
            result = IPv6AddressOf(mod, addrType);
        else
```

```
            result = IPv6AddressOf(mod, ifname.c_str(),
addrType);
        return !result.isUnspecified();
    }
    // and of change of file

    bool IPAddressResolver::tryResolve(const char *s,
IPvXAddress& result, int addrType)
    {
        // empty address
        result = IPvXAddress();
        if (!s || !*s)
            return true;

        // handle address literal
        if (result.tryParse(s))
            return true;

        // must be "modulename/interfacename(protocol)" syntax
then,
        // "/interfacename" and "(protocol)" being optional
        const char *slashp = strchr(s,'/');
        const char *leftparenp = strchr(s,'(');
        const char *rightparenp = strchr(s,')');
        const char *endp = s+strlen(s);

        // rudimentary syntax check
        if ((slashp && leftparenp && slashp>leftparenp) ||
            (leftparenp && !rightparenp) ||
            (!leftparenp && rightparenp) ||
            (rightparenp && rightparenp!=endp-1))
        {
            opp_error("IPAddressResolver: syntax error parsing
address spec `%s'", s);
        }

        // parse fields: modname, ifname, protocol
        std::string modname, ifname, protocol;
        modname.assign(s,
(slashp?slashp:leftparenp?leftparenp:endp)-s);
        if (slashp)
```

```
            ifname.assign(slashp+1, (leftparenp?leftparenp:endp)-
slashp-1);
        if (leftparenp)
            protocol.assign(leftparenp+1, rightparenp-leftparenp-
1);

        // find module and check protocol
        cModule *mod = simulation.moduleByPath(modname.c_str());
        if (!mod)
            opp_error("IPAddressResolver: module `%s' not found",
modname.c_str());
        if (!protocol.empty() && protocol!="ipv4" &&
protocol!="ipv6")
            opp_error("IPAddressResolver: error parsing address
spec `%s': address type must be `(ipv4)' or `(ipv6)'", s);
        if (!protocol.empty())
            addrType = protocol=="ipv4" ? ADDR_IPv4 : ADDR_IPv6;

        // get address from the given module/interface
        if (ifname.empty())
            result = addressOf(mod, addrType);
        else if (ifname == "routerId")
            result = IPvXAddress(routerIdOf(mod)); // addrType is
meaningless here, routerId is protocol independent
        else
            result = addressOf(mod, ifname.c_str(), addrType);
        return !result.isUnspecified();
    }

    IPAddress IPAddressResolver::routerIdOf(cModule *host)
    {
        RoutingTable *rt = routingTableOf(host);
        return rt->routerId();
    }

    IPvXAddress IPAddressResolver::addressOf(cModule *host, int
addrType)
    {
        InterfaceTable *ift = interfaceTableOf(host);
        return getAddressFrom(ift, addrType);
    }
```

```
    //insert by mroenick at 06/fev/08
    IPv6Address IPAddressResolver::IPv6AddressOf(cModule *host,
int addrType)
    {
        InterfaceTable *ift = interfaceTableOf(host);
        return getIPv6AddressFrom(ift);
    }

    IPvXAddress IPAddressResolver::addressOf(cModule *host, const
char *ifname, int addrType)
    {
        InterfaceTable *ift = interfaceTableOf(host);
        InterfaceEntry *ie = ift->interfaceByName(ifname);
        if (!ie)
            opp_error("IPAddressResolver: no interface called
`%s' in `%s'", ifname, ift->fullPath().c_str());
        return getAddressFrom(ie, addrType);
    }

    //insert by mroenick at 06/fev/08
    IPv6Address IPAddressResolver::IPv6AddressOf(cModule *host,
const char *ifname, int addrType)
    {
        InterfaceTable *ift = interfaceTableOf(host);
        InterfaceEntry *ie = ift->interfaceByName(ifname);
        if (!ie)
            opp_error("IPAddressResolver: no interface called
`%s' in `%s'", ifname, ift->fullPath().c_str());
        return getIPv6AddressFrom(ie, addrType);
    }

    IPvXAddress IPAddressResolver::getAddressFrom(InterfaceTable
*ift, int addrType)
    {
        IPvXAddress ret;
        if (addrType==ADDR_IPv6 || addrType==ADDR_PREFER_IPv6)
        {
            ret = getIPv6AddressFrom(ift);
            if (ret.isUnspecified() &&
addrType==ADDR_PREFER_IPv6)
```

```
                ret = getIPv4AddressFrom(ift);
        }
        else if (addrType==ADDR_IPv4 ||
addrType==ADDR_PREFER_IPv4)
        {
            ret = getIPv4AddressFrom(ift);
            if (ret.isUnspecified() &&
addrType==ADDR_PREFER_IPv4)
                ret = getIPv6AddressFrom(ift);
        }
        else
        {
            opp_error("IPAddressResolver: unknown addrType %d",
addrType);
        }
        return ret;
    }

    IPvXAddress IPAddressResolver::getAddressFrom(InterfaceEntry
*ie, int addrType)
    {
        IPvXAddress ret;
        if (addrType==ADDR_IPv6 || addrType==ADDR_PREFER_IPv6)
        {
            if (ie->ipv6())
                ret = getInterfaceIPv6Address(ie);
            if (ret.isUnspecified() && addrType==ADDR_PREFER_IPv6
&& ie->ipv4())
                ret = ie->ipv4()->inetAddress();
        }
        else if (addrType==ADDR_IPv4 ||
addrType==ADDR_PREFER_IPv4)
        {
            if (ie->ipv4())
                ret = ie->ipv4()->inetAddress();
            if (ret.isUnspecified() && addrType==ADDR_PREFER_IPv4
&& ie->ipv6())
                ret = getInterfaceIPv6Address(ie);
        }
        else
        {
```

```
            opp_error("IPAddressResolver: unknown addrType %d",
addrType);
        }
        return ret;
    }

    //insert by mroenick at 06/fev/08
    IPv6Address
IPAddressResolver::getIPv6AddressFrom(InterfaceEntry *ie, int
addrType)
    {
        IPv6Address ret;
        if (addrType==ADDR_IPv6 || addrType==ADDR_PREFER_IPv6)
        {
            if (ie->ipv6())
                ret = getInterfaceIPv6Address(ie);
        } else {
            opp_error("IPAddressResolver: no IPv6 addrType %d",
addrType);
        }
        return ret;
    }

    IPAddress
IPAddressResolver::getIPv4AddressFrom(InterfaceTable *ift)
    {
        IPAddress addr;
        if (ift->numInterfaces()==0)
            opp_error("IPAddressResolver: interface table `%s'
has no interface registered "
                      "(yet? try in a later init stage!)", ift-
>fullPath().c_str());

        // choose first usable interface address (configured for
IPv4, non-loopback if, addr non-null)
        for (int i=0; i<ift->numInterfaces(); i++)
        {
            InterfaceEntry *ie = ift->interfaceAt(i);
            if (ie->ipv4() && !ie->ipv4()-
>inetAddress().isUnspecified() && !ie->isLoopback())
            {
```

```
                addr = ie->ipv4()->inetAddress();
                break;
            }
        }
        return addr;
    }

    IPv6Address
IPAddressResolver::getIPv6AddressFrom(InterfaceTable *ift)
    {
    #ifdef WITH_IPv6
        // browse interfaces and pick a globally routable address
        if (ift->numInterfaces()==0)
            opp_error("IPAddressResolver: interface table `%s'
has no interface registered "
                      "(yet? try in a later init stage!)", ift-
>fullPath().c_str());

        IPv6Address addr;
        for (int i=0; i<ift->numInterfaces() &&
addr.isUnspecified(); i++)
        {
            InterfaceEntry *ie = ift->interfaceAt(i);
            if (!ie->ipv6() || ie->isLoopback())
                continue;
            IPv6Address ifAddr = ie->ipv6()->preferredAddress();
            if (addr.isGlobal() && ifAddr.isGlobal() &&
addr!=ifAddr)
                EV << ift->fullPath() << " has at least two
globally routable addresses on different interfaces\n";
            if (ifAddr.isGlobal())
                addr = ifAddr;
        }
        return addr;
    #else
        return IPv6Address();
    #endif
    }

    IPv6Address
IPAddressResolver::getInterfaceIPv6Address(InterfaceEntry *ie)
```

```
    {
    #ifdef WITH_IPv6
        if (!ie->ipv6())
            return IPv6Address();
        return ie->ipv6()->preferredAddress();
    #else
        return IPv6Address();
    #endif
    }

    InterfaceTable *IPAddressResolver::interfaceTableOf(cModule
*host)
    {
        // find InterfaceTable
        cModule *mod = host->submodule("interfaceTable");
        if (!mod)
            opp_error("IPAddressResolver: InterfaceTable not
found as submodule "
                      " `interfaceTable' in host/router `%s'",
host->fullPath().c_str());
        return check_and_cast<InterfaceTable *>(mod);
    }

    RoutingTable *IPAddressResolver::routingTableOf(cModule
*host)
    {
        // find RoutingTable
        cModule *mod = host->submodule("routingTable");
        if (!mod)
            opp_error("IPAddressResolver: RoutingTable not found
as submodule "
                      " `routingTable' in host/router `%s'",
host->fullPath().c_str());
        return check_and_cast<RoutingTable *>(mod);
    }

    #ifdef WITH_IPv6
    RoutingTable6 *IPAddressResolver::routingTable6Of(cModule
*host)
    {
        // find RoutingTable
```

```
        cModule *mod = host->submodule("routingTable6");
        if (!mod)
            opp_error("IPAddressResolver: RoutingTable6 not found
as submodule "
                      " `routingTable6' in host/router `%s'",
host->fullPath().c_str());
        return check_and_cast<RoutingTable6 *>(mod);
    }
    #endif

    NotificationBoard
*IPAddressResolver::notificationBoardOf(cModule *host)
    {
        // find NotificationBoard
        cModule *mod = host->submodule("notificationBoard");
        if (!mod)
            opp_error("IPAddressResolver: NotificationBoard not
found as submodule "
                      " notificationBoard' in host/router `%s'",
host->fullPath().c_str());
        return check_and_cast<NotificationBoard *>(mod);
    }

    InterfaceTable
*IPAddressResolver::findInterfaceTableOf(cModule *host)
    {
        cModule *mod = host->submodule("interfaceTable");
        return dynamic_cast<InterfaceTable *>(mod);
    }

    RoutingTable *IPAddressResolver::findRoutingTableOf(cModule
*host)
    {
        cModule *mod = host->submodule("routingTable");
        return dynamic_cast<RoutingTable *>(mod);
    }

    #ifdef WITH_IPv6
    RoutingTable6 *IPAddressResolver::findRoutingTable6Of(cModule
*host)
    {
```

```
        cModule *mod = host->submodule("routingTable6");
        return dynamic_cast<RoutingTable6 *>(mod);
    }
    #endif

    NotificationBoard
*IPAddressResolver::findNotificationBoardOf(cModule *host)
    {
        cModule *mod = host->submodule("notificationBoard");
        return dynamic_cast<NotificationBoard *>(mod);
    }
```

8.9. Arquivo IPv6NeighbourDiscovery.cc

```
    #include "IPv6NeighbourDiscovery.h"
    #include "NotifierConsts.h"

    #define MK_ASSIGN_LINKLOCAL_ADDRESS 0
    #define MK_SEND_PERIODIC_RTRADV 1
    #define MK_SEND_SOL_RTRADV 2
    #define MK_INITIATE_RTRDIS 3
    #define MK_DAD_TIMEOUT 4
    #define MK_RD_TIMEOUT 5
    #define MK_NUD_TIMEOUT 6
    #define MK_AR_TIMEOUT 7

    Define_Module(IPv6NeighbourDiscovery);


    IPv6NeighbourDiscovery::IPv6NeighbourDiscovery()
    {
    }

    IPv6NeighbourDiscovery::~IPv6NeighbourDiscovery()
    {
        // FIXME delete the following data structures,
cancelAndDelete timers in them etc.
        // Deleting the data structures my become unnecessary if
the lists store the
        // structs themselves and not pointers.
```

```
        //   RATimerList raTimerList;
        //   DADList dadList;
        //   RDList rdList;
        //   AdvIfList advIfList;
    }

    void IPv6NeighbourDiscovery::initialize(int stage)
    {
        // We have to wait until the 3rd stage (stage 2) with
scheduling messages,
        // because interface registration and IPv6 configuration
takes places
        // in the first two stages.
        if (stage==3)
        {
            ift = InterfaceTableAccess().get();
            rt6 = RoutingTable6Access().get();
            icmpv6 = ICMPv6Access().get();
            pendingQueue.setName("pendingQueue");
            nb = NotificationBoardAccess().get();
             nb->subscribe(this, NF_L2_ASSOCIATED);
       nb->subscribe(this, NF_L2_BEACON_LOST);


            for (int i=0; i < ift->numInterfaces(); i++)
            {
                InterfaceEntry *ie = ift->interfaceAt(i);

                if (ie->ipv6()->advSendAdvertisements() && !(ie-
>isLoopback()))
                {
                    createRATimer(ie);
                }
            }
            //This simulates random node bootup time. Link local
address assignment
            //takes place during this time.
            cMessage *msg = new cMessage("assignLinkLocalAddr",
MK_ASSIGN_LINKLOCAL_ADDRESS);
            //We want routers to boot up faster!
            if (rt6->isRouter())
```

```
                scheduleAt(uniform(0,0.3), msg);//Random Router
bootup time
            else
                scheduleAt(uniform(0.4,1), msg);//Random Host
bootup time
        }
    }

    void IPv6NeighbourDiscovery::receiveChangeNotification(int
category, cPolymorphic *details)
    {
        Enter_Method_Silent();
        printNotificationBanner(category, details);
        // start new IPv6 auto configuration
        if (category == NF_L2_ASSOCIATED) {
      int qtdeRoutes = rt6->numRoutes();
      for (int i=1; i<=qtdeRoutes ; i++)
      {
        rt6->removeRoute(rt6->route(qtdeRoutes - i));
      }
      rt6->purgeDestCache();
      InterfaceEntry *ie = ift->interfaceByName("wlan");
      if (ie->ipv6()->preferredAddress()!=ie->ipv6()-
>linkLocalAddress()) ie->ipv6()->removeAddress(ie->ipv6()-
>preferredAddress());
      createAndSendRSPacket(ie);
        }
        //clean cache for new IPv6 address
        if (category == NF_L2_BEACON_LOST) {
      //rt6->purgeDestCache();
      InterfaceEntry *ie = ift->interfaceByName("wlan");
      //if (ie->ipv6()->preferredAddress()!=ie->ipv6()-
>linkLocalAddress()) ie->ipv6()->removeAddress(ie->ipv6()-
>preferredAddress());
        }

    }

    void IPv6NeighbourDiscovery::handleMessage(cMessage *msg)
    {
        if (msg->isSelfMessage())
```

```
        {
            EV << "Self message received!\n";
            if (msg->kind()==MK_SEND_PERIODIC_RTRADV)
            {
                EV << "Sending periodic RA\n";
                sendPeriodicRA(msg);
            }
            else if (msg->kind()==MK_SEND_SOL_RTRADV)
            {
                EV << "Sending solicited RA\n";
                sendSolicitedRA(msg);
            }
            else if (msg->kind()==MK_ASSIGN_LINKLOCAL_ADDRESS)
            {
                EV << "Assigning Link Local Address\n";
                assignLinkLocalAddress(msg);
            }
            else if (msg->kind()==MK_DAD_TIMEOUT)
            {
                EV << "DAD Timeout message received\n";
                processDADTimeout(msg);
            }
            else if (msg->kind()==MK_RD_TIMEOUT)
            {
                EV << "Router Discovery message received\n";
                processRDTimeout(msg);
            }
            else if (msg->kind()==MK_INITIATE_RTRDIS)
            {
                EV << "initiate router discovery.\n";
                initiateRouterDiscovery(msg);
            }
            else if (msg->kind()==MK_NUD_TIMEOUT)
            {
                EV << "NUD Timeout message received\n";
                processNUDTimeout(msg);
            }
            else if (msg->kind()==MK_AR_TIMEOUT)
            {
                EV << "Address Resolution Timeout message
received\n";
```

```
                processARTimeout(msg);
            }
            else
                error("Unrecognized Timer");//stops sim w/ error
msg.
        }
        else if (dynamic_cast<ICMPv6Message *>(msg))
        {
            //This information will serve as input parameters to
various processors.
            IPv6ControlInfo *ctrlInfo
                = check_and_cast<IPv6ControlInfo*>(msg-
>removeControlInfo());
            ICMPv6Message *ndMsg = (ICMPv6Message *)msg;
            processNDMessage(ndMsg, ctrlInfo);
        }
        else if (dynamic_cast<IPv6Datagram *>(msg))// not ND
message
        {
            IPv6Datagram *datagram = (IPv6Datagram *)msg;
            processIPv6Datagram(datagram);
        }
        else
            error("Unknown message type received.\n");
    }

    void IPv6NeighbourDiscovery::processNDMessage(ICMPv6Message
*msg,
        IPv6ControlInfo *ctrlInfo)
    {

        if (dynamic_cast<IPv6RouterSolicitation *>(msg))
        {
            IPv6RouterSolicitation *rs = (IPv6RouterSolicitation
*)msg;
            processRSPacket(rs, ctrlInfo);
        }
        else if (dynamic_cast<IPv6RouterAdvertisement *>(msg))
        {
            IPv6RouterAdvertisement *ra =
(IPv6RouterAdvertisement *)msg;
```

```
            processRAPacket(ra, ctrlInfo);
        }
        else if (dynamic_cast<IPv6NeighbourSolicitation *>(msg))
        {
            IPv6NeighbourSolicitation *ns =
(IPv6NeighbourSolicitation *)msg;
            processNSPacket(ns, ctrlInfo);
        }
        else if (dynamic_cast<IPv6NeighbourAdvertisement *>(msg))
        {
            IPv6NeighbourAdvertisement *na =
(IPv6NeighbourAdvertisement *)msg;
            processNAPacket(na, ctrlInfo);
        }
        else if (dynamic_cast<IPv6Redirect *>(msg))
        {
            IPv6Redirect *redirect = (IPv6Redirect *)msg;
            processRedirectPacket(redirect, ctrlInfo);
        }
        else
        {
            error("Unrecognized ND message!");
        }
    }

    void IPv6NeighbourDiscovery::finish()
    {
    }

    void IPv6NeighbourDiscovery::processIPv6Datagram(IPv6Datagram
*msg)
    {
        EV << "Packet " << msg << " arrived from IPv6 module.\n";

        int nextHopIfID;
        EV << "Determining Next Hop" << endl;
        IPv6Address nextHopAddr = determineNextHop(msg-
>destAddress(), nextHopIfID);
        if (nextHopIfID == -1)
        {
```

```
            //draft-ietf-ipv6-2461bis-04 has omitted on-link
assumption.
            //draft-ietf-v6ops-onlinkassumption-03 explains why.
            icmpv6->sendErrorMessage(msg,
ICMPv6_DESTINATION_UNREACHABLE, NO_ROUTE_TO_DEST);
            return;
        }
        EV << "Next Hop Address is: " << nextHopAddr << " on
interface: " << nextHopIfID << endl;

        //RFC2461: Section 5.2 Conceptual Sending Algorithm
        //Once the IP address of the next-hop node is known, the
sender examines the
        //Neighbor Cache for link-layer information about that
neighbor.
        Neighbour *nce = neighbourCache.lookup(nextHopAddr,
nextHopIfID);

        if (nce==NULL)
        {
            //If no entry exists,
            EV << "No Entry exists in the Neighbour Cache.\n";

            //the sender creates one, sets its state to
INCOMPLETE,
            EV << "Creating an INCOMPLETE entry in the neighbour
cache.\n";
            nce = neighbourCache.addNeighbour(nextHopAddr,
nextHopIfID);

            //initiates Address Resolution,
            EV << "Initiating Address Resolution for:" <<
nextHopAddr
               << " on Interface:" << nextHopIfID << endl;
            initiateAddressResolution(msg->srcAddress(), nce);

            //and then queues the data packet pending completion
of address resolution.
            EV << "Add packet to entry's queue until Address
Resolution is complete.\n";
            nce->pendingPackets.push_back(msg);
```

```
            pendingQueue.insert(msg);
        }
        else if (nce->reachabilityState ==
IPv6NeighbourCache::INCOMPLETE)
        {
            EV << "Reachability State is INCOMPLETE.Address
Resolution already initiated.\n";
            bubble("Packet added to queue until Address
Resolution is complete.");
            nce->pendingPackets.push_back(msg);
            pendingQueue.insert(msg);
        }
        else if (nce->macAddress.isUnspecified())
        {
            EV << "NCE's MAC address is unspecified.\n";
            EV << "Initiate Address Resolution and add packet to
queue.\n";
            initiateAddressResolution(msg->srcAddress(), nce);
            nce->pendingPackets.push_back(msg);
            pendingQueue.insert(msg);
        }
        else if (nce->reachabilityState ==
IPv6NeighbourCache::STALE)
        {
            EV << "Reachability State is STALE.\n";
            send(msg,"toIPv6");
            initiateNeighbourUnreachabilityDetection(nce);
        }
        else if (nce->reachabilityState ==
IPv6NeighbourCache::REACHABLE)
        {
            EV << "Next hop is REACHABLE, sending packet to next-
hop address.";
            sendPacketToIPv6Module(msg, nextHopAddr, msg-
>srcAddress(), nextHopIfID);
        }
        else if (nce->reachabilityState ==
IPv6NeighbourCache::DELAY)//TODO: What if NCE is in PROBE state?
        {
            EV << "Next hop is in DELAY state, sending packet to
next-hop address.";
```

```
            sendPacketToIPv6Module(msg, nextHopAddr, msg-
>srcAddress(), nextHopIfID);
         }
         else
             error("Unknown Neighbour cache entry state.");
     }

     IPv6NeighbourDiscovery::AdvIfEntry
*IPv6NeighbourDiscovery::fetchAdvIfEntry(InterfaceEntry *ie)
     {
        for (AdvIfList::iterator it=advIfList.begin();
it!=advIfList.end(); it++)
        {
            AdvIfEntry *advIfEntry = (*it);
            if (advIfEntry->interfaceId == ie->interfaceId())
            {
                return advIfEntry;
            }
        }
        return NULL;
     }

     IPv6NeighbourDiscovery::RDEntry
*IPv6NeighbourDiscovery::fetchRDEntry(InterfaceEntry *ie)
     {
        for (RDList::iterator it=rdList.begin(); it!=rdList.end();
it++)
        {
            RDEntry *rdEntry = (*it);
            if (rdEntry->interfaceId == ie->interfaceId())
            {
                return rdEntry;
            }
        }
        return NULL;
     }

     const MACAddress&
IPv6NeighbourDiscovery::resolveNeighbour(const IPv6Address&
nextHop, int interfaceId)
     {
```

```
        Enter_Method("resolveNeighbor(%s,if=%d)",
nextHop.str().c_str(), interfaceId);

        Neighbour *nce = neighbourCache.lookup(nextHop,
interfaceId);
        //InterfaceEntry *ie = ift->interfaceAt(interfaceId);

        if (!nce || nce-
>reachabilityState==IPv6NeighbourCache::INCOMPLETE)
            return MACAddress::UNSPECIFIED_ADDRESS;
        else if (nce-
>reachabilityState==IPv6NeighbourCache::STALE)
            initiateNeighbourUnreachabilityDetection(nce);
        else if (nce-
>reachabilityState==IPv6NeighbourCache::REACHABLE &&
            simTime() > nce->reachabilityExpires)
        {
            nce->reachabilityState = IPv6NeighbourCache::STALE;
            initiateNeighbourUnreachabilityDetection(nce);
        }
        else if (nce-
>reachabilityState!=IPv6NeighbourCache::REACHABLE)
        {
            //reachability state must be either in DELAY or PROBE
            ASSERT(nce-
>reachabilityState==IPv6NeighbourCache::DELAY ||
                   nce-
>reachabilityState==IPv6NeighbourCache::PROBE);
            EV << "NUD in progress.\n";
        }
        //else the entry is REACHABLE and no further action is
required here.
        return nce->macAddress;
    }

    void IPv6NeighbourDiscovery::reachabilityConfirmed(const
IPv6Address& neighbour, int interfaceId)
    {
        Enter_Method("reachabilityConfirmed(%s,if=%d)",
neighbour.str().c_str(), interfaceId);
```

```
        //hmmm... this should only be invoked if a TCP ACK was
received and NUD is
        //currently being performed on the neighbour where the
TCP ACK was received from.

        Neighbour *nce = neighbourCache.lookup(neighbour,
interfaceId);

        cMessage *msg = nce->nudTimeoutEvent;
        if (msg != NULL)
        {
            EV << "NUD in progress. Cancelling NUD Timer\n";
            bubble("Reachability Confirmed via NUD.");
            cancelEvent(msg);
            delete msg;
        }
    IPv6Address IPv6NeighbourDiscovery::determineNextHop(
        const IPv6Address& destAddr, int& outIfID)
    {
        IPv6Address nextHopAddr;

        //RFC 2461 Section 5.2
        //Next-hop determination for a given unicast destination
operates as follows.

        //The sender performs a longest prefix match against the
Prefix List to
        //determine whether the packet's destination is on- or
off-link.
        EV << "Find out if supplied dest addr is on-link or off-
link.\n";
        const IPv6Route *route = rt6-
>doLongestPrefixMatch(destAddr);

        if (route != NULL)
        {
            //If the destination is on-link, the next-hop address
is the same as the
            //packet's destination address.
            if (route->nextHop().isUnspecified())
            {
```

```
                EV << "Dest is on-link, next-hop addr is same as
dest addr.\n";
                outIfID = route->interfaceID();
                nextHopAddr = destAddr;
            }
            else
            {
                EV << "A next-hop address was found with the
route, dest is off-link\n";
                EV << "Assume next-hop address as the selected
default router.\n";
                outIfID = route->interfaceID();
                nextHopAddr = route->nextHop();
            }
        }
        else
        {
            //Otherwise, the sender selects a router from the
Default Router List
            //(following the rules described in Section 6.3.6).

            EV << "No routes were found, Dest addr is off-
link.\n";
            nextHopAddr = selectDefaultRouter(outIfID);

            if (outIfID == -1) EV << "No Default Routers were
found.";
            else EV << "Default router found.\n";
        }

        /*the results of next-hop determination computations are
saved in the Destination
        Cache (which also contains updates learned from Redirect
messages).*/
        rt6->updateDestCache(destAddr, nextHopAddr, outIfID);
        return nextHopAddr;
    }

    void
IPv6NeighbourDiscovery::initiateNeighbourUnreachabilityDetection(
        Neighbour *nce)
```

```
    {
        ASSERT(nce-
>reachabilityState==IPv6NeighbourCache::STALE);
        const Key *nceKey = nce->nceKey;
        EV << "Initiating Neighbour Unreachability Detection";
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);
        EV << "Setting NCE state to DELAY.\n";
        /*The first time a node sends a packet to a neighbor
whose entry is
        STALE, the sender changes the state to DELAY*/
        nce->reachabilityState = IPv6NeighbourCache::DELAY;

        /*and sets a timer to expire in DELAY_FIRST_PROBE_TIME
seconds.*/
        cMessage *msg = new cMessage("NUDTimeout",
MK_NUD_TIMEOUT);
        msg->setContextPointer(nce);
        nce->nudTimeoutEvent = msg;
        scheduleAt(simTime()+ie->ipv6()->_delayFirstProbeTime(),
msg);
    }

    void IPv6NeighbourDiscovery::processNUDTimeout(cMessage
*timeoutMsg)
    {
        EV << "NUD has timed out\n";
        Neighbour *nce = (Neighbour *) timeoutMsg-
>contextPointer();
        const Key *nceKey = nce->nceKey;
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);

        if (nce->reachabilityState == IPv6NeighbourCache::DELAY)
        {
            /*If the entry is still in the DELAY state when the
timer expires, the
            entry's state changes to PROBE. If reachability
confirmation is received,
            the entry's state changes to REACHABLE.*/
            EV << "Neighbour Entry is still in DELAY state.\n";
```

```
            EV << "Entering PROBE state. Sending NS probe.\n";
            nce->reachabilityState = IPv6NeighbourCache::PROBE;
            nce->numProbesSent = 0;
        }

        /*If no response is received after waiting RetransTimer
milliseconds
        after sending the MAX_UNICAST_SOLICIT solicitations,
retransmissions cease
        and the entry SHOULD be deleted. Subsequent traffic to
that neighbor will
        recreate the entry and performs address resolution
again.*/
        if (nce->numProbesSent == ie->ipv6()-
>_maxUnicastSolicit())
        {
            EV << "Max number of probes have been sent." << endl;
            EV << "Neighbour is Unreachable, removing NCE." <<
endl;
            neighbourCache.remove(nceKey->address, nceKey-
>interfaceID);
            return;
        }

        /*Upon entering the PROBE state, a node sends a unicast
Neighbor Solicitation
        message to the neighbor using the cached link-layer
address.*/
        createAndSendNSPacket(nceKey->address, nceKey->address,
            ie->ipv6()->preferredAddress(), ie);
        nce->numProbesSent++;
        /*While in the PROBE state, a node retransmits Neighbor
Solicitation messages
        every RetransTimer milliseconds until reachability
confirmation is obtained.
        Probes are retransmitted even if no additional packets
are sent to the
        neighbor.*/
        scheduleAt(simTime()+ie->ipv6()->_retransTimer(),
timeoutMsg);
    }
```

```
    IPv6Address IPv6NeighbourDiscovery::selectDefaultRouter(int&
outIfID)
    {
        EV << "Selecting default router...\n";
        //draft-ietf-ipv6-2461bis-04.txt Section 6.3.6
        /*The algorithm for selecting a router depends in part on
whether or not a
        router is known to be reachable. The exact details of how
a node keeps track
        of a neighbor's reachability state are covered in Section
7.3.  The algorithm
        for selecting a default router is invoked during next-hop
determination when
        no Destination Cache entry exists for an off-link
destination or when
        communication through an existing router appears to be
failing.  Under normal
        conditions, a router would be selected the first time
traffic is sent to a
        destination, with subsequent traffic for that destination
using the same router
        as indicated in the Destination Cache modulo any changes
to the Destination
        Cache caused by Redirect messages.

        The policy for selecting routers from the Default Router
List is as
        follows:*/

        IPv6Address routerAddr;
        //Cycle through all entries in the neighbour cache entry.
        for(IPv6NeighbourCache::iterator
it=neighbourCache.begin(); it != neighbourCache.end(); it++)
        {
            Key key = it->first;
            Neighbour nce = it->second;
            bool routerExpired = false;
            if (nce.isDefaultRouter)
            {
                if (simTime()>nce.routerExpiryTime)
```

```
                {
                    EV << "Found an expired default router.
Deleting entry...\n";

neighbourCache.remove(key.address,key.interfaceID);
                    routerExpired = true;
                }
            }

            if (routerExpired == false)
            {
                if (nce.reachabilityState ==
IPv6NeighbourCache::REACHABLE ||
                    nce.reachabilityState ==
IPv6NeighbourCache::STALE ||
                    nce.reachabilityState ==
IPv6NeighbourCache::DELAY)//TODO: Need to improve this algorithm!
                {
                    EV << "Found a router in the neighbour
cache(default router list).\n";
                    outIfID = key.interfaceID;
                    if (routerExpired == false) return
key.address;
                }
            }
        }
        EV << "No suitable routers found.\n";

        /*1) Routers that are reachable or probably reachable
(i.e., in any state
        other than INCOMPLETE) SHOULD be preferred over routers
whose reachability
        is unknown or suspect (i.e., in the INCOMPLETE state, or
for which no Neighbor
        Cache entry exists). An implementation may choose to
always return the same
        router or cycle through the router list in a round-robin
fashion as long as
        it always returns a reachable or a probably reachable
router when one is
        available.*/
```

```
        /*2) When no routers on the list are known to be
reachable or probably
        reachable, routers SHOULD be selected in a round-robin
fashion, so that
        subsequent requests for a default router do not return
the same router until
        all other routers have been selected.

        Cycling through the router list in this case ensures that
all available
        routers are actively probed by the Neighbor
Unreachability Detection algorithm.
        A request for a default router is made in conjunction
with the sending of a
        packet to a router, and the selected router will be
probed for reachability
        as a side effect.*/

        outIfID = -1;//nothing found yet
        return IPv6Address();
    }

    void IPv6NeighbourDiscovery::timeoutPrefixEntry(const
IPv6Address& destPrefix,
        int prefixLength)//REDUNDANT
    {
        //RFC 2461: Section 6.3.5
        /*Whenever the invalidation timer expires for a Prefix
List entry, that
        entry is discarded.*/
        rt6->removeOnLinkPrefix(destPrefix, prefixLength);
        //hmmm... should the unicast address associated with this
prefix be deleted
        //as well?-TODO: The address should be timeout/deleted as
well!!

        /*No existing Destination Cache entries need be updated,
however. Should a
        reachability problem arise with an existing Neighbor
Cache entry, Neighbor
```

```
        Unreachability Detection will perform any needed
recovery.*/
    }

    void IPv6NeighbourDiscovery::timeoutDefaultRouter(const
IPv6Address& addr,
        int interfaceID)
    {
        //RFC 2461: Section 6.3.5
        /*Whenever the Lifetime of an entry in the Default Router
List expires,
        that entry is discarded.*/
        neighbourCache.remove(addr, interfaceID);

        /*When removing a router from the Default Router list,
the node MUST update
        the Destination Cache in such a way that all entries
using the router perform
        next-hop determination again rather than continue sending
traffic to the
        (deleted) router.*/
        rt6->purgeDestCacheEntriesToNeighbour(addr, interfaceID);
    }

    void IPv6NeighbourDiscovery::initiateAddressResolution(const
IPv6Address& dgSrcAddr,
        Neighbour *nce)
    {
        const Key *nceKey = nce->nceKey;
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);
        IPv6Address neighbourAddr = nceKey->address;
        int ifID = nceKey->interfaceID;

        //RFC2461: Section 7.2.2
        //When a node has a unicast packet to send to a neighbor,
but does not
        //know the neighbor's link-layer address, it performs
address
        //resolution.  For multicast-capable interfaces this
entails creating a
```

```
        //Neighbor Cache entry in the INCOMPLETE state(already
created if not done yet)
        //WEI-If entry already exists, we still have to ensure
that its state is INCOMPLETE.
        nce->reachabilityState = IPv6NeighbourCache::INCOMPLETE;

        //and transmitting a Neighbor Solicitation message
targeted at the
        //neighbor.  The solicitation is sent to the solicited-
node multicast
        //address "corresponding to"(or "derived from") the
target address.
        //(in this case, the target address is the address we are
trying to resolve)
        EV << "Preparing to send NS to solicited-node multicast
group\n";
        EV << "on the next hop interface\n";
        IPv6Address nsDestAddr =
neighbourAddr.formSolicitedNodeMulticastAddress();//for NS
datagram
        IPv6Address nsTargetAddr = neighbourAddr;//for the field
within the NS
        IPv6Address nsSrcAddr;

        /*If the source address of the packet prompting the
solicitation is the
        same as one of the addresses assigned to the outgoing
interface,*/
        if (ie->ipv6()->hasAddress(dgSrcAddr))
            /*that address SHOULD be placed in the IP Source
Address of the outgoing
            solicitation.*/
            nsSrcAddr = dgSrcAddr;
        else
            /*Otherwise, any one of the addresses assigned to the
interface
            should be used.*/
            nsSrcAddr = ie->ipv6()->preferredAddress();
        ASSERT(ifID != -1);
        //Sending NS on specified interface.
```

```
        createAndSendNSPacket(nsTargetAddr, nsDestAddr,
nsSrcAddr, ie);
        nce->numOfARNSSent = 1;
        nce->nsSrcAddr = nsSrcAddr;

        /*While awaiting a response, the sender SHOULD retransmit
Neighbor Solicitation
        messages approximately every RetransTimer milliseconds,
even in the absence
        of additional traffic to the neighbor. Retransmissions
MUST be rate-limited
        to at most one solicitation per neighbor every
RetransTimer milliseconds.*/
        cMessage *msg = new cMessage("arTimeout",
MK_AR_TIMEOUT);//AR msg timer
        nce->arTimer = msg;
        msg->setContextPointer(nce);
        scheduleAt(simTime()+ie->ipv6()->_retransTimer(), msg);
    }


    void IPv6NeighbourDiscovery::processARTimeout(cMessage
*arTimeoutMsg)
    {
        //AR timeouts are cancelled when a valid solicited NA is
received.
        Neighbour *nce = (Neighbour *)arTimeoutMsg-
>contextPointer();
        const Key *nceKey = nce->nceKey;
        IPv6Address nsTargetAddr = nceKey->address;
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);
        EV << "Num Of NS Sent:" << nce->numOfARNSSent << endl;
        EV << "Max Multicast Solicitation:" << ie->ipv6()-
>_maxMulticastSolicit() << endl;
        if (nce->numOfARNSSent < ie->ipv6()-
>_maxMulticastSolicit())
        {
            EV << "Sending another Address Resolution NS message"
<< endl;
```

```
            IPv6Address nsDestAddr =
nsTargetAddr.formSolicitedNodeMulticastAddress();
            createAndSendNSPacket(nsTargetAddr, nsDestAddr, nce-
>nsSrcAddr, ie);
            nce->numOfARNSSent++;
            scheduleAt(simTime()+ie->ipv6()->_retransTimer(),
arTimeoutMsg);
            return;
        }
        EV << "Address Resolution has failed." << endl;
        dropQueuedPacketsAwaitingAR(nce);
        EV << "Deleting AR timeout msg\n";
        delete arTimeoutMsg;
    }

    void
IPv6NeighbourDiscovery::dropQueuedPacketsAwaitingAR(Neighbour
*nce)
    {
        const Key *nceKey = nce->nceKey;
        //RFC 2461: Section 7.2.2
        /*If no Neighbor Advertisement is received after
MAX_MULTICAST_SOLICIT
        solicitations, address resolution has failed. The sender
MUST return ICMP
        destination unreachable indications with code 3 (Address
Unreachable) for
        each packet queued awaiting address resolution.*/
        MsgPtrVector pendingPackets = nce->pendingPackets;
        EV << "Pending Packets empty:" << pendingPackets.empty()
<< endl;
        while (!pendingPackets.empty())
        {
            MsgPtrVector::iterator i = pendingPackets.begin();
            cMessage *msg = (*i);
            IPv6Datagram *ipv6Msg = (IPv6Datagram *)msg;
            //Assume msg is the packet itself. I need the
datagram's source addr.
            //The datagram's src addr will be the destination of
the unreachable msg.
```

```
            EV << "Sending ICMP unreachable destination." <<
endl;
            pendingPackets.erase(i);
            pendingQueue.remove(msg);
            icmpv6->sendErrorMessage(ipv6Msg,
ICMPv6_DESTINATION_UNREACHABLE, ADDRESS_UNREACHABLE);
        }

        //RFC 2461: Section 7.3.3
        /*If address resolution fails, the entry SHOULD be
deleted, so that subsequent
        traffic to that neighbor invokes the next-hop
determination procedure again.*/
        EV << "Removing neighbour cache entry" << endl;
        neighbourCache.remove(nceKey->address, nceKey-
>interfaceID);
    }

    void IPv6NeighbourDiscovery::sendPacketToIPv6Module(cMessage
*msg, const IPv6Address& destAddr,
        const IPv6Address& srcAddr, int interfaceId)
    {
        IPv6ControlInfo *controlInfo = new IPv6ControlInfo();
        controlInfo->setProtocol(IP_PROT_IPv6_ICMP);
        controlInfo->setDestAddr(destAddr);
        controlInfo->setSrcAddr(srcAddr);
        controlInfo->setHopLimit(255);
        controlInfo->setInterfaceId(interfaceId);
        msg->setControlInfo(controlInfo);

        send(msg,"toIPv6");
    }

    /**Not used yet-unsure if we really need it. --DELETED,
Andras*/

    void
IPv6NeighbourDiscovery::sendQueuedPacketsToIPv6Module(Neighbour
*nce)
    {
        MsgPtrVector& pendingPackets = nce->pendingPackets;
```

```
        while(!pendingPackets.empty())//FIXME: pendingPackets are
always empty!!!!
        {
            MsgPtrVector::iterator i = pendingPackets.begin();
            cMessage *msg = (*i);
            pendingPackets.erase(i);
            pendingQueue.remove(msg);
            EV << "Sending queued packet " << msg << endl;
            send(msg,"toIPv6");
        }
    }

    void IPv6NeighbourDiscovery::assignLinkLocalAddress(cMessage
*timerMsg)
    {
        //Node has booted up. Start assigning a link-local
address for each
        //interface in this node.
        for (int i=0; i < ift->numInterfaces(); i++)
    {
            InterfaceEntry *ie = ift->interfaceAt(i);

            //Skip the loopback interface.
            if (ie->isLoopback()) continue;

        IPv6Address linkLocalAddr = ie->ipv6()-
>linkLocalAddress();
        if (linkLocalAddr.isUnspecified())
        {
            //if no link local address exists for this interface,
we assign one to it.
            EV << "No link local address exists. Forming one" <<
endl;
            linkLocalAddr =
IPv6Address().formLinkLocalAddress(ie->interfaceToken());
            ie->ipv6()->assignAddress(linkLocalAddr, true, 0, 0);
        }

            //Before we can use this address, we MUST initiate
DAD first.
            initiateDAD(linkLocalAddr, ie);
```

```
        }
        delete timerMsg;
    }

    void IPv6NeighbourDiscovery::initiateDAD(const IPv6Address&
tentativeAddr,
        InterfaceEntry *ie)
    {
        DADEntry *dadEntry = new DADEntry();
        dadEntry->interfaceId = ie->interfaceId();
        dadEntry->address = tentativeAddr;
        dadEntry->numNSSent = 0;
        dadList.insert(dadEntry);
        /*
        RFC2462: Section 5.4.2
        To check an address, a node sends DupAddrDetectTransmits
Neighbor
        Solicitations, each separated by RetransTimer
milliseconds. The
        solicitation's Target Address is set to the address being
checked,
        the IP source is set to the unspecified address and the
IP
        destination is set to the solicited-node multicast
address of the
        target address.*/
        IPv6Address destAddr =
tentativeAddr.formSolicitedNodeMulticastAddress();
        //Send a NS
        createAndSendNSPacket(tentativeAddr, destAddr,
            IPv6Address::UNSPECIFIED_ADDRESS, ie);
        dadEntry->numNSSent++;

        cMessage *msg = new cMessage("dadTimeout",
MK_DAD_TIMEOUT);
        msg->setContextPointer(dadEntry);
        scheduleAt(simTime()+ie->ipv6()->retransTimer(), msg);
    }

    void IPv6NeighbourDiscovery::processDADTimeout(cMessage *msg)
    {
```

```
        DADEntry *dadEntry = (DADEntry *)msg->contextPointer();
        InterfaceEntry *ie = (InterfaceEntry *)ift-
>interfaceAt(dadEntry->interfaceId);
        IPv6Address tentativeAddr = dadEntry->address;
        //Here, we need to check how many DAD messages for the
interface entry were
        //sent vs. DupAddrDetectTransmits
        EV << "numOfDADMessagesSent is: " << dadEntry->numNSSent
<< endl;
        EV << "dupAddrDetectTrans is: " << ie->ipv6()-
>dupAddrDetectTransmits() << endl;
        if (dadEntry->numNSSent < ie->ipv6()-
>dupAddrDetectTransmits())
        {
            bubble("Sending another DAD NS message.");
            IPv6Address destAddr =
tentativeAddr.formSolicitedNodeMulticastAddress();
            createAndSendNSPacket(dadEntry->address, destAddr,
IPv6Address::UNSPECIFIED_ADDRESS, ie);
            dadEntry->numNSSent++;
            //Reuse the received msg
            scheduleAt(simTime()+ie->ipv6()->retransTimer(),
msg);
        }
        else
        {
            bubble("Max number of DAD messages for interface
sent. Address is unique.");
            ie->ipv6()->permanentlyAssign(tentativeAddr);
            dadList.erase(dadEntry);
            EV << "delete dadEntry and msg\n";
            delete dadEntry;
            delete msg;
            /*RFC 2461: Section 6.3.7 2nd Paragraph
            Before a host sends an initial solicitation, it
SHOULD delay the
            transmission for a random amount of time between 0
and
            MAX_RTR_SOLICITATION_DELAY.  This serves to alleviate
congestion when
```

```
            many hosts start up on a link at the same time, such
as might happen
            after recovery from a power failure.*/
            //TODO: Placing these operations here means fast
router solicitation is
            //not adopted. Will relocate.
            if (ie->ipv6()->advSendAdvertisements() == false)
            {
                EV << "creating router discovery message
timer\n";
                cMessage *rtrDisMsg = new
cMessage("initiateRTRDIS",MK_INITIATE_RTRDIS);
                rtrDisMsg->setContextPointer(ie);
                simtime_t interval = uniform(0,ie->ipv6()-
>_maxRtrSolicitationDelay());//random delay
                scheduleAt(simTime()+interval, rtrDisMsg);
            }
        }
    }

    IPv6RouterSolicitation
*IPv6NeighbourDiscovery::createAndSendRSPacket(InterfaceEntry *ie)
    {
        ASSERT(ie->ipv6()->advSendAdvertisements() == false);
        //RFC 2461: Section 6.3.7 Sending Router Solicitations
        //A host sends Router Solicitations to the All-Routers
multicast address. The
        //IP source address is set to either one of the
interface's unicast addresses
        //or the unspecified address.
        IPv6Address myIPv6Address = ie->ipv6()-
>preferredAddress();
        if (myIPv6Address.isUnspecified())
            myIPv6Address = ie->ipv6()->linkLocalAddress();//so
we use the link local address instead
        if (ie->ipv6()->isTentativeAddress(myIPv6Address))
            myIPv6Address =
IPv6Address::UNSPECIFIED_ADDRESS;//set my IPv6 address to
unspecified.
        IPv6Address destAddr =
IPv6Address::ALL_ROUTERS_2;//all_routers multicast
```

```
        IPv6RouterSolicitation *rs = new
IPv6RouterSolicitation("RSpacket");
        rs->setType(ICMPv6_ROUTER_SOL);

        //The Source Link-Layer Address option SHOULD be set to
the host's link-layer
        //address, if the IP source address is not the
unspecified address.
        if (!myIPv6Address.isUnspecified())
            rs->setSourceLinkLayerAddress(ie->macAddress());

        //Construct a Router Solicitation message
        sendPacketToIPv6Module(rs, destAddr, myIPv6Address, ie-
>interfaceId());
        return rs;
    }

    void IPv6NeighbourDiscovery::initiateRouterDiscovery(cMessage
*msg)
    {
        EV << "Initiating Router Discovery" << endl;
        InterfaceEntry *ie = (InterfaceEntry *)msg-
>contextPointer();
        delete msg;
        //RFC2461: Section 6.3.7
        /*When an interface becomes enabled, a host may be
unwilling to wait for the
        next unsolicited Router Advertisement to locate default
routers or learn
        prefixes.  To obtain Router Advertisements quickly, a
host SHOULD transmit up
        to MAX_RTR_SOLICITATIONS Router Solicitation messages
each separated by at
        least RTR_SOLICITATION_INTERVAL seconds.(FIXME:Therefore
this should be invoked
        at the beginning of the simulation-WEI)*/
        RDEntry *rdEntry = new RDEntry();
        rdEntry->interfaceId = ie->interfaceId();
        rdEntry->numRSSent = 0;
        createAndSendRSPacket(ie);
        rdEntry->numRSSent++;
```

```
        //Create and schedule a message for retransmission to
this module
        cMessage *rdTimeoutMsg = new cMessage("processRDTimeout",
MK_RD_TIMEOUT);
        rdTimeoutMsg->setContextPointer(ie);
        rdEntry->timeoutMsg = rdTimeoutMsg;
        rdList.insert(rdEntry);
        /*Before a host sends an initial solicitation, it SHOULD
delay the
        transmission for a random amount of time between 0 and
        MAX_RTR_SOLICITATION_DELAY.  This serves to alleviate
congestion when
        many hosts start up on a link at the same time, such as
might happen
        after recovery from a power failure.  If a host has
already performed
        a random delay since the interface became (re)enabled
(e.g., as part
        of Duplicate Address Detection [ADDRCONF]) there is no
need to delay
        again before sending the first Router Solicitation
message.*/
        //simtime_t rndInterval = uniform(0, ie->ipv6()-
>_maxRtrSolicitationDelay());
        scheduleAt(simTime()+ie->ipv6()-
>_rtrSolicitationInterval(), rdTimeoutMsg);
    }

    void
IPv6NeighbourDiscovery::cancelRouterDiscovery(InterfaceEntry *ie)
    {
        //Next we retrieve the rdEntry with the Interface Entry.
        RDEntry *rdEntry = fetchRDEntry(ie);
        if (rdEntry != NULL)
        {
            EV << "rdEntry is not NULL, RD cancelled!" << endl;
            cancelEvent(rdEntry->timeoutMsg);
            rdList.erase(rdEntry);
            delete rdEntry;
        }
```

```
        else
            EV << "rdEntry is NULL, not cancelling RD!" << endl;
    }

    void IPv6NeighbourDiscovery::processRDTimeout(cMessage *msg)
    {
        InterfaceEntry *ie = (InterfaceEntry *)msg-
>contextPointer();
        RDEntry *rdEntry = fetchRDEntry(ie);
        if (rdEntry->numRSSent < ie->ipv6()-
>_maxRtrSolicitations())
        {
            bubble("Sending another RS message.");
            createAndSendRSPacket(ie);
            rdEntry->numRSSent++;
            //Need to find out if this is the last RS we are
sending out.
            if (rdEntry->numRSSent == ie->ipv6()-
>_maxRtrSolicitations())
                scheduleAt(simTime()+ie->ipv6()-
>_maxRtrSolicitationDelay(), msg);
            else
                scheduleAt(simTime()+ie->ipv6()-
>_rtrSolicitationInterval(), msg);
        }
        else
        {
            //RFC 2461, Section 6.3.7
            /*If a host sends MAX_RTR_SOLICITATIONS
solicitations, and receives no Router
            Advertisements after having waited
MAX_RTR_SOLICITATION_DELAY seconds after
            sending the last solicitation, the host concludes
that there are no routers
            on the link for the purpose of [ADDRCONF]. However,
the host continues to
            receive and process Router Advertisements messages in
the event that routers
            appear on the link.*/
            bubble("Max number of RS messages sent");
```

```
            EV << "No RA messages were received. Assume no
routers are on-link";
            delete rdEntry;
            delete msg;
        }
    }

    void
IPv6NeighbourDiscovery::processRSPacket(IPv6RouterSolicitation
*rs,
        IPv6ControlInfo *rsCtrlInfo)
    {
        if (validateRSPacket(rs, rsCtrlInfo) == false) return;
        //Find out which interface the RS message arrived on.
        InterfaceEntry *ie = ift->interfaceAt(rsCtrlInfo-
>interfaceId());
        AdvIfEntry *advIfEntry = fetchAdvIfEntry(ie);//fetch
advertising interface entry.

        //RFC 2461: Section 6.2.6
        //A host MUST silently discard any received Router
Solicitation messages.
        if (ie->ipv6()->advSendAdvertisements())
        {
            EV << "This is an advertising interface, processing
RS\n";

            if (validateRSPacket(rs, rsCtrlInfo) == false)
return;
            EV << "RS message validated\n";

            //First we extract RS specific information from the
received message
            MACAddress macAddr = rs->sourceLinkLayerAddress();
            EV << "MAC Address extracted\n";
            delete rs;

            /*A router MAY choose to unicast the response
directly to the soliciting
            host's address (if the solicitation's source address
is not the unspecified
```

```
            address), but the usual case is to multicast the
response to the
            all-nodes group. In the latter case, the interface's
interval timer is
            reset to a new random value, as if an unsolicited
advertisement had just
            been sent(see Section 6.2.4).*/

            /*In all cases, Router Advertisements sent in
response to a Router
            Solicitation MUST be delayed by a random time between
0 and
            MAX_RA_DELAY_TIME seconds. (If a single advertisement
is sent in
            response to multiple solicitations, the delay is
relative to the
            first solicitation.)  In addition, consecutive Router
Advertisements
            sent to the all-nodes multicast address MUST be rate
limited to no
            more than one advertisement every
MIN_DELAY_BETWEEN_RAS seconds.*/

            /*A router might process Router Solicitations as
follows:
            - Upon receipt of a Router Solicitation, compute a
random delay
            within the range 0 through MAX_RA_DELAY_TIME. If the
computed
            value corresponds to a time later than the time the
next multicast
            Router Advertisement is scheduled to be sent, ignore
the random
            delay and send the advertisement at the already-
scheduled time.*/
            cMessage *msg = new cMessage("sendSolicitedRA",
MK_SEND_SOL_RTRADV);
            msg->setContextPointer(ie);
            simtime_t interval = uniform(0,ie->ipv6()-
>_maxRADelayTime());
```

```
            if (interval < advIfEntry->nextScheduledRATime)
            {
                simtime_t nextScheduledTime;
                nextScheduledTime = simTime()+interval;
                scheduleAt(nextScheduledTime, msg);
                advIfEntry->nextScheduledRATime =
nextScheduledTime;
            }
            //else we ignore the generate interval and send it at
the next scheduled time.

            //We need to keep a log here each time an RA is sent.
Not implemented yet.
            //Assume the first course of action.
            /*- If the router sent a multicast Router
Advertisement (solicited or
            unsolicited) within the last MIN_DELAY_BETWEEN_RAS
seconds,
            schedule the advertisement to be sent at a time
corresponding to
            MIN_DELAY_BETWEEN_RAS plus the random value after the
previous
            advertisement was sent. This ensures that the
multicast Router
            Advertisements are rate limited.

            - Otherwise, schedule the sending of a Router
Advertisement at the
            time given by the random value.*/
        }
        else
        {
            EV << "This interface is a host, discarding RA
message\n";
            delete rs;
        }
    }

    bool
IPv6NeighbourDiscovery::validateRSPacket(IPv6RouterSolicitation
*rs,
```

```
        IPv6ControlInfo *rsCtrlInfo)
    {
        bool result = true;
        /*6.1.1.  Validation of Router Solicitation Messages
        A router MUST silently discard any received Router
Solicitation
        messages that do not satisfy all of the following
validity checks:

        - The IP Hop Limit field has a value of 255, i.e., the
packet
        could not possibly have been forwarded by a router.*/
        if (rsCtrlInfo->hopLimit() != 255)
        {
            EV << "Hop limit is not 255! RS validation
failed!\n";
            result = false;
        }
        //- ICMP Code is 0.
        if (rsCtrlInfo->protocol() != IP_PROT_IPv6_ICMP)
        {
            EV << "ICMP Code is not 0! RS validation failed!\n";
            result = false;
        }
        //- If the IP source address is the unspecified address,
there is no
        //source link-layer address option in the message.
        if (rsCtrlInfo->srcAddr().isUnspecified())
        {
            EV << "IP source address is unspecified\n";
            if (rs->sourceLinkLayerAddress().isUnspecified() ==
false)
            {
                EV << " but source link layer address is
provided. RS validation failed!\n";
            }
        }
        return result;
    }
```

```
    IPv6RouterAdvertisement
*IPv6NeighbourDiscovery::createAndSendRAPacket(
        const IPv6Address& destAddr, InterfaceEntry *ie)
    {
        EV << "Create and send RA invoked!\n";
        //Must use link-local addr. See: RFC2461 Section 6.1.2
        IPv6Address sourceAddr = ie->ipv6()->linkLocalAddress();

        //This operation includes all options, regardless whether
it is solicited or unsolicited.
        if (ie->ipv6()->advSendAdvertisements()) //if this is an
advertising interface
        {
            //Construct a Router Advertisment message
            IPv6RouterAdvertisement *ra = new
IPv6RouterAdvertisement("RApacket");
            ra->setType(ICMPv6_ROUTER_AD);

            //RFC 2461: Section 6.2.3 Router Advertisment Message
Content
            /*A router sends periodic as well as solicited Router
Advertisements out
            its advertising interfaces.  Outgoing Router
Advertisements are filled
            with the following values consistent with the message
format given in
            Section 4.2:*/

            //- In the Router Lifetime field: the interface's
configured AdvDefaultLifetime.
            ra->setRouterLifetime(ie->ipv6()-
>advDefaultLifetime());

            //- In the M and O flags: the interface's configured
AdvManagedFlag and
            //AdvOtherConfigFlag, respectively.  See [ADDRCONF].
            ra->setManagedAddrConfFlag(ie->ipv6()-
>advManagedFlag());
            ra->setOtherStatefulConfFlag(ie->ipv6()-
>advOtherConfigFlag());
```

```
            //- In the Cur Hop Limit field: the interface's
configured CurHopLimit.
            ra->setCurHopLimit(ie->ipv6()->advCurHopLimit());

            //- In the Reachable Time field: the interface's
configured AdvReachableTime.
            ra->setReachableTime(ie->ipv6()->advReachableTime());

            //- In the Retrans Timer field: the interface's
configured AdvRetransTimer.
            ra->setRetransTimer(ie->ipv6()->advRetransTimer());

            //- In the options:
            /*o Source Link-Layer Address option: link-layer
address of the sending
                interface.  (Assumption: We always send this)*/
            ra->setSourceLinkLayerAddress(ie->macAddress());
            ra->setMTU(ie->ipv6()->advLinkMTU());

            //Add all Advertising Prefixes to the RA
            int numAdvPrefixes = ie->ipv6()->numAdvPrefixes();
            EV << "Number of Adv Prefixes: " << numAdvPrefixes <<
endl;
            ra->setPrefixInformationArraySize(numAdvPrefixes);
            for (int i = 0; i < numAdvPrefixes; i++)
            {
                IPv6InterfaceData::AdvPrefix advPrefix = ie-
>ipv6()->advPrefix(i);
                IPv6NDPrefixInformation prefixInfo;
                prefixInfo.setPrefix(advPrefix.prefix);

prefixInfo.setPrefixLength(advPrefix.prefixLength);

                //- In the "on-link" flag: the entry's
AdvOnLinkFlag.

prefixInfo.setOnlinkFlag(advPrefix.advOnLinkFlag);
                //- In the Valid Lifetime field: the entry's
AdvValidLifetime.

prefixInfo.setValidLifetime(advPrefix.advValidLifetime);
```

```
                //- In the "Autonomous address configuration"
flag: the entry's
                //AdvAutonomousFlag.

prefixInfo.setAutoAddressConfFlag(advPrefix.advAutonomousFlag);
                //- In the Preferred Lifetime field: the entry's
AdvPreferredLifetime.

prefixInfo.setPreferredLifetime(advPrefix.advPreferredLifetime);
                //Now we pop the prefix info into the RA.
                ra->setPrefixInformation(i, prefixInfo);
            }
            sendPacketToIPv6Module(ra, destAddr, sourceAddr, ie-
>interfaceId());
            return ra;
        }
    }

    void
IPv6NeighbourDiscovery::processRAPacket(IPv6RouterAdvertisement
*ra,
        IPv6ControlInfo *raCtrlInfo)
    {
        InterfaceEntry *ie = ift->interfaceAt(raCtrlInfo-
>interfaceId());

        if (ie->ipv6()->advSendAdvertisements())
        {
            EV << "Interface is an advertising interface,
dropping RA message.\n";
            delete ra;
            return;
        }
        else
        {
            if (validateRAPacket(ra, raCtrlInfo) == false)
            {
                delete ra;
                return;
            }
```

```
            cancelRouterDiscovery(ie);//Cancel router discovery
if it is in progress.
            EV << "Interface is a host, processing RA.\n";

            processRAForRouterUpdates(ra, raCtrlInfo);//See
RFC2461: Section 6.3.4

            //Possible options
            MACAddress macAddress = ra->sourceLinkLayerAddress();
            uint mtu = ra->MTU();
            for (int i = 0; i < ra->prefixInformationArraySize();
i++)
            {
                IPv6NDPrefixInformation& prefixInfo = ra-
>prefixInformation(i);
                if (prefixInfo.autoAddressConfFlag() == true)//If
auto addr conf is set

processRAPrefixInfoForAddrAutoConf(prefixInfo, ie);//We process
prefix Info and form an addr
            }
        }
        delete raCtrlInfo;
        delete ra;
    }

    void
IPv6NeighbourDiscovery::processRAForRouterUpdates(IPv6RouterAdvert
isement *ra,
        IPv6ControlInfo *raCtrlInfo)
    {
        EV << "Processing RA for Router Updates\n";
        //RFC2461: Section 6.3.4
        //Paragraphs 1 and 2 omitted.

        //On receipt of a valid Router Advertisement, a host
extracts the source
        //address of the packet and does the following:
        IPv6Address raSrcAddr = raCtrlInfo->srcAddr();
        InterfaceEntry *ie = ift->interfaceAt(raCtrlInfo-
>interfaceId());
```

```
        int ifID = ie->interfaceId();

        /*- If the address is not already present in the host's
Default Router List,
        and the advertisement's Router Lifetime is non-zero,
create a new entry in
        the list, and initialize its invalidation timer value
from the advertisement's
        Router Lifetime field.*/
        Neighbour *neighbour = neighbourCache.lookup(raSrcAddr,
ifID);
        if (neighbour == NULL)
        {
            EV << "Neighbour Cache Entry does not contain RA's
source address\n";
            if (ra->routerLifetime() != 0)
            {
                EV << "RA's router lifetime is non-zero, creating
an entry in the "
                   << "Host's default router list.\n" << ra-
>routerLifetime();
                //If a Neighbor Cache entry is created for the
router its reachability
                //state MUST be set to STALE as specified in
Section 7.3.3.
                if (ra->sourceLinkLayerAddress().isUnspecified())
                {
                    neighbour =
neighbourCache.addRouter(raSrcAddr, ifID,
                        simTime()+ra->routerLifetime());
                    //Note:invalidation timers are not explicitly
defined.
                }
                else
                {
                    neighbour =
neighbourCache.addRouter(raSrcAddr, ifID,
                        ra->sourceLinkLayerAddress(),
simTime()+ra->routerLifetime());
                    //According to Greg, we should add a default
route for hosts as well!
```

```
                    rt6->addDefaultRoute(raSrcAddr, ifID,
simTime()+ra->routerLifetime());
                }
            }
            else
            {
                EV << "Router Lifetime is 0, adding NON-default
router.\n";
                //WEI-The router is advertising itself, BUT not
as a default router.
                if (ra->sourceLinkLayerAddress().isUnspecified())
                    neighbour =
neighbourCache.addNeighbour(raSrcAddr, ifID);
                else
                    neighbour =
neighbourCache.addNeighbour(raSrcAddr, ifID,
                        ra->sourceLinkLayerAddress());
                neighbour->isRouter = true;
            }
        }
        else
        {
            //If no Source Link-Layer Address is included, but a
corresponding Neighbor
            //Cache entry exists, its IsRouter flag MUST be set
to TRUE.
            neighbour->isRouter = true;

            //If a cache entry already exists and is updated with
a different link-
            //layer address the reachability state MUST also be
set to STALE.
            if (ra->sourceLinkLayerAddress().isUnspecified() ==
false &&
                neighbour->macAddress.equals(ra-
>sourceLinkLayerAddress()) == false)
                neighbour->macAddress = ra-
>sourceLinkLayerAddress();

            /*- If the address is already present in the host's
Default Router List
```

```
            as a result of a previously-received advertisement,
reset its invalidation
            timer to the Router Lifetime value in the newly-
received advertisement.*/
            neighbour->routerExpiryTime = simTime()+ra-
>routerLifetime();

            /*- If the address is already present in the host's
Default Router List
            and the received Router Lifetime value is zero,
immediately time-out the
            entry as specified in Section 6.3.5.*/
            if (ra->routerLifetime() == 0)
            {
                EV << "RA's router lifetime is ZERO. Timing-out
entry.\n";
                timeoutDefaultRouter(raSrcAddr, ifID);
            }
        }

        //Paragraph Omitted.

        //If the received Cur Hop Limit value is non-zero the
host SHOULD set
        //its CurHopLimit variable to the received value.
        if (ra->curHopLimit() != 0)
        {
            EV << "RA's Cur Hop Limit is non-zero. Setting host's
Cur Hop Limit to "
               << "received value.\n";
            ie->ipv6()->setCurHopLimit(ra->curHopLimit());
        }

        //If the received Reachable Time value is non-zero the
host SHOULD set its
        //BaseReachableTime variable to the received value.
        if (ra->reachableTime() != 0)
        {
            EV << "RA's reachable time is non-zero ";
            if (ra->reachableTime() != ie->ipv6()-
>reachableTime())
```

```
            {
                EV << " and RA's and Host's reachable time
differ, \nsetting host's base"
                   << " reachable time to received value.\n";
                ie->ipv6()->setBaseReachableTime(ra-
>reachableTime());
                //If the new value differs from the previous
value, the host SHOULD
                //recompute a new random ReachableTime value.
                ie->ipv6()->setReachableTime(ie->ipv6()-
>generateReachableTime());
            }
            EV << endl;
        }

        //The RetransTimer variable SHOULD be copied from the
Retrans Timer field,
        //if the received value is non-zero.
        if (ra->retransTimer() != 0)
        {
            EV << "RA's retrans timer is non-zero, copying
retrans timer variable.\n";
            ie->ipv6()->setRetransTimer(ra->retransTimer());
        }

        /*If the MTU option is present, hosts SHOULD copy the
option's value into
        LinkMTU so long as the value is greater than or equal to
the minimum link MTU
        [IPv6] and does not exceed the default LinkMTU value
specified in the link
        type specific document (e.g., [IPv6-ETHER]).*/
        //TODO: not done yet

        processRAPrefixInfo(ra, ie);
    }

    void
IPv6NeighbourDiscovery::processRAPrefixInfo(IPv6RouterAdvertisemen
t *ra,
        InterfaceEntry *ie)
```

```
    {
        //Continued from section 6.3.4
        /*Prefix Information options that have the "on-link" (L)
flag set indicate a
        prefix identifying a range of addresses that should be
considered on-link.
        Note, however, that a Prefix Information option with the
on-link flag set to
        zero conveys no information concerning on-link
determination and MUST NOT be
        interpreted to mean that addresses covered by the prefix
are off-link. The
        only way to cancel a previous on-link indication is to
advertise that prefix
        with the L-bit set and the Lifetime set to zero. The
default behavior (see
        Section 5.2) when sending a packet to an address for
which no information is
        known about the on-link status of the address is to
forward the packet to a
        default router; the reception of a Prefix Information
option with the "on-link "
        (L) flag set to zero does not change this behavior. The
reasons for an address
        being treated as on-link is specified in the definition
of "on-link" in
        Section 2.1. Prefixes with the on-link flag set to zero
would normally have
        the autonomous flag set and be used by [ADDRCONF].*/
        IPv6NDPrefixInformation prefixInfo;
        //For each Prefix Information option
        for (int i = 0; i < ra->prefixInformationArraySize();
i++)
        {
            prefixInfo = ra->prefixInformation(i);
            if (!prefixInfo.onlinkFlag()) break;//skip to next
prefix option

            //with the on-link flag set, a host does the
following:
            EV << "Fetching Prefix Information:" << i+1 << " of "
```

```
                << ra->prefixInformationArraySize() << endl;
            uint prefixLength = prefixInfo.prefixLength();
            simtime_t validLifetime = prefixInfo.validLifetime();
            uint preferredLifetime =
prefixInfo.preferredLifetime();
            IPv6Address prefix = prefixInfo.prefix();

            //- If the prefix is the link-local prefix, silently
ignore the Prefix
            //Information option.
            if (prefix.isLinkLocal())
            {
                EV << "Prefix is link-local, ignoring prefix.\n";
                return;
            }

            //- If the prefix is not already present in the
Prefix List,
            if (!rt6->isPrefixPresent(prefix)) {
                //and the Prefix Information option's Valid
Lifetime field is non-zero,
                if (validLifetime != 0)
                {
                    /*create a new entry for the prefix and
initialize its invalidation
                    timer to the Valid Lifetime value in the
Prefix Information option.*/
                    rt6->addOrUpdateOnLinkPrefix(prefix,
prefixLength, ie->interfaceId(),
                        simTime()+validLifetime);
                }
                /*- If the Prefix Information option's Valid
Lifetime field is zero,
                and the prefix is not present in the host's
Prefix List,
                silently ignore the option.*/
            }
            else
            {
                /* If the new Lifetime value is zero, time-out
the prefix immediately
```

```
                (see Section 6.3.5).*/
                if (validLifetime == 0)
                {
                    EV << "Prefix Info's valid lifetime is 0,
time-out prefix\n";
                    rt6->removeOnLinkPrefix(prefix,
prefixLength);
                    return;
                }
                /*- If the prefix is already present in the
host's Prefix List as
                the result of a previously-received
advertisement, reset its
                invalidation timer to the Valid Lifetime value in
the Prefix
                Information option.*/
                rt6->addOrUpdateOnLinkPrefix(prefix,
prefixLength, ie->interfaceId(),
                    simTime()+validLifetime);
            }


    void
IPv6NeighbourDiscovery::processRAPrefixInfoForAddrAutoConf(
        IPv6NDPrefixInformation& prefixInfo, InterfaceEntry *ie)
    {
        EV << "Processing Prefix Info for address auto-
configuration.\n";
        IPv6Address prefix = prefixInfo.prefix();
        uint prefixLength = prefixInfo.prefixLength();
        simtime_t preferredLifetime =
prefixInfo.preferredLifetime();
        simtime_t validLifetime = prefixInfo.validLifetime();

        //RFC 2461: Section 5.5.3
        //First condition tested, the autonomous flag is already
set

        //b) If the prefix is the link-local prefix, silently
ignore the Prefix
        //Information option.
```

```
        if (prefixInfo.prefix().isLinkLocal() == true)
        {
            EV << "Prefix is link-local, ignore Prefix
Information Option\n";
            return;
        }

        //c) If the preferred lifetime is greater than the valid
lifetime, silently
        //ignore the Prefix Information option. A node MAY wish
to log a system
        //management error in this case.
        if (preferredLifetime > validLifetime)
        {
            EV << "Preferred lifetime is greater than valid
lifetime, ignore Prefix Information\n";
            return;
        }

        bool isPrefixAssignedToInterface = false;
        for (int i = 0; i < ie->ipv6()->numAddresses(); i++)
        {
            if (ie->ipv6()->address(i).matches(prefix,
prefixLength) == true)
                isPrefixAssignedToInterface = true;
        }
        /*d) If the prefix advertised does not match the prefix
of an address already
             in the list, and the Valid Lifetime is not 0, form
an address (and add
             it to the list) by combining the advertised prefix
with the link's
             interface identifier as follows:*/
        if (isPrefixAssignedToInterface == false && validLifetime
!= 0)
        {
            IPv6Address linkLocalAddress = ie->ipv6()-
>linkLocalAddress();
            // ASSERT(linkLocalAddress.isUnspecified() == false);
      if (linkLocalAddress.isUnspecified() == false) {
```

```
            IPv6Address newAddr =
linkLocalAddress.setPrefix(prefix, prefixLength);
                  //TODO: for now we leave the newly formed
address as not tentative,
                  //according to Greg, we have to always perform
DAD for a newly formed address.
                  EV << "Assigning new address to: " << ie->name()
<< endl;
                  ie->ipv6()->assignAddress(newAddr, false,
simTime()+validLifetime, simTime()+preferredLifetime);
            nb-
>fireChangeNotification(NF_IPv6_INTERFACECONFIG_CHANGED, ie-
>ipv6());
      }
         }
     void IPv6NeighbourDiscovery::createRATimer(InterfaceEntry
*ie)
     {
         cMessage *msg = new cMessage("sendPeriodicRA",
MK_SEND_PERIODIC_RTRADV);
         msg->setContextPointer(ie);
         AdvIfEntry *advIfEntry = new AdvIfEntry();
         advIfEntry->interfaceId = ie->interfaceId();
         advIfEntry->numRASent = 0;
         simtime_t interval
             = uniform(ie->ipv6()->minRtrAdvInterval(),ie->ipv6()-
>maxRtrAdvInterval());
         advIfEntry->raTimeoutMsg = msg;

         simtime_t nextScheduledTime = simTime() + interval;
         advIfEntry->nextScheduledRATime = nextScheduledTime;
         advIfList.insert(advIfEntry);
         EV << "Interval: " << interval << endl;
         EV << "Next scheduled time: " << nextScheduledTime <<
endl;
         //now we schedule the msg for whatever time that was
derived
         scheduleAt(nextScheduledTime, msg);
     }

     void IPv6NeighbourDiscovery::resetRATimer(InterfaceEntry *ie)
```

```
    {//Not used yet but could be useful later on.-WEI
        //Iterate through all RA timers within the Neighbour
Discovery module.
    /*
        for (RATimerList::iterator it=raTimerList.begin(); it !=
raTimerList.end(); it++)
        {
            cMessage *msg = (*it);
            InterfaceEntry *msgIE = (InterfaceEntry *)msg-
>contextPointer();
            //Find the timer that matches the given Interface
Entry.
            if (msgIE->outputPort() == ie->outputPort())
            {
                EV << "Resetting RA timer for port: " << ie-
>outputPort();
                cancelEvent(msg);//Cancel the next scheduled msg.
                simtime_t interval
                    = uniform(ie->ipv6()->minRtrAdvInterval(),ie-
>ipv6()->maxRtrAdvInterval());
                scheduleAt(simTime()+interval, msg);
            }
        }
    */
    }

    void IPv6NeighbourDiscovery::sendPeriodicRA(cMessage *msg)
    {
        InterfaceEntry *ie = (InterfaceEntry *)msg-
>contextPointer();
        AdvIfEntry *advIfEntry = fetchAdvIfEntry(ie);
        IPv6Address destAddr = IPv6Address("FF02::1");
        createAndSendRAPacket(destAddr, ie);
        advIfEntry->numRASent++;
        simtime_t nextScheduledTime;

        //RFC 2461, Section 6.2.4
        /*Whenever a multicast advertisement is sent from an
interface, the timer is
        reset to a uniformly-distributed random value between the
interface's
```

```
        configured MinRtrAdvInterval and MaxRtrAdvInterval;
expiration of the timer
        causes the next advertisement to be sent and a new random
value to be chosen.*/
        simtime_t interval
            = uniform(ie->ipv6()->minRtrAdvInterval(),ie->ipv6()-
>maxRtrAdvInterval());
        nextScheduledTime = simTime() + interval;

        /*For the first few advertisements (up to
MAX_INITIAL_RTR_ADVERTISEMENTS)
        sent from an interface when it becomes an advertising
interface,*/
        EV << "Num RA sent is: " << advIfEntry->numRASent <<
endl;
        EV << "maxInitialRtrAdvertisements is: " << ie->ipv6()-
>_maxInitialRtrAdvertisements() << endl;
        if(advIfEntry->numRASent <= ie->ipv6()-
>_maxInitialRtrAdvertisements())
        {
            if (interval > ie->ipv6()-
>_maxInitialRtrAdvertInterval())
            {
                //if the randomly chosen interval is greater than
MAX_INITIAL_RTR_ADVERT_INTERVAL,
                //the timer SHOULD be set to
MAX_INITIAL_RTR_ADVERT_INTERVAL instead.
                nextScheduledTime = simTime() + ie->ipv6()-
>_maxInitialRtrAdvertInterval();
                EV << "Sending initial RA but interval is too
long. Using default value." << endl;
            }
            else
                EV << "Sending initial RA. Using randomly
generated interval." << endl;
        }
        EV << "Next scheduled time: " << nextScheduledTime <<
endl;
        advIfEntry->nextScheduledRATime = nextScheduledTime;
        ASSERT(nextScheduledTime > simTime());
        scheduleAt(nextScheduledTime, msg);
```

```
    }

    void IPv6NeighbourDiscovery::sendSolicitedRA(cMessage *msg)
    {
        EV << "Send Solicited RA invoked!\n";
        InterfaceEntry *ie = (InterfaceEntry *)msg-
>contextPointer();
        IPv6Address destAddr = IPv6Address("FF02::1");
        EV << "Testing condition!\n";
        createAndSendRAPacket(destAddr, ie);
        delete msg;
    }

    bool
IPv6NeighbourDiscovery::validateRAPacket(IPv6RouterAdvertisement
*ra,
        IPv6ControlInfo *raCtrlInfo)
    {
        bool result = true;

        //RFC 2461: Section 6.1.2 Validation of Router
Advertisement Messages
        /*A node MUST silently discard any received Router
Advertisement
        messages that do not satisfy all of the following
validity checks:*/
        raCtrlInfo->srcAddr();
        //- IP Source Address is a link-local address.  Routers
must use
        //  their link-local address as the source for Router
Advertisement
        //  and Redirect messages so that hosts can uniquely
identify
        //  routers.
        if (raCtrlInfo->srcAddr().isLinkLocal() == false)
        {
            EV << "RA source address is not link-local. RA
validation failed!\n";
            result = false;
        }
```

```
        //- The IP Hop Limit field has a value of 255, i.e., the
packet
        //  could not possibly have been forwarded by a router.
        if (raCtrlInfo->hopLimit() != 255)
        {
            EV << "Hop limit is not 255! RA validation
failed!\n";
            result = false;
        }

        //- ICMP Code is 0.
        if (raCtrlInfo->protocol() != IP_PROT_IPv6_ICMP)
        {
            EV << "ICMP Code is not 0! RA validation failed!\n";
            result = false;
        }

        return result;
    }

    IPv6NeighbourSolicitation
*IPv6NeighbourDiscovery::createAndSendNSPacket(
        const IPv6Address& nsTargetAddr, const IPv6Address&
dgDestAddr,
        const IPv6Address& dgSrcAddr, InterfaceEntry *ie)
    {
        MACAddress myMacAddr = ie->macAddress();

        //Construct a Neighbour Solicitation message
        IPv6NeighbourSolicitation *ns = new
IPv6NeighbourSolicitation("NSpacket");
        ns->setType(ICMPv6_NEIGHBOUR_SOL);

        //Neighbour Solicitation Specific Information
        ns->setTargetAddress(nsTargetAddr);

        /*If the solicitation is being sent to a solicited-node
multicast
        address, the sender MUST include its link-layer address
(if it has
        one) as a Source Link-Layer Address option.*/
```

```
        if (dgDestAddr.matches(IPv6Address("FF02::1:FF00:0"),104)
&& // FIXME what's this? make constant...
            !dgSrcAddr.isUnspecified())
            ns->setSourceLinkLayerAddress(myMacAddr);

        sendPacketToIPv6Module(ns, dgDestAddr, dgSrcAddr, ie-
>interfaceId());

        return ns;
    }

    void
IPv6NeighbourDiscovery::processNSPacket(IPv6NeighbourSolicitation
*ns,
        IPv6ControlInfo *nsCtrlInfo)
    {
        //Control Information
        InterfaceEntry *ie = ift->interfaceAt(nsCtrlInfo-
>interfaceId());

        IPv6Address nsTargetAddr = ns->targetAddress();

        //RFC 2461:Section 7.2.3
        //If target address is not a valid "unicast" or anycast
address assigned to the
        //receiving interface, we should silently discard the
packet.
        if (validateNSPacket(ns, nsCtrlInfo) == false
            || ie->ipv6()->hasAddress(nsTargetAddr) == false)
        {
            bubble("NS validation failed\n");
            delete nsCtrlInfo;
            delete ns;
            return;
        }
        bubble("NS validation passed.\n");
        if (ie->ipv6()->isTentativeAddress(nsTargetAddr))
        {
            //If the Target Address is tentative, the Neighbor
Solicitation should
            //be processed as described in [ADDRCONF].
```

```
            EV << "Process NS for Tentative target address.\n";
            processNSForTentativeAddress(ns, nsCtrlInfo);
        }
        else
        {
            //Otherwise, the following description applies.
            EV << "Process NS for Non-Tentative target
address.\n";
            processNSForNonTentativeAddress(ns, nsCtrlInfo, ie);
        }
        delete nsCtrlInfo;
        delete ns;
    }

    bool
IPv6NeighbourDiscovery::validateNSPacket(IPv6NeighbourSolicitation
*ns,
        IPv6ControlInfo *nsCtrlInfo)
    {
        bool result = true;
        /*RFC 2461:7.1.1. Validation of Neighbor
Solicitations(some checks are omitted)
        A node MUST silently discard any received Neighbor
Solicitation
        messages that do not satisfy all of the following
validity checks:*/
        //- The IP Hop Limit field has a value of 255, i.e., the
packet
        //could not possibly have been forwarded by a router.
        if (nsCtrlInfo->hopLimit() != 255)
        {
            EV << "Hop limit is not 255! NS validation
failed!\n";
            result = false;
        }
        //- Target Address is not a multicast address.
        if (ns->targetAddress().isMulticast() == true)
        {
            EV << "Target address is a multicast address! NS
validation failed!\n";
            result = false;
```

```
        }
        //- If the IP source address is the unspecified address,
        if (nsCtrlInfo->srcAddr().isUnspecified())
        {
            EV << "Source Address is unspecified\n";
            //the IP destination address is a solicited-node
multicast address.
            if (nsCtrlInfo-
>destAddr().matches(IPv6Address::SOLICITED_NODE_PREFIX,104) ==
false)
            {
                EV << " but IP dest address is not a solicited-
node multicast address! NS validation failed!\n";
                result = false;
            }
            //there is no source link-layer address option in the
message.
            else if (ns->sourceLinkLayerAddress().isUnspecified()
== false)
            {
                EV << " but Source link-layer address is not
empty! NS validation failed!\n";
                result = false;
            }
            else
                EV << "NS Validation Passed\n";
        }

        return result;
    }

    void
IPv6NeighbourDiscovery::processNSForTentativeAddress(IPv6Neighbour
Solicitation *ns,
        IPv6ControlInfo *nsCtrlInfo)
    {
        //Control Information
        IPv6Address nsSrcAddr = nsCtrlInfo->srcAddr();
        IPv6Address nsDestAddr = nsCtrlInfo->destAddr();
```

```
        ASSERT(nsSrcAddr.isUnicast() ||
nsSrcAddr.isUnspecified());
        //solicitation is processed as described in
RFC2462:section 5.4.3

        if (nsSrcAddr.isUnspecified())
        {
            EV << "Source Address is UNSPECIFIED. Sender is
performing DAD\n";
            //Sender performing Duplicate Address Detection
            if (rt6->localDeliver(nsSrcAddr))
                EV << "NS comes from myself. Ignoring NS\n";
            else
                EV << "NS comes from another node. Address is
duplicate!\n";
                error("Duplicate Address Detected! Manual
Attention Required!");
        }
        else if (nsSrcAddr.isUnicast())
        {
            //Sender performing address resolution
            EV << "Sender is performing Address Resolution\n";
            EV << "Target Address is tentative. Ignoring NS.\n";
        }
    }

    void
IPv6NeighbourDiscovery::processNSForNonTentativeAddress(IPv6Neighb
ourSolicitation *ns,
        IPv6ControlInfo *nsCtrlInfo, InterfaceEntry *ie)
    {
        //Neighbour Solicitation Information
        MACAddress nsMacAddr = ns->sourceLinkLayerAddress();

        int ifID = ie->interfaceId();

        //target addr is not tentative addr
        //solicitation processed as described in RFC2461:section
7.2.3
        if (nsCtrlInfo->srcAddr().isUnspecified())
        {
```

```
            EV << "Address is duplicate! Inform Sender of
duplicate address!\n";
            sendSolicitedNA(ns, nsCtrlInfo, ie);
        }
        else
        {
            processNSWithSpecifiedSrcAddr(ns, nsCtrlInfo, ie);
        }
    }

    void
IPv6NeighbourDiscovery::processNSWithSpecifiedSrcAddr(IPv6Neighbou
rSolicitation *ns,
        IPv6ControlInfo *nsCtrlInfo, InterfaceEntry *ie)
    {
        //RFC 2461, Section 7.2.3
        /*If the Source Address is not the unspecified address
and, on link layers
        that have addresses, the solicitation includes a Source
Link-Layer Address
        option, then the recipient SHOULD create or update the
Neighbor Cache entry
        for the IP Source Address of the solicitation.*/

        //Neighbour Solicitation Information
        MACAddress nsMacAddr = ns->sourceLinkLayerAddress();

        int ifID = ie->interfaceId();

        //Look for the Neighbour Cache Entry
        Neighbour *entry = neighbourCache.lookup(nsCtrlInfo-
>srcAddr(), ifID);

        if (entry == NULL)
        {
            /*If an entry does not already exist, the node SHOULD
create a new one
            and set its reachability state to STALE as specified
in Section 7.3.3.*/
            EV << "Neighbour Entry not found. Create a Neighbour
Cache Entry.\n";
```

```
            neighbourCache.addNeighbour(nsCtrlInfo->srcAddr(),
ifID, nsMacAddr);
        }
        else
        {
            /*If an entry already exists, and the cached link-
layer address differs from
            the one in the received Source Link-Layer option,*/
            if (!(entry->macAddress.equals(nsMacAddr)) &&
!nsMacAddr.isUnspecified())
            {
                //the cached address should be replaced by the
received address
                entry->macAddress = nsMacAddr;
                //and the entry's reachability state MUST be set
to STALE.
                entry->reachabilityState =
IPv6NeighbourCache::STALE;
            }
        }
        /*After any updates to the Neighbor Cache, the node sends
a Neighbor
        Advertisement response as described in the next
section.*/
        sendSolicitedNA(ns, nsCtrlInfo, ie);
    }

    void
IPv6NeighbourDiscovery::sendSolicitedNA(IPv6NeighbourSolicitation
*ns,
        IPv6ControlInfo *nsCtrlInfo, InterfaceEntry *ie)
    {
        IPv6NeighbourAdvertisement *na = new
IPv6NeighbourAdvertisement("NApacket");
        //RFC 2461: Section 7.2.4
        /*A node sends a Neighbor Advertisement in response to a
valid Neighbor
        Solicitation targeting one of the node's assigned
addresses.  The
        Target Address of the advertisement is copied from the
Target Address
```

```
        of the solicitation.*/
        na->setTargetAddress(ns->targetAddress());

        /*If the solicitation's IP Destination Address is not a
multicast address,
        the Target Link-Layer Address option MAY be omitted; the
neighboring node's
        cached value must already be current in order for the
solicitation to have
        been received. If the solicitation's IP Destination
Address is a multicast
        address, the Target Link-Layer option MUST be included in
the advertisement.*/
        na->setTargetLinkLayerAddress(ie->macAddress());//here,
we always include the MAC addr.

        /*Furthermore, if the node is a router, it MUST set the
Router flag to one;
        otherwise it MUST set the flag to zero.*/
        na->setRouterFlag(rt6->isRouter());

        /*If the (NS)Target Address is either an anycast address
or a unicast
        address for which the node is providing proxy service, or
the Target
        Link-Layer Address option is not included,*/
        //TODO:ANYCAST will not be implemented here!
        if (ns->sourceLinkLayerAddress().isUnspecified())
            //the Override flag SHOULD be set to zero.
            na->setOverrideFlag(false);
        else
            //Otherwise, the Override flag SHOULD be set to one.
            na->setOverrideFlag(true);
        /*Proper setting of the Override flag ensures that nodes
give preference to
        non-proxy advertisements, even when received after proxy
advertisements, and
        also ensures that the first advertisement for an anycast
address "wins".*/

        IPv6Address naDestAddr;
```

```
        //If the source of the solicitation is the unspecified
address,
        if(nsCtrlInfo->srcAddr().isUnspecified())
        {
            /*the node MUST set the Solicited flag to zero and
multicast the advertisement
            to the all-nodes address.*/
            na->setSolicitedFlag(false);
            naDestAddr = IPv6Address::ALL_NODES_2;
        }
        else
        {
            /*Otherwise, the node MUST set the Solicited flag to
one and unicast
            the advertisement to the Source Address of the
solicitation.*/
            na->setSolicitedFlag(true);
            naDestAddr = nsCtrlInfo->srcAddr();
        }

        /*If the Target Address is an anycast address the sender
SHOULD delay sending
        a response for a random time between 0 and
MAX_ANYCAST_DELAY_TIME seconds.*/
        /*TODO: More associated complexity for this one. We will
have to delay
        sending off the solicitation. Perhaps the self message
could have a context
        pointer pointing to a struct with enough info to create
and send a NA packet.*/

        /*Because unicast Neighbor Solicitations are not required
to include a
        Source Link-Layer Address, it is possible that a node
sending a
        solicited Neighbor Advertisement does not have a
corresponding link-
        layer address for its neighbor in its Neighbor Cache.  In
such
        situations, a node will first have to use Neighbor
Discovery to
```

```
        determine the link-layer address of its neighbor (i.e,
send out a
        multicast Neighbor Solicitation).*/
        //TODO: if above mentioned happens, can addr resolution
be performed for ND messages?
        //if no link-layer addr exists for unicast addr when
sending solicited NA, we should
        //add the NA to the list of queued packets. What if we
have a list of queued
        //packets for different unicast solicitations? each time
addr resolution is
        //done we should check the destinations of the list of
queued packets and send
        //off the respective ones.
        IPv6Address myIPv6Addr = ie->ipv6()->preferredAddress();
        sendPacketToIPv6Module(na, naDestAddr, myIPv6Addr, ie-
>interfaceId());
    }

    void IPv6NeighbourDiscovery::sendUnsolicitedNA(InterfaceEntry
*ie)
    }

    void
IPv6NeighbourDiscovery::processNAPacket(IPv6NeighbourAdvertisement
*na,
        IPv6ControlInfo *naCtrlInfo)
    {
        if (validateNAPacket(na, naCtrlInfo) == false)
        {
            delete naCtrlInfo;
            delete na;
            return;
        }

        //Neighbour Advertisement Information
        IPv6Address naTargetAddr = na->targetAddress();

        //First, we check if the target address in NA is found in
the interface it
        //was received on is tentative.
```

```
        InterfaceEntry *ie = ift->interfaceAt(naCtrlInfo-
>interfaceId());
        if (ie->ipv6()->isTentativeAddress(naTargetAddr))
        {
            error("Duplicate Address Detected! Manual attention
needed!");
        }
        //Logic as defined in Section 7.2.5
        Neighbour *neighbourEntry =
neighbourCache.lookup(naTargetAddr, ie->interfaceId());

        if (neighbourEntry == NULL)
        {
            EV << "NA received. Target Address not found in
Neighbour Cache\n";
            EV << "Dropping NA packet.\n";
            delete naCtrlInfo;
            delete na;
            return;
        }

        //Target Address has entry in Neighbour Cache
        EV << "NA received. Target Address found in Neighbour
Cache\n";

        if (neighbourEntry->reachabilityState ==
IPv6NeighbourCache::INCOMPLETE)
            processNAForIncompleteNCEState(na, neighbourEntry);
        else
            processNAForOtherNCEStates(na, neighbourEntry);
        delete naCtrlInfo;
        delete na;
    }

    bool
IPv6NeighbourDiscovery::validateNAPacket(IPv6NeighbourAdvertisemen
t *na,
        IPv6ControlInfo *naCtrlInfo)
    {
        bool result = true;//adopt optimistic approach
```

```
        //RFC 2461:7.1.2 Validation of Neighbor
Advertisments(some checks are omitted)
        //A node MUST silently discard any received Neighbor
Advertisment messages
        //that do not satisfy all of the following validity
checks:

        //- The IP Hop Limit field has a value of 255, i.e., the
packet
        //  could not possibly have been forwarded by a router.
        if (naCtrlInfo->hopLimit() != 255)
        {
            EV << "Hop Limit is not 255! NA validation
failed!\n";
            result = false;
        }

        //- Target Address is not a multicast address.
        if (na->targetAddress().isMulticast() == true)
        {
            EV << "Target Address is a multicast address! NA
validation failed!\n";
            result = false;
        }

        //- If the IP Destination Address is a multicast address
the Solicited flag
        //  is zero.
        if (naCtrlInfo->destAddr().isMulticast())
        {
            if (na->solicitedFlag() == true)
            {
                EV << "Dest Address is multicast address but
solicted flag is 0!\n";
                result = false;
            }
        }

        if (result == true) bubble("NA validation passed.");
        else bubble("NA validation failed.");
        return result;
```

```
    }

    void IPv6NeighbourDiscovery::processNAForIncompleteNCEState(
                        IPv6NeighbourAdvertisement *na, Neighbour
*nce)
    {
        MACAddress naMacAddr = na->targetLinkLayerAddress();
        bool naRouterFlag = na->routerFlag();
        bool naSolicitedFlag = na->solicitedFlag();
        const Key *nceKey = nce->nceKey;
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);

        /*If the target's neighbour Cache entry is in the
INCOMPLETE state when the
        advertisement is received, one of two things happens.*/
        if (naMacAddr.isUnspecified())
        {
            /*If the link layer has addresses and no Target Link-
Layer address option
            is included, the receiving node SHOULD silently
discard the received
            advertisement.*/
            EV << "No MAC Address specified in NA. Ignoring
NA\n";
            return;
        }
        else
        {
            //Otherwise, the receiving node performs the
following steps:
            //- It records the link-layer address in the
neighbour Cache entry.
            EV << "ND is updating Neighbour Cache Entry.\n";
            nce->macAddress = naMacAddr;

            //- If the advertisement's Solicited flag is set, the
state of the
            //  entry is set to REACHABLE, otherwise it is set to
STALE.
            if (naSolicitedFlag == true)
```

```
            {
                nce->reachabilityState =
IPv6NeighbourCache::REACHABLE;
                EV << "Reachability confirmed through successful
Addr Resolution.\n";
                nce->reachabilityExpires = simTime() + ie-
>ipv6()->_reachableTime();
            }
            else
                nce->reachabilityState =
IPv6NeighbourCache::STALE;

            //- It sets the IsRouter flag in the cache entry
based on the Router
            //  flag in the received advertisement.
            nce->isRouter = naRouterFlag;

            //- It sends any packets queued for the neighbour
awaiting address
            //  resolution.
            sendQueuedPacketsToIPv6Module(nce);
            cancelEvent(nce->arTimer);
        }
    }

    void IPv6NeighbourDiscovery:: processNAForOtherNCEStates(
        IPv6NeighbourAdvertisement *na, Neighbour *nce)
    {
        bool naRouterFlag = na->routerFlag();
        bool naSolicitedFlag = na->solicitedFlag();
        bool naOverrideFlag = na->overrideFlag();
        MACAddress naMacAddr = na->targetLinkLayerAddress();
        const Key *nceKey = nce->nceKey;
        InterfaceEntry *ie = ift->interfaceAt(nceKey-
>interfaceID);

        /*draft-ietf-ipv6-2461bis-04
        Section 7.2.5: Receipt of Neighbour Advertisements
        If the target's Neighbor Cache entry is in any state
other than INCOMPLETE
```

```
        when the advertisement is received, the following actions
take place:*/

        if (naOverrideFlag == false && !(naMacAddr.equals(nce-
>macAddress))
            && !(naMacAddr.isUnspecified()))
        {
            EV << "NA override is FALSE and NA MAC addr is
different.\n";
            //I. If the Override flag is clear and the supplied
link-layer address
            //   differs from that in the cache, then one of two
actions takes place:
            //(Note: An unspecified MAC should not be compared
with the NCE's mac!)
            //a. If the state of the entry is REACHABLE,
            if (nce->reachabilityState ==
IPv6NeighbourCache::REACHABLE)
            {
                EV << "NA mac is different. Change NCE state from
REACHABLE to STALE\n";
                //set it to STALE, but do not update the entry in
any other way.
                nce->reachabilityState =
IPv6NeighbourCache::STALE;
            }
            else
                //b. Otherwise, the received advertisement should
be ignored and
                //MUST NOT update the cache.
                EV << "NCE is not in REACHABLE state. Ignore
NA.\n";
        }
        else if (naOverrideFlag == true || naMacAddr.equals(nce-
>macAddress)
            || naMacAddr.isUnspecified())
        {
            EV << "NA override flag is TRUE. or Advertised MAC is
same as NCE's. or"
               << " NA MAC is not specified.\n";
```

```
            /*II. If the Override flag is set, or the supplied
link-layer address
            is the same as that in the cache, or no Target Link-
layer address
            option was supplied, the received advertisement MUST
update the
            Neighbor Cache entry as follows:*/

            /*- The link-layer address in the Target Link-Layer
Address option
                MUST be inserted in the cache (if one is supplied
and is
                Different than the already recorded address).*/
            if (!(naMacAddr.isUnspecified()) &&
                !(naMacAddr.equals(nce->macAddress)))
            {
                EV << "Updating NCE's MAC addr with NA's.\n";
                nce->macAddress = naMacAddr;
            }

            //- If the Solicited flag is set,
            if (naSolicitedFlag == true)
            {
                EV << "Solicited Flag is TRUE. Set NCE state to
REACHABLE.\n";
                //the state of the entry MUST be set to
REACHABLE.
                nce->reachabilityState =
IPv6NeighbourCache::REACHABLE;
                //We have to cancel the NUD self timer message if
there is one.
                cMessage *msg = nce->nudTimeoutEvent;
                if (msg != NULL)
                {
                    EV << "NUD in progress. Cancelling NUD
Timer\n";
                    bubble("Reachability Confirmed via NUD.");
                    nce->reachabilityExpires = simTime() + ie-
>ipv6()->_reachableTime();
                    cancelEvent(msg);
                    delete msg;
```

```
                }
            }
            else
            {
                //If the Solicited flag is zero
                EV << "Solicited Flag is FALSE.\n";
                //and the link layer address was updated with a
different address
                if (!(naMacAddr.equals(nce->macAddress)))
                {
                    EV << "NA's MAC is different from NCE's.Set
NCE state to STALE\n";
                    //the state MUST be set to STALE.
                    nce->reachabilityState =
IPv6NeighbourCache::STALE;
                }
                else
                    //Otherwise, the entry's state remains
unchanged.
                    EV << "NA's MAC is the same as NCE's. State
remains unchanged.\n";
            }
            //(Next paragraph with explanation is omitted.-WEI)

            /*- The IsRouter flag in the cache entry MUST be set
based on the
            Router flag in the received advertisement.*/
            EV << "Updating NCE's router flag to " <<
naRouterFlag << endl;
            nce->isRouter = naRouterFlag;

            //TODO: To be implemented
            /*In those cases where the IsRouter flag changes from
TRUE to FALSE as a
            result of this update, the node MUST remove that
router from the Default
            Router List and update the Destination Cache entries
for all destinations
            using that neighbor as a router as specified in
Section 7.3.3. This is
```

```
            needed to detect when a node that is used as a router
stops forwarding
            packets due to being configured as a host.*/
        }
    }

    IPv6Redirect
*IPv6NeighbourDiscovery::createAndSendRedirectPacket(InterfaceEntr
y *ie)
    {
        //Construct a Redirect message
        IPv6Redirect *redirect = new IPv6Redirect("redirectMsg");
        redirect->setType(ICMPv6_REDIRECT);

        //Redirect Message Specific Information
        //redirect->setTargetAddress();
        //redirect->setDestinationAddress();

        //Possible Option
        //redirect->setTargetLinkLayerAddress();

        return redirect;
    }

    void
IPv6NeighbourDiscovery::processRedirectPacket(IPv6Redirect
*redirect,
        IPv6ControlInfo *ctrlInfo)
    {
        //First we need to extract information from the redirect
message
        IPv6Address targetAddr = redirect-
>targetAddress();//Addressed to me
        IPv6Address destAddr = redirect-
>destinationAddress();//new dest addr

        //Optional
        MACAddress macAddr = redirect->targetLinkLayerAddress();
    }
```

8.10. Arquivo RoutingTable6.h

```
#ifndef __ROUTINGTABLE6_H
#define __ROUTINGTABLE6_H

#include <vector>
#include <omnetpp.h>
#include "INETDefs.h"
#include "IPv6Address.h"
#include "InterfaceTable.h"


/**
 * Represents a route in the route table. Routes with
src=FROM_RA represent
 * on-link prefixes advertised by routers.
 */
class INET_API IPv6Route : public cPolymorphic
{
  public:
    /** Specifies where the route comes from */
    enum RouteSrc
    {
        FROM_RA,        ///< on-link prefix, from Router
Advertisement
        OWN_ADV_PREFIX, ///< on routers: on-link prefix that
the router **itself** advertises on the link
        STATIC,         ///< static route
        ROUTING_PROT, ///< route is managed by a routing
protocol (OSPF,BGP,etc)
    };

  private:
    IPv6Address _destPrefix;
    short _length;
    RouteSrc _src;
    int _interfaceID;
    IPv6Address _nextHop; // unspecified means "direct"
    simtime_t _expiryTime; // if route is an advertised
prefix: prefix lifetime
```

```
        int _metric;

      public:
        /**
         * Constructor. The destination prefix and the route
source is passed
         * to the constructor and cannot be changed afterwards.
         */
        IPv6Route(IPv6Address destPrefix, int length, RouteSrc
src) {
            _destPrefix = destPrefix;
            _length = length;
            _src = src;
            _interfaceID = -1;
            _expiryTime = 0;
            _metric = 0;
        }

        virtual std::string info() const;
        virtual std::string detailedInfo() const;
        static const char *routeSrcName(RouteSrc src);

        void setInterfaceID(int interfaceId)  {_interfaceID =
interfaceId;}
        void setNextHop(const IPv6Address& nextHop)  {_nextHop =
nextHop;}
        void setExpiryTime(simtime_t expiryTime)  {_expiryTime =
expiryTime;}
        void setMetric(int metric)  {_metric = _metric;}

        const IPv6Address& destPrefix() const {return
_destPrefix;}
        int prefixLength() const  {return _length;}
        RouteSrc src() const  {return _src;}
        int interfaceID() const  {return _interfaceID;}
        const IPv6Address& nextHop() const  {return _nextHop;}
        simtime_t expiryTime() const  {return _expiryTime;}
        int metric() const  {return _metric;}
    };
```

```
    /**
     * Represents the IPv6 routing table and neighbour discovery
data structures.
     * This object has one instance per host or router.
     *
     * See the NED documentation for general overview.
     *
     * This is a simple module without gates, it requires
function calls to it
     * (message handling does nothing). Methods are provided for
reading and
     * updating the interface table and the route table, as well
as for unicast
     * and multicast routing.
     *
     * The route table is read from a file. The route table can
also
     * be read and modified during simulation, typically by
routing protocol
     * implementations.
     */
    class INET_API RoutingTable6 : public cSimpleModule
    {
      private:
        InterfaceTable *ift; // cached pointer
        bool isrouter;

        // Destination Cache maps dest address to next hop and
interfaceId.
        // NOTE: nextHop might be a link-local address from which
interfaceId cannot be deduced
        struct DestCacheEntry
        {
            int interfaceId;
            IPv6Address nextHopAddr;
            // more destination specific data may be added here,
e.g. path MTU
        };
        friend std::ostream& operator<<(std::ostream& os, const
DestCacheEntry& e);
        typedef std::map<IPv6Address,DestCacheEntry> DestCache;
```

```
    DestCache destCache;

    // RouteList contains local prefixes, and (for routers)
    // static, OSPF, RIP etc routes as well
    typedef std::vector<IPv6Route*> RouteList;
    RouteList routeList;

  private:
    // internal: routes of different type can only be added
via well-defined functions
    void addRoute(IPv6Route *route);
    // helper for addRoute()
    static bool routeLessThan(const IPv6Route *a, const
IPv6Route *b);
    // internal
    void configureInterfaceForIPv6(InterfaceEntry *ie);
    /**
     *  RFC 3513: Section 2.8 A Node's Required Address
     *  Assign the various addresses to the node's respective
interface. This
     *  should be done when the IPv6 Protocol stack is
created.
     */
    void assignRequiredNodeAddresses(InterfaceEntry *ie);
    // internal
    void configureInterfaceFromXML(InterfaceEntry *ie,
cXMLElement *cfg);
    void configureRoutesFromXML();

  protected:
    // displays summary above the icon
    void updateDisplayString();

  public:
    RoutingTable6();
    virtual ~RoutingTable6();

  protected:
    int numInitStages() const  {return 5;}
    void initialize(int stage);
    void parseXMLConfigFile();
```

```
         /**
          * Raises an error.
          */
         void handleMessage(cMessage *);

       public:
         /** @name Interfaces */
         //@{
         /**
          * Returns an interface given by its address. Returns
NULL if not found.
          */
         InterfaceEntry *interfaceByAddress(const IPv6Address&
address);
         //@}

         /**
          * IP forwarding on/off
          */
         bool isRouter()  const {return isrouter;}

         /** @name Routing functions */
         //@{
         /**
          * Checks if the address is one of the host's addresses,
i.e.
          * assigned to one of its interfaces (tentatively or
not).
          */
         bool localDeliver(const IPv6Address& dest);

         /**
          * Looks up the given destination address in the
Destination Cache,
          * then returns the next-hop address and the interface in
the outInterfaceId
          * variable. If the destination is not in the cache,
outInterfaceId is set to
          * -1 and the unspecified address is returned. The caller
should check
```

```
         * for interfaceId==-1, because unspecified address is
also returned
         * if the link layer doesn't use addresses at all (e.g.
PPP).
         *
         * NOTE: outInterfaceId is an OUTPUT parameter -- its
initial value is ignored,
         * and the lookupDestCache() sets it to the correct value
instead.
         */
        const IPv6Address& lookupDestCache(const IPv6Address&
dest, int& outInterfaceId);

        /**
         * Performs longest prefix match in the routing table and
returns
         * the resulting route, or NULL if there was no match.
         */
        const IPv6Route *doLongestPrefixMatch(const IPv6Address&
dest);

        /**
         * Checks if the given prefix already exists in the
routing table (prefix list)
         */
        bool isPrefixPresent(const IPv6Address& prefix);

        //TBD multicast delivery
        //@}

        /** @name Managing the destination cache */
        /**
         * Add or update a destination cache entry.
         */
        void updateDestCache(const IPv6Address& dest, const
IPv6Address& nextHopAddr, int interfaceId);

        /**
         * Discard all entries in destination cache
         */
        void purgeDestCache();
```

```
        /**
         * Discard all entries in destination cache where next
hop is the given
         * address on the given interface. This is typically
called when a router
         * becomes unreachable, and all destinations going via
that router have to
         * go though router selection again.
         */
        void purgeDestCacheEntriesToNeighbour(const IPv6Address&
nextHopAddr, int interfaceId);
        //@}

        /** @name Managing prefixes and the route table */
        //@{
        /**
         * Add on-link prefix (route of type FROM_RA), or update
existing one.
         * To be called from code processing on-link prefixes in
Router Advertisements.
         * Expiry time can be derived from the Valid Lifetime
field
         * in the Router Advertisements.
         *
         * NOTE: This method does NOT update the lifetime of
matching addresses
         * in the InterfaceTable (see IPv6InterfaceData); that
has to be done
         * separately.
         */
        void addOrUpdateOnLinkPrefix(const IPv6Address&
destPrefix, int prefixLength,
                                     int interfaceId, simtime_t
expiryTime);

        /**
         * Remove an on-link prefix. To be called when the prefix
gets advertised
         * with zero lifetime, or to purge an expired prefix.
         *
```

```
         * NOTE: This method does NOT remove the matching
addresses from the
         * InterfaceTable (see IPv6InterfaceData); that has to be
done separately.
         */
        void removeOnLinkPrefix(const IPv6Address& destPrefix,
int prefixLength);

        /**
         * Add route of type OWN_ADV_PREFIX. This is a prefix
that *this* router
         * advertises on this interface.
         */
        void addOrUpdateOwnAdvPrefix(const IPv6Address&
destPrefix, int prefixLength,
                                     int interfaceId, simtime_t
expiryTime);

        /**
         * Creates a static route. If metric is omitted, it gets
initialized
         * to the interface's metric value.
         */
        void addStaticRoute(const IPv6Address& destPrefix, int
prefixLength,
                            unsigned int interfaceId, const
IPv6Address& nextHop,
                            int metric=0);

        /**
         *  Adds a default route for a host. This method requires
the RA's source
         *  address and the router expiry time plus the
simTime().
         */
        void addDefaultRoute(const IPv6Address& raSrcAddr,
unsigned int ifID,
            simtime_t routerLifetime);

        /**
```

```
         * Adds the given route (which can be OSPF, BGP, RIP or
any other route)
         * with src==ROUTING_PROT. To store additional
information with the route,
         * one can subclass from IPv6Route and add more fields.
         */
        void addRoutingProtocolRoute(IPv6Route *route);

        /**
         * Deletes the given route from the route table.
         */
        void removeRoute(IPv6Route *route);

        /**
         * Return the number of routes.
         */
        int numRoutes() const;

        /**
         * Return the ith route.
         */
        IPv6Route *route(int i);
        //@}

    };

    #endif
```

8.11. Arquivo RoutingTable6.cc

```
#include <algorithm>
#include "opp_utils.h"
#include "RoutingTable6.h"
#include "IPv6InterfaceData.h"
#include "InterfaceTableAccess.h"
#include "IPAddressResolver.h"


Define_Module(RoutingTable6);
```

```
std::string IPv6Route::info() const
{
   std::stringstream out;
   out << destPrefix() << "/" << prefixLength() << " --> ";
   out << "if=" << interfaceID() << " next hop:" << nextHop(); // FIXME try printing
interface name
   out << " " << routeSrcName(src());
   if (expiryTime()>0)
      out << " exp:" << simtimeToStr(expiryTime());
   return out.str();
}

std::string IPv6Route::detailedInfo() const
{
   return std::string();
}

const char *IPv6Route::routeSrcName(RouteSrc src)
{
   switch (src)
   {
      case FROM_RA:        return "FROM_RA";
      case OWN_ADV_PREFIX:  return "OWN_ADV_PREFIX";
      case STATIC:         return "STATIC";
      case ROUTING_PROT:    return "ROUTING_PROT";
      default:             return "???";
   }
}

//----

std::ostream& operator<<(std::ostream& os, const IPv6Route& e)
{
   os << e.info();
   return os;
};

std::ostream& operator<<(std::ostream& os, const RoutingTable6::DestCacheEntry& e)
{
```

```
    os << "if=" << e.interfaceId << " " << e.nextHopAddr;  //FIXME try printing interface
name
    return os;
};

RoutingTable6::RoutingTable6()
{
}

RoutingTable6::~RoutingTable6()
{
  for (unsigned int i=0; i<routeList.size(); i++)
    delete routeList[i];
}

void RoutingTable6::initialize(int stage)
{
  if (stage==1)
  {
    ift = InterfaceTableAccess().get();

    WATCH_PTRVECTOR(routeList);
    WATCH_MAP(destCache); // FIXME commented out for now
    isrouter = par("isRouter");
    WATCH(isrouter);

    // add IPv6InterfaceData to interfaces
    for (int i=0; i<ift->numInterfaces(); i++)
    {
      InterfaceEntry *ie = ift->interfaceAt(i);
      configureInterfaceForIPv6(ie);
    }

    parseXMLConfigFile();

    // skip hosts
    //if (isrouter)
  if (isRouter())
    {
      // add globally routable prefixes to routing table
```

```
        for (int x = 0; x < ift->numInterfaces(); x++)
        {
          InterfaceEntry *ie = ift->interfaceAt(x);

          if (ie->isLoopback())
            continue;

          for (int y = 0; y < ie->ipv6()->numAdvPrefixes(); y++)
            if (ie->ipv6()->advPrefix(y).prefix.isGlobal())
              addOrUpdateOwnAdvPrefix(ie->ipv6()->advPrefix(y).prefix,
                                      ie->ipv6()->advPrefix(y).prefixLength,
                                      x, 0);
        }
      }
    }
    else if (stage==4)
    {
      // configurator adds routes only in stage==3
      updateDisplayString();
    }
}

typedef cXMLElementList::iterator NodeIt;

static const char *getRequiredAttr(cXMLElement *elem, const char *attrName)
{
    const char *s = elem->getAttribute(attrName);
    if (!s)
        opp_error("element <%s> misses required attribute %s at %s",
                  elem->getTagName(), attrName, elem->getSourceLocation());
    return s;
}

static bool toBool(const char *s, bool defaultValue=false)
{
    if (!s)
        return defaultValue;
    return !strcmp(s,"on") || !strcmp(s,"true") || !strcmp(s,"yes");
}
```

```
void RoutingTable6::parseXMLConfigFile()
{
  // TODO to be revised by Andras
  // configure interfaces from XML config file
  cXMLElement *config = par("routingTableFile");
  for (cXMLElement *child=config->getFirstChild(); child; child = child->getNextSibling())
  {
    //std::cout << "configuring interfaces from XML file." << endl;
    //std::cout << "selected element is: " << child->getTagName() << endl;
    // we ensure that the selected element is local.
    if (opp_strcmp(child->getTagName(),"local")!=0) continue;
    //ensure that this is the right parent module we are configuring.
    if (opp_strcmp(child->getAttribute("node"),parentModule()->fullName())!=0)
      continue;
    //Go one level deeper.
    //child = child->getFirstChild();
    for (cXMLElement *ifTag=child->getFirstChild(); ifTag; ifTag = ifTag->getNextSibling())
    {
      //The next tag should be "interface".
      if (opp_strcmp(ifTag->getTagName(),"interface")!=0)
        continue;
      //std::cout << "Getting attribute: name" << endl;
      const char *ifname = ifTag->getAttribute("name");
      if (!ifname)
        error("<interface> without name attribute at %s", child->getSourceLocation());
      InterfaceEntry *ie = ift->interfaceByName(ifname);
      if (!ie)
        error("no interface named %s was registered, %s", ifname, child->getSourceLocation());
      configureInterfaceFromXML(ie, ifTag);
    }
   configureRoutesFromXML();
  }
}

void RoutingTable6::configureRoutesFromXML()
{
  cXMLElement *config = par("routingTableFile");
```

```
            for (cXMLElement *child=config->getFirstChild(); child; child = child-
>getNextSibling())
            {
               //std::cout << "configuring interfaces from XML file." << endl;
               //std::cout << "selected element is: " << child->getTagName() << endl;
               // we ensure that the selected element is local.
               if (opp_strcmp(child->getTagName(),"local")!=0) continue;
               //ensure that this is the right parent module we are configuring.
               if (opp_strcmp(child->getAttribute("node"),parentModule()->fullName())!=0)
                  continue;
               //Go one level deeper.
               //child = child->getFirstChild();
               const char* nameNode = child->getAttribute("node");
               std::cout << "Rotas para " << nameNode << endl;
               for (cXMLElement *ifTag=child->getFirstChild(); ifTag; ifTag = ifTag-
>getNextSibling())
               {
                 cXMLElementList routeList = ifTag->getElementsByTagName("routeEntry");
                 for (unsigned int i=0; i<routeList.size(); i++)
                        {
                  cXMLElement *nre = routeList[i];
               const char* iface = getRequiredAttr(nre, "routeIface");
               const char* dest = getRequiredAttr(nre, "routeDestination");
               const char* prefixLen = getRequiredAttr(nre, "routePrefix");
               const char* nextHop = getRequiredAttr(nre, "routeNextHop");
                  std::cout << " iface=" << iface << " dest=" << dest << " prefixLenght=" <<
prefixLen << " nextHop=" << nextHop << endl;
                  if (dest == "") { std::cout << "Destino Vazio" << endl; return; }
               InterfaceEntry *ie = ift->interfaceByName(iface);
                  if(!ie)  { std::cout << "Interface nÃ£o encontrada" << endl; return; }
               int ifaceIdx = ie->interfaceId();
               IPv6Address nextHopAddr(nextHop);
                        IPv6Address destAddr;
               if (nextHop == "") ASSERT(nextHopAddr ==
IPv6Address::UNSPECIFIED_ADDRESS);
                  destAddr=IPAddressResolver().resolveIPv6(dest);
                  int lenPrefix = atoi(prefixLen);
                  addStaticRoute(destAddr, lenPrefix, ifaceIdx, nextHopAddr, 0);
              }
               }
```

```
    }
}

void RoutingTable6::updateDisplayString()
{
   if (!ev.isGUI())
      return;

   char buf[80];
   sprintf(buf, "%d routes\n%d destcache entries", numRoutes(), destCache.size());
   displayString().setTagArg("t",0,buf);
}

void RoutingTable6::handleMessage(cMessage *msg)
{
   opp_error("This module doesn't process messages");
}

void RoutingTable6::configureInterfaceForIPv6(InterfaceEntry *ie)
{
   IPv6InterfaceData *ipv6IfData = new IPv6InterfaceData();
   ie->setIPv6Data(ipv6IfData);

   // for routers, turn on advertisements by default
   //FIXME: we will use this isRouter flag for now. what if future implementations
   //have 2 interfaces where one interface is configured as a router and the other
   //as a host?
   ipv6IfData->setAdvSendAdvertisements(isrouter);//Added by WEI

   // metric: some hints: OSPF cost (2e9/bps value), MS KB article Q299540, ...
   //d->setMetric((int)ceil(2e9/ie->datarate())); // use OSPF cost as default
   //FIXME TBD fill in the rest

   assignRequiredNodeAddresses(ie);
}

void RoutingTable6::assignRequiredNodeAddresses(InterfaceEntry *ie)
{
   //RFC 3513 Section 2.8:A Node's Required Addresses
   /*A host is required to recognize the following addresses as
```

```
            identifying itself:*/

            //o  The loopback address.
            if (ie->isLoopback())
            {
               ie->ipv6()->assignAddress(IPv6Address("::1"), false, 0, 0);
               return;
            }
            //o  Its required Link-Local Address for each interface.
            //IPv6Address linkLocalAddr = IPv6Address().formLinkLocalAddress(ie-
>interfaceToken());
            //ie->ipv6()->assignAddress(linkLocalAddr, true, 0, 0);

            /*o  Any additional Unicast and Anycast Addresses that have been configured
            for the node's interfaces (manually or automatically).*/

            // FIXME FIXME Andras: commented out the following lines, because these addresses
            // are implicitly checked for in localDeliver()  (we don't want redundancy,
            // and manually adding solicited-node mcast address for each and every address
            // is very error-prone!)
            //
            //o  The All-Nodes Multicast Addresses defined in section 2.7.1.

            /*o  The Solicited-Node Multicast Address for each of its unicast and anycast
            addresses.*/

            //o  Multicast Addresses of all other groups to which the node belongs.

            /*A router is required to recognize all addresses that a host is
            required to recognize, plus the following addresses as identifying
            itself:*/
            /*o  The Subnet-Router Anycast Addresses for all interfaces for
            which it is configured to act as a router.*/

            //o  All other Anycast Addresses with which the router has been configured.
            //o  The All-Routers Multicast Addresses defined in section 2.7.1.
         }

         void RoutingTable6::configureInterfaceFromXML(InterfaceEntry *ie, cXMLElement *cfg)
         {
```

```
            /*XML parsing capabilities tweaked by WEI. For now, we can configure a specific
            node's interface. We can set advertising prefixes and other variables to be used
            in RAs. The IPv6 interface data gets overwritten if lines 249 to 262 is uncommented.
            The fix is to create an XML file with all the default values. Customised XML files
            can be used for future protocols that requires different values. (MIPv6)*/
            IPv6InterfaceData *d = ie->ipv6();

            // parse basic config (attributes)
            d->setAdvSendAdvertisements(toBool(getRequiredAttr(cfg,
"AdvSendAdvertisements")));
            //TODO: leave this off first!! They overwrite stuff!
          /* TODO: Wei commented out the stuff below. To be checked why (Andras).
            d->setMaxRtrAdvInterval(OPP_Global::atod(getRequiredAttr(cfg,
"MaxRtrAdvInterval")));
            d->setMinRtrAdvInterval(OPP_Global::atod(getRequiredAttr(cfg,
"MinRtrAdvInterval")));
            d->setAdvManagedFlag(toBool(getRequiredAttr(cfg, "AdvManagedFlag")));
            d->setAdvOtherConfigFlag(toBool(getRequiredAttr(cfg, "AdvOtherConfigFlag")));
            d->setAdvLinkMTU(OPP_Global::atoul(getRequiredAttr(cfg, "AdvLinkMTU")));
            d->setAdvReachableTime(OPP_Global::atoul(getRequiredAttr(cfg,
"AdvReachableTime")));
            d->setAdvRetransTimer(OPP_Global::atoul(getRequiredAttr(cfg,
"AdvRetransTimer")));
            d->setAdvCurHopLimit(OPP_Global::atoul(getRequiredAttr(cfg, "AdvCurHopLimit")));
            d->setAdvDefaultLifetime(OPP_Global::atoul(getRequiredAttr(cfg,
"AdvDefaultLifetime")));
            ie->setMtu(OPP_Global::atoul(getRequiredAttr(cfg, "HostLinkMTU")));
            d->setCurHopLimit(OPP_Global::atoul(getRequiredAttr(cfg, "HostCurHopLimit")));
            d->setBaseReachableTime(OPP_Global::atoul(getRequiredAttr(cfg,
"HostBaseReachableTime")));
            d->setRetransTimer(OPP_Global::atoul(getRequiredAttr(cfg, "HostRetransTimer")));
            d->setDupAddrDetectTransmits(OPP_Global::atoul(getRequiredAttr(cfg,
"HostDupAddrDetectTransmits")));
          */

            // parse prefixes (AdvPrefix elements; they should be inside an AdvPrefixList
            // element, but we don't check that)
            cXMLElementList prefixList = cfg->getElementsByTagName("AdvPrefix");
            for (unsigned int i=0; i<prefixList.size(); i++)
            {
```

```
            cXMLElement *node = prefixList[i];
            IPv6InterfaceData::AdvPrefix prefix;

            // FIXME todo implement: advValidLifetime, advPreferredLifetime can
            // store (absolute) expiry time (if >0) or lifetime (delta) (if <0);
            // 0 should be treated as infinity
            int pfxLen;
            if (!prefix.prefix.tryParseAddrWithPrefix(node->getNodeValue(),pfxLen))
               opp_error("element <%s> at %s: wrong IPv6Address/prefix syntax %s",
                     node->getTagName(), node->getSourceLocation(), node->getNodeValue());
            prefix.prefixLength = pfxLen;
            prefix.advValidLifetime = OPP_Global::atoul(getRequiredAttr(node,
"AdvValidLifetime"));
            prefix.advOnLinkFlag = toBool(getRequiredAttr(node, "AdvOnLinkFlag"));
            prefix.advPreferredLifetime = OPP_Global::atoul(getRequiredAttr(node,
"AdvPreferredLifetime"));
            prefix.advAutonomousFlag = toBool(getRequiredAttr(node, "AdvAutonomousFlag"));
            d->addAdvPrefix(prefix);
         }

         // parse addresses
         cXMLElementList addrList = cfg->getChildrenByTagName("inetAddr");
         for (unsigned int k=0; k<addrList.size(); k++)
         {
            cXMLElement *node = addrList[k];
            IPv6Address address(node->getNodeValue());
            //We can now decide if the address is tentative or not.
            d->assignAddress(address, toBool(getRequiredAttr(node, "tentative")), 0, 0);  // set up
with infinite lifetimes
         }
      }

      InterfaceEntry *RoutingTable6::interfaceByAddress(const IPv6Address& addr)
      {
         Enter_Method("interfaceByAddress(%s)=?", addr.str().c_str());

         if (addr.isUnspecified())
            return NULL;
         for (int i=0; i<ift->numInterfaces(); ++i)
         {
```

```
            InterfaceEntry *ie = ift->interfaceAt(i);
            if (ie->ipv6()->hasAddress(addr))
                return ie;
        }
        return NULL;
    }

    bool RoutingTable6::localDeliver(const IPv6Address& dest)
    {
        Enter_Method("localDeliver(%s) y/n", dest.str().c_str());

        // first, check if we have an interface with this address
        for (int i=0; i<ift->numInterfaces(); i++)
        {
            InterfaceEntry *ie = ift->interfaceAt(i);
            if (ie->ipv6()->hasAddress(dest))
                return true;
        }

        // then check for special, preassigned multicast addresses
        // (these addresses occur more rarely than specific interface addresses,
        // that's why we check for them last)

        if (dest==IPv6Address::ALL_NODES_1 || dest==IPv6Address::ALL_NODES_2)
            return true;
        if (isRouter() && (dest==IPv6Address::ALL_ROUTERS_1 ||
dest==IPv6Address::ALL_ROUTERS_2 || dest==IPv6Address::ALL_ROUTERS_5))
            return true;

        // check for solicited-node multicast address
        if (dest.matches(IPv6Address::SOLICITED_NODE_PREFIX, 104))
        {
            for (int i=0; i<ift->numInterfaces(); i++)
            {
                InterfaceEntry *ie = ift->interfaceAt(i);
                if (ie->ipv6()->matchesSolicitedNodeMulticastAddress(dest))
                    return true;
            }
        }
        return false;
```

```
}

const IPv6Address& RoutingTable6::lookupDestCache(const IPv6Address& dest, int&
outInterfaceId)
{
  Enter_Method("lookupDestCache(%s)", dest.str().c_str());

  DestCache::iterator it = destCache.find(dest);
  if (it == destCache.end())
  {
    outInterfaceId = -1;
    return IPv6Address::UNSPECIFIED_ADDRESS;
  }
  outInterfaceId = it->second.interfaceId;
  return it->second.nextHopAddr;
}

const IPv6Route *RoutingTable6::doLongestPrefixMatch(const IPv6Address& dest)
{
  Enter_Method("doLongestPrefixMatch(%s)", dest.str().c_str());

  // we'll just stop at the first match, because the table is sorted
  // by prefix lengths and metric (see addRoute())
  for (RouteList::iterator it=routeList.begin(); it!=routeList.end(); it++)
  {
    if (dest.matches((*it)->destPrefix(),(*it)->prefixLength()))
    {
      // FIXME proofread this code, iterator invalidation-wise, etc
      bool entryExpired = false;
      if (simTime() > (*it)->expiryTime() && (*it)->expiryTime() != 0)//since 0
represents infinity.
      {
        EV << "Expired prefix detected!!" << endl;
        removeOnLinkPrefix((*it)->destPrefix(), (*it)->prefixLength());
        entryExpired = true;
      }
      if (entryExpired == false) return *it;
    }
  }
  // FIXME todo: if we selected an expired route, throw it out and select again!
```

```
    return NULL;
}

bool RoutingTable6::isPrefixPresent(const IPv6Address& prefix)
{
    for (RouteList::iterator it=routeList.begin(); it!=routeList.end(); it++)
        if (prefix.matches((*it)->destPrefix(),128))
            return true;
    return false;
}

void RoutingTable6::updateDestCache(const IPv6Address& dest, const IPv6Address&
nextHopAddr, int interfaceId)
{
    // FIXME this performs 2 lookups -- optimize to do only one
    destCache[dest].nextHopAddr = nextHopAddr;
    destCache[dest].interfaceId = interfaceId;

    updateDisplayString();
}

void RoutingTable6::purgeDestCache()
{
    destCache.clear();
    updateDisplayString();
}

void RoutingTable6::purgeDestCacheEntriesToNeighbour(const IPv6Address&
nextHopAddr, int interfaceId)
{
    for (DestCache::iterator it=destCache.begin(); it!=destCache.end(); )
    {
        if (it->second.interfaceId==interfaceId && it->second.nextHopAddr==nextHopAddr)
        {
            // move the iterator past this element before removing it
            DestCache::iterator oldIt = it++;
            destCache.erase(oldIt);
        }
        else
        {
```

```
            it++;
          }
        }

        updateDisplayString();
      }

      void RoutingTable6::addOrUpdateOnLinkPrefix(const IPv6Address& destPrefix, int
prefixLength,
                                                  int interfaceId, simtime_t expiryTime)
      {
        // see if prefix exists in table
        IPv6Route *route = NULL;
        for (RouteList::iterator it=routeList.begin(); it!=routeList.end(); it++)
        {
          if ((*it)->src()==IPv6Route::FROM_RA && (*it)->destPrefix()==destPrefix &&
(*it)->prefixLength()==prefixLength)
          {
            route = *it;
            break;
          }
        }

        if (route==NULL)
        {
          // create new route object
          IPv6Route *route = new IPv6Route(destPrefix, prefixLength,
IPv6Route::FROM_RA);
          route->setInterfaceID(interfaceId);
          route->setExpiryTime(expiryTime);
          route->setMetric(0);

          // then add it
          addRoute(route);
        }
        else
        {
          // update existing one
          route->setInterfaceID(interfaceId);
          route->setExpiryTime(expiryTime);
```

```
    }

    updateDisplayString();
  }

  void RoutingTable6::addOrUpdateOwnAdvPrefix(const IPv6Address& destPrefix, int
prefixLength,
                                              int interfaceId, simtime_t expiryTime)
  {
    // FIXME this is very similar to the one above -- refactor!!

    // see if prefix exists in table
    IPv6Route *route = NULL;
    for (RouteList::iterator it=routeList.begin(); it!=routeList.end(); it++)
    {
      if ((*it)->src()==IPv6Route::OWN_ADV_PREFIX && (*it)-
>destPrefix()==destPrefix && (*it)->prefixLength()==prefixLength)
      {
        route = *it;
        break;
      }
    }

    if (route==NULL)
    {
      // create new route object
      IPv6Route *route = new IPv6Route(destPrefix, prefixLength,
IPv6Route::OWN_ADV_PREFIX);
      route->setInterfaceID(interfaceId);
      route->setExpiryTime(expiryTime);
      route->setMetric(0);

      // then add it
      addRoute(route);
    }
    else
    {
      // update existing one
      route->setInterfaceID(interfaceId);
      route->setExpiryTime(expiryTime);
```

```
    }

    updateDisplayString();
}

void RoutingTable6::removeOnLinkPrefix(const IPv6Address& destPrefix, int
prefixLength)
{
    // scan the routing table for this prefix and remove it
    for (RouteList::iterator it=routeList.begin(); it!=routeList.end(); it++)
    {
        if ((*it)->src()==IPv6Route::FROM_RA && (*it)->destPrefix()==destPrefix &&
(*it)->prefixLength()==prefixLength)
        {
            routeList.erase(it);
            return; // there can be only one such route, addOrUpdateOnLinkPrefix() guarantees
that
        }
    }

    updateDisplayString();
}

void RoutingTable6::addStaticRoute(const IPv6Address& destPrefix, int prefixLength,
                unsigned int interfaceId, const IPv6Address& nextHop,
                int metric)
{
    // create route object
    IPv6Route *route = new IPv6Route(destPrefix, prefixLength, IPv6Route::STATIC);
    route->setInterfaceID(interfaceId);
    route->setNextHop(nextHop);
    if (metric==0)
    {
        metric = 10; // TBD should be filled from interface metric
    }
    route->setMetric(metric);

    // then add it
    addRoute(route);
}
```

```
void RoutingTable6::addDefaultRoute(const IPv6Address& nextHop, unsigned int ifID,
   simtime_t routerLifetime)
{
   // create route object
   IPv6Route *route = new IPv6Route(IPv6Address(), 0, IPv6Route::FROM_RA);
   route->setInterfaceID(ifID);
   route->setNextHop(nextHop);
   route->setMetric(10);//FIXME:should be filled from interface metric

   // then add it
   addRoute(route);
}

void RoutingTable6::addRoutingProtocolRoute(IPv6Route *route)
{
   ASSERT(route->src()==IPv6Route::ROUTING_PROT);
   addRoute(route);
}

bool RoutingTable6::routeLessThan(const IPv6Route *a, const IPv6Route *b)
{
   // helper for sort() in addRoute(). We want routes with longer
   // prefixes to be at front, so we compare them as "less".
   // For metric, a smaller value is better (we report that as "less").
   if (a->prefixLength()!=b->prefixLength())
      return a->prefixLength() > b->prefixLength();
   return a->metric() < b->metric();
}

void RoutingTable6::addRoute(IPv6Route *route)
{
   routeList.push_back(route);

   // we keep entries sorted by prefix length in routeList, so that we can
   // stop at the first match when doing the longest prefix matching
   std::sort(routeList.begin(), routeList.end(), routeLessThan);

   updateDisplayString();
}
```

```
void RoutingTable6::removeRoute(IPv6Route *route)
{
   RouteList::iterator it = std::find(routeList.begin(), routeList.end(), route);
   ASSERT(it!=routeList.end());
   routeList.erase(it);

   updateDisplayString();
}

int RoutingTable6::numRoutes() const
{
   return routeList.size();
}

IPv6Route *RoutingTable6::route(int i)
{
   ASSERT(i>=0 && i<routeList.size());
   return routeList[i];
}
```

8.12. Arquivo StandardHostHIP6.ned

```
import
    "NotificationBoard",
    "InterfaceTable",
    "RoutingTable6",
    "TCPApp.ned",
    "TCP.ned",
    "UDP",
    "UDPApp",
    "NetworkHIPLayer",
    "PingApp",
    "EthernetInterface";

//
// \IPv6 host with TCP, UDP layers and applications.
//
module StandardHostHIP6
    parameters:
```

```
        numTcpApps: numeric const,
        numUdpApps: numeric const,
        tcpAppType: string,
        udpAppType: string;
    gates:
        in: ethIn[];
        out: ethOut[];
        in: in[];
        out: out[];
    submodules:
        notificationBoard: NotificationBoard;
            display: "p=60,70;i=block/control";
        interfaceTable: InterfaceTable;
            display: "p=60,150;i=block/table";
        routingTable6: RoutingTable6;
            parameters:
                isRouter = false;
            display: "p=60,230;i=block/table";
        tcpApp: tcpAppType[numTcpApps] like TCPApp;
            display: "p=163,67;i=block/app";
        tcp: TCP;
            display: "p=163,154;i=block/wheelbarrow";
        udpApp: udpAppType[numUdpApps] like UDPApp;
            display: "i=block/app;p=272,67";
        udp: UDP;
            display: "p=272,154;i=block/transport";
        pingApp: PingApp;
            display: "i=block/app;p=343,200";
        networkLayer: NetworkHIPLayer;
            gatesizes:
                ifIn[sizeof(ethOut)],
                ifOut[sizeof(ethOut)];
            display: "p=248,247;i=block/fork;q=queue";
        eth: EthernetInterface[sizeof(ethOut)];
            display:
"p=240,350,row,90;q=txQueue;i=block/ifcard";
    connections nocheck:
        for i=0..numTcpApps-1 do
            tcpApp[i].tcpOut --> tcp.from_appl++;
            tcpApp[i].tcpIn <-- tcp.to_appl++;
        endfor;
```

```
        tcp.to_ipv6 --> networkLayer.NHLTCPIn;
        tcp.from_ipv6 <-- networkLayer.NHLTCPOut;

        for i=0..numUdpApps-1 do
            udpApp[i].to_udp --> udp.from_app++;
            udpApp[i].from_udp <-- udp.to_app++;
        endfor;

        udp.to_ipv6 --> networkLayer.NHLUDPIn;
        udp.from_ipv6 <-- networkLayer.NHLUDPOut;

        networkLayer.NHLpingOut --> pingApp.pingIn;
        networkLayer.NHLpingIn <-- pingApp.pingOut;

        // connections to network outside
        for i=0..sizeof(ethOut)-1 do
            ethIn[i] --> eth[i].physIn;
            ethOut[i] <-- eth[i].physOut;
            eth[i].netwOut --> networkLayer.ifIn[i];
            eth[i].netwIn <-- networkLayer.ifOut[i];
        endfor;
endmodule
```

8.13. Arquivo RVS_HIP.ned

```
import
    "NotificationBoard",
    "InterfaceTable",
    "RoutingTable6",
    "NetworkLayer6",
    "HIPRendezvous",
    "PingApp",
    "EthernetInterface";

//
// Rendezvous Server for HIP Protocol.
//
module RVS_HIP
    gates:
```

```
        in: ethIn[];
        out: ethOut[];
        in: in[];
        out: out[];
    submodules:
        notificationBoard: NotificationBoard;
            display: "p=60,70;i=block/control";
        interfaceTable: InterfaceTable;
            display: "p=60,150;i=block/table";
        routingTable6: RoutingTable6;
            parameters:
                isRouter = false;
            display: "p=60,230;i=block/table";
        pingApp: PingApp;
            display: "i=block/app;p=343,200";
        networkLayer: NetworkLayer6;
            gatesizes:
                ifIn[sizeof(ethOut)],
                ifOut[sizeof(ethOut)];
            display: "p=248,247;i=block/fork;q=queue";
        hipRVS: HIPRendezvous;
            display: "p=180,200;i=block/fork;q=queue";
        eth: EthernetInterface[sizeof(ethOut)];
            display:
"p=240,350,row,90;q=txQueue;i=block/ifcard";

    connections nocheck:
        networkLayer.pingOut --> pingApp.pingIn;
        networkLayer.pingIn <-- pingApp.pingOut;

        networkLayer.HIPOut --> hipRVS.hipIPIn;
        networkLayer.HIPIn <-- hipRVS.hipIPOut;

        // connections to network outside
        for i=0..sizeof(ethOut)-1 do
            ethIn[i] --> eth[i].physIn;
            ethOut[i] <-- eth[i].physOut;
            eth[i].netwOut --> networkLayer.ifIn[i];
            eth[i].netwIn <-- networkLayer.ifOut[i];
        endfor;
endmodule
```

8.14. Arquivo NetworkLayer6.ned

```
    import
        "IPv6",
        "IPv6.ned",
        "ICMPv6",
        "IPv6NeighbourDiscovery",
        "IPv6ErrorHandling";

    //
    // Represents an IPv6 network layer (L3).
    //
    // The module has ports to connect to a higher layer
(TCP,UDP) and
    // several network interfaces.
    //
    module NetworkLayer6
        gates:
            in: ifIn[];
            out: ifOut[];
            in: TCPIn;
            out: TCPOut;
            in: UDPIn;
            out: UDPOut;
            in: HIPIn;
            out: HIPOut;
            in: RSVPIn;
            out: RSVPOut;
            in: OSPFIn;
            out: OSPFOut;
            in: pingIn;
            out: pingOut;
            out: errorOut;
        submodules:
            ipv6: IPv6;
                parameters:
                    protocolMapping = "6:0,17:1,46:2,89:3,253:4";
                gatesizes:
                    transportIn[5],
```

```
                transportOut[5],
                queueIn[sizeof(ifIn)],
                queueOut[sizeof(ifIn)];
            display: "i=block/network2;p=84,129";
        icmpv6: ICMPv6;
            display: "i=block/control;p=208,79";
        ipv6ErrorHandling: IPv6ErrorHandling;
            display: "p=296,79;i=block/process_s";
        neighbourDiscovery: IPv6NeighbourDiscovery;
            display: "p=208,171;i=block/network";
    connections nocheck: // FIXME remove 'nocheck'!

        ipv6.transportOut[0] --> TCPOut display "m=n";
        ipv6.transportIn[0] <-- TCPIn display "m=n";

        ipv6.transportOut[1] --> UDPOut display "m=n";
        ipv6.transportIn[1] <-- UDPIn display "m=n";

        ipv6.transportOut[2] --> RSVPOut display "m=n";
        ipv6.transportIn[2] <-- RSVPIn display "m=n";

        ipv6.transportOut[3] --> OSPFOut display "m=n";
        ipv6.transportIn[3] <-- OSPFIn display "m=n";

        ipv6.transportOut[4] --> HIPOut display "m=n";
        ipv6.transportIn[4] <-- HIPIn display "m=n";

        // IPv6 to ICMPv6
        ipv6.icmpOut --> icmpv6.fromIPv6;
        ipv6.icmpIn <-- icmpv6.toIPv6;

        // ICMPv6 to IPv6ErrorHandling
        icmpv6.errorOut --> ipv6ErrorHandling.in;

        // ICMPv6 to ping I/O
        icmpv6.pingOut --> pingOut;
        icmpv6.pingIn <-- pingIn;

        // IPv6 to Neighbour Discovery
        ipv6.ndOut --> neighbourDiscovery.fromIPv6;
        ipv6.ndIn <-- neighbourDiscovery.toIPv6;
```

```
        // IPv6 to L2
        for i=0..sizeof(ifOut)-1 do
            ifIn[i] --> ipv6.queueIn[i] display "m=s";
            ifOut[i] <-- ipv6.queueOut[i] display "m=s";
        endfor;
endmodule
```

8.15. Arquivo NetworkHIPLayer.ned

```
import
    "NetworkLayer6",
    "HIP.ned"; //

//
// IPv6Layer with HIP.
//
module NetworkHIPLayer
    gates:
        in: ifIn[];
        out: ifOut[];
        in: NHLRSVPIn;
        out: NHLRSVPOut;
        in: NHLOSPFIn;
        out: NHLOSPFOut;
        in: NHLpingIn;
        out: NHLpingOut;
        out: NHLerrorOut;
        in: NHLUDPIn;
        out: NHLUDPOut;
        in: NHLTCPIn;
        out: NHLTCPOut;

    submodules:
        networkLayer: NetworkLayer6;
            gatesizes:
                ifIn[sizeof(ifIn)],
                ifOut[sizeof(ifOut)];
            display: "p=248,247;i=block/fork;q=queue";
 hip: HIP;
```

```
            display: "p=105,149;i=block/network2";
        connections nocheck:
            for i=0..sizeof(ifOut)-1 do
                ifIn[i] --> networkLayer.ifIn[i] display "m=s";
                ifOut[i] <-- networkLayer.ifOut[i] display "m=s";
            endfor;

            NHLpingIn --> networkLayer.pingIn;
            NHLpingOut <-- networkLayer.pingOut;

            NHLRSVPIn --> networkLayer.RSVPIn;
            NHLRSVPOut <-- networkLayer.RSVPOut;

            NHLOSPFIn --> networkLayer.OSPFIn;
            NHLOSPFOut <-- networkLayer.OSPFOut;

            NHLerrorOut <-- networkLayer.errorOut;

     networkLayer.TCPOut --> hip.hipIPTcpIn;
     networkLayer.TCPIn <-- hip.hipIPTcpOut;

     networkLayer.UDPOut --> hip.hipIPUdpIn;
     networkLayer.UDPIn <-- hip.hipIPUdpOut;

     networkLayer.HIPOut --> hip.hipIPIn;
     networkLayer.HIPIn <-- hip.hipIPOut;

     NHLTCPOut <-- hip.hipTcpOut;
     NHLTCPIn --> hip.hipTcpIn;

     NHLUDPOut <-- hip.hipUdpOut;
     NHLUDPIn --> hip.hipUdpIn;

    Endmodule
```

8.16. Arquivo WirelessHIPHost6.ned

```
import
    "NotificationBoard",
    "InterfaceTable",
```

```
        "RoutingTable6",
        "TCPApp.ned",
        "TCP.ned",
        "UDP",
        "UDPApp",
        "NetworkHIPLayer",
        "PingApp",
        "PPPInterface",
        "EthernetInterface",
        "Ieee80211NicSTA";


    //
    // Models a host with one wireless (802.11b) card in
infrastructure mode,
    // This module is basically a StandardHost with an
Ieee80211NicSTASimplified
    // added. It should be used in conjunction with
WirelessAPSimplified,
    // or any other AP model which contains
Ieee80211NicAPSimplified.
    //
    // @see WirelessAP, WirelessAPSimplified, WirelessAPWithEth,
WirelessAPWithEthSimplified
    // @see WirelessHost, WirelessHostSimplified
    // @see MobileHost, MFMobileHost
    //
    module WirelessHIPHost6
        parameters:
            numTcpApps: numeric const,
            numUdpApps: numeric const,
            tcpAppType: string,
            udpAppType: string,
            routingFile: string,
            mobilityType: string;
        gates:
            in: in[];
            out: out[];
            in: ethIn[];
            out: ethOut[];
            in: radioIn;
```

```
        submodules:
            notificationBoard: NotificationBoard;
                display: "p=60,70;i=block/control";
            interfaceTable: InterfaceTable;
                display: "p=60,150;i=block/table";
            routingTable6: RoutingTable6;
                display: "p=60,230;i=block/table";
            tcpApp: tcpAppType[numTcpApps] like TCPApp;
                display: "p=163,67;i=block/app";
            tcp: TCP;
                display: "p=163,154;i=block/wheelbarrow";
            udpApp: udpAppType[numUdpApps] like UDPApp;
                display: "i=block/app;p=272,67";
            udp: UDP;
                display: "p=272,154;i=block/transport";
            pingApp: PingApp;
                display: "i=block/app;p=343,200";
            networkLayer: NetworkHIPLayer;
                gatesizes:
                    ifIn[sizeof(out)+sizeof(ethOut)+1],
                    ifOut[sizeof(out)+sizeof(ethOut)+1];
                display: "p=248,247;i=block/fork;q=queue";
            ppp: PPPInterface[sizeof(out)];
                display:
"p=205,350,row,90;q=txQueue;i=block/ifcard";
            eth: EthernetInterface[sizeof(ethOut)];
                display:
"p=240,350,row,90;q=txQueue;i=block/ifcard";
            wlan: Ieee80211NicSTA;
                display: "p=120,350;q=queue;i=block/ifcard";
            mobility: mobilityType like BasicMobility;
                display: "p=58,301;i=block/cogwheel_s";
        connections nocheck:
            for i=0..numTcpApps-1 do
                tcpApp[i].tcpOut --> tcp.from_appl++;
                tcpApp[i].tcpIn <-- tcp.to_appl++;
            endfor;

            tcp.to_ipv6 --> networkLayer.NHLTCPIn;
            tcp.from_ipv6 <-- networkLayer.NHLTCPOut;
```

```
            for i=0..numUdpApps-1 do
                udpApp[i].to_udp --> udp.from_app++;
                udpApp[i].from_udp <-- udp.to_app++;
            endfor;

            udp.to_ipv6 --> networkLayer.NHLUDPIn;
            udp.from_ipv6 <-- networkLayer.NHLUDPOut;

            networkLayer.NHLpingOut --> pingApp.pingv6In;
            networkLayer.NHLpingIn <-- pingApp.pingv6Out;

            // connections to network outside
            for i=0..sizeof(out)-1 do
                in[i] --> ppp[i].physIn;
                out[i] <-- ppp[i].physOut;
                ppp[i].netwOut --> networkLayer.ifIn[i];
                ppp[i].netwIn <-- networkLayer.ifOut[i];
            endfor;

            for i=0..sizeof(ethOut)-1 do
                ethIn[i] --> eth[i].physIn;
                ethOut[i] <-- eth[i].physOut;
                eth[i].netwOut -->
networkLayer.ifIn[sizeof(out)+i];
                eth[i].netwIn <--
networkLayer.ifOut[sizeof(out)+i];
            endfor;

            radioIn --> wlan.radioIn;
            wlan.uppergateOut -->
networkLayer.ifIn[sizeof(out)+sizeof(ethOut)];
            wlan.uppergateIn <--
networkLayer.ifOut[sizeof(out)+sizeof(ethOut)];
    endmodule
```

8.17. Arquivo netconf2.dtd

```
    <?xml encoding="ISO-8859-1"?>
    <!ELEMENT netconf (global?, local*, misc?)>
    <!ATTLIST netconf
```

```
                version NMTOKEN "0.0.5"
                debugChannel NMTOKEN #IMPLIED
                >

        <!ELEMENT global (WirelessEtherInfo?, ObjectMovement?)>
        <!ATTLIST global
                gSendAdvertisements (on|off) "off"
                gMaxRtrAdvInterval NMTOKEN #IMPLIED
                gMinRtrAdvInterval NMTOKEN #IMPLIED
                gAdvManagedFlag (on|off) "off"
                gAdvOtherConfigFlag (on|off) "off"
                gAdvLinkMTU NMTOKEN #IMPLIED
                gAdvReachableTime NMTOKEN #IMPLIED
                gAdvRetransTimer NMTOKEN #IMPLIED
                gAdvCurHopLimit NMTOKEN #IMPLIED
                gAdvDefaultLifetime NMTOKEN #IMPLIED
                gHostLinkMTU NMTOKEN #IMPLIED
                gHostCurHopLimit NMTOKEN #IMPLIED
                gHostBaseReachableTime NMTOKEN #IMPLIED
                gHostRetransTimer NMTOKEN #IMPLIED
                gHostDupAddrDetectTransmits NMTOKEN #IMPLIED
                >

        <!-- Should all these be attributes? If they can't why aren't
the elements -->
        <!-- shared with the per interface ones? -->
        <!ELEMENT gSendAdvertisements (#PCDATA)>
        <!ELEMENT gMaxRtrAdvInterval (#PCDATA)>
        <!ELEMENT gMinRtrAdvInterval (#PCDATA)>
        <!ELEMENT gAdvManagedFlag (#PCDATA)>
        <!ELEMENT gAdvOtherConfigFlag (#PCDATA)>
        <!ELEMENT gAdvLinkMTU (#PCDATA)>
        <!ELEMENT gAdvReachableTime (#PCDATA)>
        <!ELEMENT gAdvRetransTimer (#PCDATA)>
        <!ELEMENT gAdvCurHopLimit (#PCDATA)>
        <!ELEMENT gAdvDefaultLifetime (#PCDATA)>
        <!ELEMENT gHostLinkMTU (#PCDATA)>
        <!ELEMENT gHostCurHopLimit (#PCDATA)>
        <!ELEMENT gHostBaseReachableTime (#PCDATA)>
        <!ELEMENT gHostRetransTimer (#PCDATA)>
        <!ELEMENT gHostDupAddrDetectTransmits (#PCDATA)>
```

```
    <!ELEMENT local (interface*, (route?| tunnel?|
sourceRoute?|mobileConfig?)*)>
    <!ATTLIST local
              node NMTOKEN #REQUIRED
              routePackets (on|off) "off"
              forwardSitePackets (on|off) "on"
              optimisticDAD (on|off) "off"
              mobileIPv6Support (on|off) "off"
              mobileIPv6Role (None|HomeAgent|MobileNode) "None"
              routeOptimisation (on|off) "on"
              returnRoutability (on|off) "off"
              earlyBU (on|off) "off"
       signalingEnhance (None|Direct|CellResidency) "None"
              respondBindingRequest (on|off) "off"
              eagerHandover (on|off) "off"
              hierarchicalMIPv6Support (on|off) "off"
              map (on|off) "off"
              mapMode (Basic|Extended) "Basic"
              mapMNMAYSetRoCAAsSource (on|off) "on"
              mapMNMUSTSetRoCAAsSource (on|off) "on"
              mapReverseTunnel (on|off) "on"
              edgeHandoverType CDATA ""
              ewuOutVectorHODelays (on|off) "off"
              >

    <!ELEMENT route (routeEntry)+>
    <!ELEMENT routeEntry EMPTY>
    <!ATTLIST routeEntry
              routeIface NMTOKEN #REQUIRED
              routeDestination CDATA #REQUIRED
              routePrefix CDATA #REQUIRED
              routeNextHop CDATA ""
              isRouter (on|off) "off"
              >

    <!ELEMENT sourceRoute (sourceRouteEntry)+>
    <!ELEMENT sourceRouteEntry (nextHop)+>
    <!ATTLIST sourceRouteEntry finalDestination CDATA #REQUIRED>
    <!ELEMENT nextHop EMPTY>
    <!ATTLIST nextHop address CDATA #REQUIRED>
```

```
    <!ELEMENT tunnel (tunnelEntry)+>
    <!ELEMENT tunnelEntry  (triggers)*>
    <!ATTLIST tunnelEntry
              exitIface NMTOKEN #REQUIRED
              entryPoint CDATA #REQUIRED
              exitPoint  CDATA #REQUIRED
              >
    <!ELEMENT triggers EMPTY>
    <!ATTLIST triggers destination CDATA #REQUIRED>

    <!ELEMENT mobileConfig (homeAgent|mobileNode)>
    <!ELEMENT homeAgent EMPTY>
    <!ELEMENT mobileNode (staticConfig?)>

    <!-- Used when mobile node is away from home during node
startup -->
    <!-- homeAgentAddr is the home agent's globally routable
address with prefix specified. The home address is formed from
this prefix on homeAddrIface. This can simply be the home prefix
if the bits after prefix are 0 for testing of DHAAD -->
    <!-- homeAddrIface references one of the interface elements
of this local node by its name on this interface that has a route
path to the home agent and will be where the home address is
actually bound to -->

    <!ELEMENT staticConfig EMPTY>
    <!ATTLIST staticConfig
              homeAddrIface NMTOKEN #REQUIRED
              homeAgentAddr CDATA #REQUIRED
              >

    <!ELEMENT interface (inetAddr*, hipAddr*, AdvPrefixList?,
AdvMAPList?, WirelessEtherInfo?)>

    <!-- Require non ID type for name attribute as that requires
each "interface
    name" to be unique across whole xml document not just inside
interface
    element. -->
```

```
    <!-- Use of MIPv6MaxRtrAdvInterval and MIPv6MinRtrAdvInterval
when mobileIPv6Support is on -->
    <!-- minRtrSolInterval is the relaxed Neighbour Discovery rtr
solicitation interval-->
    <!-- maxInterval is the maximum interval at which no further
rtr sol attempts are made -->
    <!ATTLIST interface
              name NMTOKEN #IMPLIED
              AdvSendAdvertisements (on|off) "on"
              AdvHomeAgent (on|off) "off"
              MaxRtrAdvInterval NMTOKEN "600"
              MinRtrAdvInterval NMTOKEN "198"
              MIPv6MaxRtrAdvInterval NMTOKEN "1.5"
              MIPv6MinRtrAdvInterval NMTOKEN "1"
              MIPv6MinRtrSolInterval NMTOKEN "1"
              MIPv6MaxInterval NMTOKEN "8"
              MaxFastRAS       NMTOKEN "10"
              MaxConsecMissRtrAdv NMTOKEN "1"
              >

    <!-- Suggested additions to phase out elements (non
conformant default values)
    UNPARSED -->
    <!-- Do not use hexadecimal values as XML parsing does not
cater for this
    AdvDefaultLifetime NMTOKEN "0xFFFFFF"
    -->
     <!ATTLIST interface
              AdvManagedFlag (on|off) "off"
              AdvOtherConfigFlag (on|off) "off"
              AdvLinkMTU NMTOKEN "16000"
              AdvReachableTime NMTOKEN "2"
              AdvRetransTimer NMTOKEN "6"
              AdvCurHopLimit NMTOKEN "32"
              AdvDefaultLifetime NMTOKEN "1800"
              HostLinkMTU NMTOKEN "16000"
              HostCurHopLimit NMTOKEN "32"
              HostBaseReachableTime NMTOKEN "40"
              HostRetransTimer NMTOKEN "120"
              HostDupAddrDetectTransmits NMTOKEN "2"
              HostMaxRtrSolDelay NMTOKEN "1"
```

```
                HMIPAdvMAP (on|off) "off"
                HMIPForwardMAPOption (on|off) "off"
                >

      <!ELEMENT inetAddr (#PCDATA)>
      <!ATTLIST inetAddr
               tentative (on|off) "off"
               >
      <!ELEMENT hipAddr (#PCDATA)>
      <!ELEMENT AdvPrefixList (AdvPrefix)+>
      <!ELEMENT AdvPrefix (#PCDATA)>
      <!ATTLIST AdvPrefix
                AdvValidLifetime NMTOKEN "2592000"
                AdvOnLinkFlag (on|off) "off"
                AdvPreferredLifetime NMTOKEN "605800"
                AdvAutonomousFlag (on|off) "on"
                AdvRtrAddrFlag (on|off) "off"
                >
      <!ELEMENT AdvMAPList (AdvMAPEntry)+>
      <!ELEMENT AdvMAPEntry (#PCDATA)>
      <!ATTLIST AdvMAPEntry
                AdvMAPDist NMTOKEN "1"
                AdvMAPPref NMTOKEN "1"
                AdvMAPValidLifetime NMTOKEN "2592"
                >

      <!-- These elements will be deprecated although there is no
other alternative at this time -->
      <!ELEMENT AdvManagedFlag (#PCDATA)>
      <!ELEMENT AdvOtherConfigFlag (#PCDATA)>
      <!ELEMENT AdvLinkMTU (#PCDATA)>
      <!ELEMENT AdvReachableTime (#PCDATA)>
      <!ELEMENT AdvRetransTimer (#PCDATA)>
      <!ELEMENT AdvCurHopLimit (#PCDATA)>
      <!ELEMENT AdvDefaultLifetime (#PCDATA)>

      <!ELEMENT HostLinkMTU (#PCDATA)>
      <!ELEMENT HostCurHopLimit (#PCDATA)>
      <!ELEMENT HostBaseReachableTime (#PCDATA)>
      <!ELEMENT HostRetransTimer (#PCDATA)>
      <!ELEMENT HostDupAddrDetectTransmits (#PCDATA)>
```

```
    <!-- Misc. options. Any conditions preset for the behaviour -
->
    <!-- the entire network. and they are part of IP
configuration -->

    <!ELEMENT misc (ObjectMovement?)>

    <!ELEMENT ObjectMovement (MovingNode*, RandomMovement*,
RandomPattern*)>
    <!ATTLIST MovingNode
              NodeName NMTOKEN #REQUIRED
              startTime NMTOKEN #REQUIRED
              >

    <!ELEMENT MovingNode (move*)>
    <!ELEMENT move EMPTY>
    <!ATTLIST move
              moveToX NMTOKEN #REQUIRED
              moveToY NMTOKEN #REQUIRED
              moveSpeed NMTOKEN #REQUIRED
              >

    <!ELEMENT RandomMovement (#PCDATA)>
    <!ATTLIST RandomMovement
              RWNodeName NMTOKEN #REQUIRED
              RWMoveKind (toroidal|bouncing) "bouncing"
              RWMinX NMTOKEN #REQUIRED
              RWMaxX NMTOKEN #REQUIRED
              RWMinY NMTOKEN #REQUIRED
              RWMaxY NMTOKEN #REQUIRED
              RWMoveInterval NMTOKEN #REQUIRED
              RWMinSpeed NMTOKEN #REQUIRED
        RWMaxSpeed NMTOKEN #REQUIRED
        RWDistance NMTOKEN #REQUIRED
        RWPauseTime NMTOKEN #REQUIRED
        RWStartTime NMTOKEN "5"
              >

    <!ELEMENT RandomPattern (RPNode*)>
    <!ATTLIST RandomPattern
```

```
          RPXSize NMTOKEN #REQUIRED
          RPYSize NMTOKEN #REQUIRED
          RPMoveInterval NMTOKEN #REQUIRED
          RPMinSpeed NMTOKEN #REQUIRED
   RPMaxSpeed NMTOKEN #REQUIRED
   RPDistance NMTOKEN #REQUIRED
   RPPauseTime NMTOKEN #REQUIRED
          >

<!ELEMENT RPNode EMPTY>
<!ATTLIST RPNode
          RPNodeName NMTOKEN #REQUIRED
          RPXOffset NMTOKEN #REQUIRED
          RPYOffset NMTOKEN #REQUIRED
          >
```

8.18. Arquivo UDPAppOnOff.ned

```
simple UDPAppOnOff
 parameters:
   startTime: numeric, // the time first packet
   average_distrib_on: numeric, // may be a random value
   average_distrib_off: numeric, // may be a random value
   packetInterval : numeric, // may be a random value
   numPackets: numeric const, // max number
   local_port: numeric const,
   dest_port: numeric const,
   packetLength : numeric, // (bits)
   file: string,
   dest_addresses: string; // list of IP addresses
  gates:
    in: from_udp;
    out: to_udp;
endsimple
```

8.19. Arquivo UDPAppOnOff.h

```
#ifndef __UDPAPPONOFF_H__
#define __UDPAPPONOFF_H__

#include <vector>
```

```
#include <omnetpp.h>
#include "UDPAppBase.h"
#include <iostream>
#include <fstream>

class UDPAppOnOff : public UDPAppBase
{
  public:
    // Stores information about end-to-end UDP packets
    struct ResultEndToEnd
    {
      unsigned long sequence;   ///< packet sequence number
      simtime_t startTime;      ///< packet creation time
      simtime_t endTime;        ///< packet reception time
    };

  protected:
    typedef std::vector<ResultEndToEnd *>
ResultEndToEndVector;
    ResultEndToEndVector myVector;
    std::string nodeName;
    const char* fileName;
    int localPort, destPort;
    int protocol;
    int msgLength;
    int numPackets;
    double average_on;
    double average_off;
    simtime_t start;
    simtime_t limitOnTime;
    std::vector<IPvXAddress> destAddresses;
    unsigned long counter; // counter for generating a global
number for each packet
    int numSent;
    int numReceived;

    // chooses random destination address
    virtual IPvXAddress chooseDestAddr();
    virtual void sendPacket();
    virtual void processPacket(cMessage *msg);
```

```
  public:
    //Module_Class_Members(UDPAppOnOff, UDPAppBase, 0);
    ~UDPAppOnOff();
    virtual int numInitStages() const {return 4;};
    virtual void initialize(int stage);
    virtual void handleMessage(cMessage *msg);
    void addMyVector(unsigned long Seq, simtime_t time1,
simtime_t time2);

  private:
    // statistics
    cOutVector OnOffVector;
    simtime_t origTime, destTime;
    unsigned long numSeq;
    std::ofstream myfile;

};
#endif
```

8.20. Arquivo UDPAppOnOff.cc

```
#include <omnetpp.h>
#include "UDPAppOnOff.h"
#include "UDPControlInfo_m.h"
#include "IPAddressResolver.h"
#include "UDPVideoStreamMsg_m.h"

Define_Module(UDPAppOnOff);

UDPAppOnOff::~UDPAppOnOff()
{
   myfile.open (fileName);
   myfile << "NÃºmero da Mensagem ; Tempo de criaÃ§Ã£o da
mensagem ; Tempo de entrega da mensagem \n";

   for (unsigned int i=0; i<myVector.size(); i++)
   {
      myfile << myVector[i]->sequence << " ; " <<
myVector[i]->startTime << " ; " << myVector[i]->endTime << " \n";
   }
```

```
        myfile.close();

        for (unsigned int i=0; i<myVector.size(); i++)
             delete myVector[i];
     }


     void UDPAppOnOff::initialize(int stage)
     {
         // because of IPAddressResolver, we need to wait until
interfaces are registered,
         // address auto-assignment takes place etc.
         if (stage!=3)
             return;

         counter = 0;
         numSent = 0;
         numReceived = 0;
         WATCH(numSent);
         WATCH(numReceived);

         localPort = par("local_port");
         destPort = par("dest_port");
         msgLength = par("packetLength");
         numPackets = par("numPackets");
         average_on = par("average_distrib_on");
         average_off = par("average_distrib_off");
         limitOnTime = 0;
         fileName=par("file");
         //std::cout << "Average On= " << average_on << " and Off
Average= " << average_off << endl;
         //std::cout << "1o Limit On Period= " << limitOnTime <<
endl;

         const char *destAddrs = par("dest_addresses");
         cStringTokenizer tokenizer(destAddrs);
         const char *token;
         while ((token = tokenizer.nextToken())!=NULL)

destAddresses.push_back(IPAddressResolver().resolve(token));
```

```
        if (destAddresses.empty())
            return;

        bindToPort(localPort);

        cMessage *timer = new cMessage("sendTimer");
        scheduleAt(double(par("startTime")), timer);
    }

    IPvXAddress UDPAppOnOff::chooseDestAddr()
    {
        int k = intrand(destAddresses.size());
        return destAddresses[k];
    }

    void UDPAppOnOff::sendPacket()
    {
        char msgName[32];
        sprintf(msgName,"UDPAppOnOffData-%d", counter++);

        //cMessage *payload = new cMessage(msgName);
        UDPVideoStreamMsg *payload = new
UDPVideoStreamMsg(msgName);
        payload->setSeq(counter);
        payload->setTimestamp();
        payload->setLength(msgLength);
        IPvXAddress destAddr = chooseDestAddr();
        sendToUDP(payload, localPort, destAddr, destPort);

        numSent++;
    }

    void UDPAppOnOff::handleMessage(cMessage *msg)
    {
        if (msg->isSelfMessage())
        {
       simtime_t actualAverageOff;
       ev<<"Self-message of UDPAppOnOff."<< endl;
       if (counter == 0) { limitOnTime = (double)par("startTime")
+ average_on; }
```

```
              //std::cout <<  "Limit of On Period is: " <<
limitOnTime << " and Simulation time is: " << simTime() << endl;
        if (simTime() <= limitOnTime)
        {
          ev << " Period on" << endl;
               // send, then reschedule next sending
                 sendPacket();

scheduleAt((simTime()+(double)par("packetInterval")), msg);
          actualAverageOff = average_off;
        } else {
                 ev << " Period off" << endl;
          average_on = par("average_distrib_on");
                 scheduleAt((simTime() + actualAverageOff) , msg);
          limitOnTime = simTime() + actualAverageOff + average_on;
            ev << " MÃ©dia On: " << average_on << " Next Pckt: "
<< simTime()+actualAverageOff << " End of new Period On: " <<
limitOnTime << endl;
                 average_off = par("average_distrib_off");
               }
        //std::cout << "Valor de LimitOnTime: " << limitOnTime <<
endl;
         } else {
      ev << "Processing incoming packet of UDPAppOnOff \n";
             // process incoming packet
             processPacket(msg);
         }

         if (ev.isGUI())
         {
             char buf[40];
             sprintf(buf, "rcvd: %d pks\nsent: %d pks",
numReceived, numSent);
             displayString().setTagArg("t",0,buf);
         }
     }

     void UDPAppOnOff::processPacket(cMessage *msg)
     {
         UDPVideoStreamMsg *packet =
check_and_cast<UDPVideoStreamMsg *>(msg);
```

```
        origTime = packet->creationTime();
        numSeq = packet->seq();
        destTime = simTime();
        addMyVector(numSeq, origTime, destTime);

        EV << "On/Off packet arrived:\n";
        printPacket(msg);
        OnOffVector.record(simTime() - msg->creationTime());
        delete msg;

        numReceived++;
    }

    void UDPAppOnOff::addMyVector( unsigned long Seq,  simtime_t
time1,  simtime_t time2)
    {
        ResultEndToEnd *conn = new ResultEndToEnd;
        conn->sequence = Seq;
        conn->startTime = time1;
        conn->endTime = time2;
        myVector.push_back(conn);
    }
```

8.21. Arquivo UDPSink.ned

```
    simple UDPSink
        parameters:
            local_port : numeric const,
            file: string;
        gates:
            in: from_udp;
            out: to_udp;
    endsimple
```

8.22. Arquivo UPDSink.h

```
    #ifndef __UDPSINK_H__
    #define __UDPSINK_H__

    #include <omnetpp.h>
```

```
#include "UDPAppBase.h"
#include <iostream>
#include <fstream>

/**
 * Consumes and prints packets received from the UDP module.
See NED for more info.
 */
class INET_API UDPSink : public UDPAppBase
{
  public:
    // Stores information about end-to-end UDP packets
    struct ResultEndToEnd
    {
      unsigned long sequence;   ///< packet sequence number
      simtime_t startTime;      ///< packet creation time
      simtime_t endTime;        ///< packet reception time
    };

  protected:
    typedef std::vector<ResultEndToEnd *>
ResultEndToEndVector;
    ResultEndToEndVector myVector;
    int numReceived;
    const char* fileName;

  protected:
    virtual void processPacket(cMessage *msg);

  protected:
    virtual void initialize();
    virtual void finish();
    virtual void handleMessage(cMessage *msg);
    void addMyVector(unsigned long Seq, simtime_t time1,
simtime_t time2);


  private:
    simtime_t origTime, destTime;
    unsigned long numSeq;
    std::ofstream myfile;
```

```
};
#endif
```

8.23. Arquivo UDPSink.cc

```
#include <omnetpp.h>
#include "UDPSink.h"
#include "UDPControlInfo_m.h"
#include "IPAddressResolver.h"
#include "UDPVideoStreamMsg_m.h"

Define_Module(UDPSink);

void UDPSink::finish()
{
   myfile.open (fileName);
   myfile << "NÃºmero da Mensagem ; Tempo de criaÃ§Ã£o da
mensagem ; Tempo de entrega da mensagem \n";

   for (unsigned int i=0; i<myVector.size(); i++)
   {
      myfile << myVector[i]->sequence << " ; " <<
myVector[i]->startTime << " ; " << myVector[i]->endTime << " \n";
   }

   myfile.close();
}

void UDPSink::initialize()
{
    numReceived = 0;
    WATCH(numReceived);

    int port = par("local_port");
    fileName = par("file");

    if (port!=-1)
        bindToPort(port);
}
```

```
    void UDPSink::handleMessage(cMessage *msg)
    {
        processPacket(msg);

        if (ev.isGUI())
        {
            char buf[32];
            sprintf(buf, "rcvd: %d pks", numReceived);
            displayString().setTagArg("t",0,buf);
        }

    }

    void UDPSink::processPacket(cMessage *msg)
    {
        UDPVideoStreamMsg *packet =
check_and_cast<UDPVideoStreamMsg *>(msg);
        origTime = packet->creationTime();
        numSeq = packet->seq();
        destTime = simTime();
        addMyVector(numSeq, origTime, destTime);

        EV << "Received packet: ";
        printPacket(msg);
        delete msg;

        numReceived++;
    }

    void UDPSink::addMyVector( unsigned long Seq,  simtime_t
time1,  simtime_t time2)
    {
        ResultEndToEnd *conn = new ResultEndToEnd;
        conn->sequence = Seq;
        conn->startTime = time1;
        conn->endTime = time2;
        myVector.push_back(conn);
    }
```

8.24.
Arquivo WirelessIPv6HIPTese.ini

```
[General]
preload-ned-files = *.ned @../../../nedfiles.lst
debug-on-errors = false
sim-time-limit = 140

network = handoverTeseNetwork

[Cmdenv]
express-mode = yes

[Tkenv]
plugin-path=../../../Etc/plugins
default-run=1

[Parameters]
# number of client computers
*.n=1

# General parameters for mobility
*.playgroundSizeX = 850
*.playgroundSizeY = 450
**.debug = true
**.coreDebug = 0
**.mobility.x = -1
**.mobility.y = -1

# channel physical parameters
*.channelcontrol.carrierFrequency = 2.4e+9
*.channelcontrol.pMax  = 1.0 ;[mW]
*.channelcontrol.sat   = -110
*.channelcontrol.alpha = 2
*.channelcontrol.numChannels = 12

# access point
**.ap1.wlan.mac.address = "10:00:00:00:00:00"
**.ap2.wlan.mac.address = "20:00:00:00:00:00"
**.ap3.wlan.mac.address = "30:00:00:00:00:00"
**.ap4.wlan.mac.address = "40:00:00:00:00:00"
```

```
**.ap1.wlan.mgmt.ssid = "AP1"
**.ap2.wlan.mgmt.ssid = "AP2"
**.ap3.wlan.mgmt.ssid = "AP3"
**.ap4.wlan.mgmt.ssid = "AP4"
**.ap*.wlan.mgmt.beaconInterval = 0.1
**.wlan.mgmt.numAuthSteps = 4

**.mgmt.frameCapacity = 10

# mobility
**.host*.mobilityType = "RectangleMobility"
**.host*.mobility.debug = true
**.host*.mobility.x1 = 45
**.host*.mobility.y1 = 355
**.host*.mobility.x2 = 825
**.host*.mobility.y2 = 375
**.host*.mobility.startPos = 0
**.host*.mobility.speed = 5
**.host*.mobility.updateInterval = 0.1

**.mn*.mobilityType = "RandomWPMobility"
**.mn*.mobility.debug = true
**.mn*.mobility.x = -1
**.mn*.mobility.y = -1
**.mn*.mobility.updateInterval = 0.1
**.mn*.mobility.speed = uniform(3,5)
**.mn*.mobility.waitTime = 2

# configurator
*.configurator.useTentativeAddrs=false # FIXME TBD to be
switched to true, for testing DAD!

# tcp apps
**.numTcpApps=0
**.tcpAppType= "TelnetApp"

# ping app (off)
**.pingApp.destAddr=""
**.pingApp.srcAddr=""
**.pingApp.packetSize=56
**.pingApp.interval=1
```

```
**.pingApp.hopLimit=32
**.pingApp.count=0
**.pingApp.startTime=1
**.pingApp.stopTime=0
**.pingApp.printPing=true

# tcp settings
**.tcp.mss = 1024
**.tcp.advertisedWindow = 14336  # 14*mss
**.tcp.sendQueueClass="TCPMsgBasedSendQueue"
**.tcp.receiveQueueClass="TCPMsgBasedRcvQueue"
**.tcp.tcpAlgorithmClass="TCPReno"
**.tcp.recordStats=true

# ip settings
#FIXME
**.routingTableFile=xmldoc("testeTese.xml")
**.ipv6.procDelay=10us
**.hip.procDelay=0
**.hipRVS.procDelay=0
**.hip.rvsHITAddress="50:50:50:50:532:ff4a:fe59:1256"

**.host0.routingFile=""
**.host0.routingTable6.isRouter=0
**.mn*.routingFile=""
**.mn*.routingTable6.isRouter=0

**.host0.networkLayer.hip.DNSFile=xmldoc("dnsHost0.xml")
**.host0.networkLayer.hip.hipAddress="50:50:50:50:2580:ebff:f
e8c:1448"

**.mn0.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
**.mn0.networkLayer.hip.hipAddress="50:50:50:50:AE12:3579:F13
B:B421"

**.mn1.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
**.mn1.networkLayer.hip.hipAddress="50:50:50:50:1234:261:EE18
:5148"

**.mn2.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
```

```
    **.mn2.networkLayer.hip.hipAddress="50:50:50:50:1111:ABCD:246
8:EF0A"

    **.mn3.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
    **.mn3.networkLayer.hip.hipAddress="50:50:50:50:5634:EF29:AA7
6:E456"

    **.mn4.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
    **.mn4.networkLayer.hip.hipAddress="50:50:50:50:0:439A:F67D:A
A32"

    **.mn5.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
    **.mn5.networkLayer.hip.hipAddress="50:50:50:50:AA32:FF19:372
8:E27B"

    **.mn6.networkLayer.hip.DNSFile=xmldoc("dnsMN.xml")
    **.mn6.networkLayer.hip.hipAddress="50:50:50:50:F67D:B340:228
8:AA23"

    **.mn7.networkLayer.hip.DNSFile=xmldoc("dnsMN10.xml")
    **.mn7.networkLayer.hip.hipAddress="50:50:50:50:4567:0:0:AAE8
"

    **.mn8.networkLayer.hip.DNSFile=xmldoc("dnsMN10.xml")
    **.mn8.networkLayer.hip.hipAddress="50:50:50:50:AA23:F67D:228
8:B340"

    **.mn9.networkLayer.hip.DNSFile=xmldoc("dnsMN10.xml")
    **.mn9.networkLayer.hip.hipAddress="50:50:50:50:DE56:874A:885
5:65E2"

    **.mn10.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn10.networkLayer.hip.hipAddress="50:50:50:50:E27B:3987:20
01:E56F"

    **.mn11.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn11.networkLayer.hip.hipAddress="50:50:50:50:EE12:A34B:87
65:BB12"

    **.mn12.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
```

```
    **.mn12.networkLayer.hip.hipAddress="50:50:50:50:AAE8:345:A02
1:65EA"

    **.mn13.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn13.networkLayer.hip.hipAddress="50:50:50:50:B340:3344:77
66:A028"

    **.mn14.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn14.networkLayer.hip.hipAddress="50:50:50:50:DDDD:65E2:87
4A:8855"

    **.mn15.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn15.networkLayer.hip.hipAddress="50:50:50:50:2001:E56F:E2
7B:3987"

    **.mn16.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn16.networkLayer.hip.hipAddress="50:50:50:50:8765:BB12:EE
12:A34B"

    **.mn17.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn17.networkLayer.hip.hipAddress="50:50:50:50:A021:65EA:AA
E8:345"

    **.mn18.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn18.networkLayer.hip.hipAddress="50:50:50:50:7766:A028:B3
40:3344"

    **.mn19.networkLayer.hip.DNSFile=xmldoc("dnsMN20.xml")
    **.mn19.networkLayer.hip.hipAddress="50:50:50:50:874A:8855:DD
DD:65E2"

    **.mn20.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn20.networkLayer.hip.hipAddress="50:50:50:50:67FF:98A1:9:
2974"

    **.mn21.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn21.networkLayer.hip.hipAddress="50:50:50:50:C02:6543:887
7:208A"

    **.mn22.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
```

```
    **.mn22.networkLayer.hip.hipAddress="50:50:50:50:D0E2:65EC:CC
87:5162"

    **.mn23.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn23.networkLayer.hip.hipAddress="50:50:50:50:87AB:0:56EF:
A083"

    **.mn24.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn24.networkLayer.hip.hipAddress="50:50:50:50:9E00:54F2:BE
50:A7A1"

    **.mn25.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn25.networkLayer.hip.hipAddress="50:50:50:50:52F0:7733:9A
22:7004"

    **.mn26.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn26.networkLayer.hip.hipAddress="50:50:50:50:88EE:76AA:5A
13:607"

    **.mn27.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn27.networkLayer.hip.hipAddress="50:50:50:50:BBBB:6704:AD
87:54EE"

    **.mn28.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn28.networkLayer.hip.hipAddress="50:50:50:50:66BB:D876:34
2A:F60F"

    **.mn29.networkLayer.hip.DNSFile=xmldoc("dnsMN30.xml")
    **.mn29.networkLayer.hip.hipAddress="50:50:50:50:26EE:4301:7A
4B:C5F8"

    **.mn30.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn30.networkLayer.hip.hipAddress="50:50:50:50:BB34:8721:50
02:F339"

    **.mn31.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn31.networkLayer.hip.hipAddress="50:50:50:50:EA65:1111:56
33:EE54"

    **.mn32.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
```

```
    **.mn32.networkLayer.hip.hipAddress="50:50:50:50:7600:EF23:D8
87:2211"

    **.mn33.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn33.networkLayer.hip.hipAddress="50:50:50:50:5544:3322:11
11:AABB"

    **.mn34.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn34.networkLayer.hip.hipAddress="50:50:50:50:6666:4300:E2
3A:7619"

    **.mn35.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn35.networkLayer.hip.hipAddress="50:50:50:50:3001:CC28:88
E5:F12A"

    **.mn36.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn36.networkLayer.hip.hipAddress="50:50:50:50:CB23:9876:99
EA:E200"

    **.mn37.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn37.networkLayer.hip.hipAddress="50:50:50:50:BE76:4302:76
52:AA65"

    **.mn38.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn38.networkLayer.hip.hipAddress="50:50:50:50:1199:C054:B5
5E:3002"

    **.mn39.networkLayer.hip.DNSFile=xmldoc("dnsMN40.xml")
    **.mn39.networkLayer.hip.hipAddress="50:50:50:50:DBA0:E0F4:67
22:1065"

    #**.mn40.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn40.networkLayer.hip.hipAddress="50:50:50:50:5002:F339:B
B34:8721"

    #**.mn41.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn41.networkLayer.hip.hipAddress="50:50:50:50:5633:EE54:E
A65:1111"

    #**.mn42.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
```

```
    #**.mn42.networkLayer.hip.hipAddress="50:50:50:50:D887:2211:7
600:EF23"

    #**.mn43.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn43.networkLayer.hip.hipAddress="50:50:50:50:1111:AABB:5
544:3322"

    #**.mn44.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn44.networkLayer.hip.hipAddress="50:50:50:50:E23A:7619:6
666:4300"

    #**.mn45.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn45.networkLayer.hip.hipAddress="50:50:50:50:88E5:F12A:3
001:CC28"

    #**.mn46.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn46.networkLayer.hip.hipAddress="50:50:50:50:99EA:E200:C
B23:9876"

    #**.mn47.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn47.networkLayer.hip.hipAddress="50:50:50:50:7652:AA65:B
E76:4302"

    #**.mn48.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn48.networkLayer.hip.hipAddress="50:50:50:50:B55E:3002:1
199:C054"

    #**.mn49.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    #**.mn49.networkLayer.hip.hipAddress="50:50:50:50:6722:1065:D
BA0:E0F4"

    **.cn1.networkLayer.hip.DNSFile=xmldoc("dnsMN50.xml")
    **.cn1.networkLayer.hip.hipAddress="50:50:50:50:F67D:AA32:0:4
39A"

    **.srv.networkLayer.hip.DNSFile=xmldoc("dnsSrv.xml")
    **.srv.networkLayer.hip.hipAddress="50:50:50:50:ac24:aff:fe11
:bba"

    **.hipRVS.DNSFile=xmldoc("rvs50.xml")
    **.hipRVS.hipAddress="50:50:50:50:532:ff4a:fe59:1256"
```

```
     # ARP configuration
     **.arp.retryTimeout = 1
     **.arp.retryCount = 3
     **.arp.cacheTimeout = 100
     **.networkLayer.proxyARP = true  # Host's is hardwired
"false"

     # PPP NIC configuration
     #**.ppp[*].queueType = "DropTailQueue" # in routers
     #**.ppp[*].queue.frameCapacity = 10  # in routers

     # Ethernet NIC configuration
     **.eth[*].queueType = "DropTailQueue" # in routers
     **.eth[*].queue.frameCapacity = 10  # in routers
     **.eth[*].encap.writeScalars = false
     **.eth[*].mac.promiscuous = false
     **.eth[*].mac.address = "auto"
     **.eth[*].mac.txrate = 10e6
     **.eth[*].mac.duplexEnabled = true
     **.eth[*].mac.writeScalars = false
     **.hub.writeScalars = false

     # APs configuration
     **.ap1.eth[0].address="3018:FFFF:0:a0:127b:c0ff:a56b:94c5"
     **.ap2.eth[0].address="3012:AAAA:0:12b4:fe25:bb4:c45a:2597"
     **.ap3.eth[0].address="3019:CCCC:0:d0:653e:39aa:bb26:aa34"
     **.ap4.eth[0].address="3013:BBBB:23a:8888:bc34:8954:563a:123a
"
     **.ap*.eth[0].txrate=10e6
     **.ap*.eth[0].duplexEnabled=1
     **.ap*.eth[0].writeScalars=0

     # Wireless channels
     **.ap1.wlan.radio.channelNumber = 1
     **.ap2.wlan.radio.channelNumber = 6
     **.ap3.wlan.radio.channelNumber = 11
     **.ap4.wlan.radio.channelNumber = 1
     **.host0.wlan.radio.channelNumber = 0  # just initially --
it'll scan
     **.mn*.wlan.radio.channelNumber = 0
```

```
      # wireless configuration
      **.wlan.agent.activeScan = 1
      **.wlan.agent.channelsToScan = "1 6 11"  # "" means all
      **.wlan.agent.probeDelay = 0.1
      **.wlan.agent.minChannelTime = 0.05
      **.wlan.agent.maxChannelTime = 0.15
      **.wlan.agent.authenticationTimeout = 5
      **.wlan.agent.associationTimeout = 5

      **.mac.address = "auto"
      **.mac.maxQueueSize = 14
      **.mac.rtsThresholdBytes = 4000
      **.mac.bitrate = 2e6  # 2Mbps
      **.wlan.mac.retryLimit = 7
      **.wlan.mac.cwMinData = 7
      **.wlan.mac.cwMinBroadcast = 31

      **.radio.bitrate=2E+6 ;in bits/second
      **.radio.transmitterPower=1.0  ;[mW]
      **.radio.carrierFrequency=2.4E+9
      **.radio.thermalNoise=-110
      **.radio.sensitivity=-85
      **.radio.pathLossAlpha=2
      **.radio.snirThreshold = 4  # in dB

      # relay unit configuration (APs)
      **.relayUnitType = "MACRelayUnitNP"
      **.relayUnit.addressTableSize = 100
      **.relayUnit.agingTime = 120s
      **.relayUnit.bufferSize = 1048576  # 1Mb
      **.relayUnit.highWatermark = 524288  # 512Kmn
      **.relayUnit.pauseUnits = 300  # pause for 300*512 bit (19200
byte) time
      **.relayUnit.addressTableFile = ""
      **.relayUnit.numCPUs = 2
      **.relayUnit.processingTime = 2us
      **.relayUnit.writeScalars = false

      **.mn*.numUdpApps=0
      **.mn*.udpAppType="UDPVideoStreamCli"
```

```
**.mn0.numUdpApps=1
**.mn0.udpAppType="UDPVideoStreamCli"
**.mn0.udpApp[0].file="mn0_udpApp.csv"
**.mn0.udpApp[0].localPort=1024
**.mn0.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn0.udpApp[0].serverPort=1024
**.mn0.udpApp[0].startTime=uniform(10,12)

**.mn1.numUdpApps=1
**.mn1.udpAppType="UDPVideoStreamCli"
**.mn1.udpApp[0].file="mn1_udpApp.csv"
**.mn1.udpApp[0].localPort=1025
**.mn1.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn1.udpApp[0].serverPort=1024
**.mn1.udpApp[0].startTime=uniform(10,12)

**.mn2.numUdpApps=1
**.mn2.udpAppType="UDPVideoStreamCli"
**.mn2.udpApp[0].file="mn2_udpApp.csv"
**.mn2.udpApp[0].localPort=1026
**.mn2.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn2.udpApp[0].serverPort=1024
**.mn2.udpApp[0].startTime=uniform(10,12)

**.mn3.numUdpApps=1
**.mn3.udpAppType="UDPVideoStreamCli"
**.mn3.udpApp[0].file="mn3_udpApp.csv"
**.mn3.udpApp[0].localPort=1027
**.mn3.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn3.udpApp[0].serverPort=1024
**.mn3.udpApp[0].startTime=uniform(10,12)

**.cn1.numUdpApps=1
**.cn1.udpAppType="UDPVideoStreamSvr"
**.cn1.udpApp[0].serverPort=1024
**.cn1.udpApp[0].waitInterval=0.2
**.cn1.udpApp[0].packetLen=1000
**.cn1.udpApp[0].videoSize=uniform(10000,20000)

**.mn4.numUdpApps=1
```

```
**.mn4.udpAppType="UDPVideoStreamCli"
**.mn4.udpApp[0].file="mn4_udpApp.csv"
**.mn4.udpApp[0].localPort=1028
**.mn4.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn4.udpApp[0].serverPort=1024
**.mn4.udpApp[0].startTime=uniform(10,12)

**.mn5.numUdpApps=1
**.mn5.udpAppType="UDPVideoStreamCli"
**.mn5.udpApp[0].file="mn5_udpApp.csv"
**.mn5.udpApp[0].localPort=1028
**.mn5.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn5.udpApp[0].serverPort=1024
**.mn5.udpApp[0].startTime=uniform(10,12)

**.mn6.numUdpApps=1
**.mn6.udpAppType="UDPVideoStreamCli"
**.mn6.udpApp[0].file="mn6_udpApp.csv"
**.mn6.udpApp[0].localPort=1028
**.mn6.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A"
**.mn6.udpApp[0].serverPort=1024
**.mn6.udpApp[0].startTime=uniform(10,12)

**.mn15.numUdpApps=1
**.mn15.udpAppType="UDPVideoStreamCli"
**.mn15.udpApp[0].file="mn15_udpApp.csv"
**.mn15.udpApp[0].localPort=1029
**.mn15.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
**.mn15.udpApp[0].serverPort=1024
**.mn15.udpApp[0].startTime=uniform(10,12)

**.mn16.numUdpApps=1
**.mn16.udpAppType="UDPVideoStreamCli"
**.mn16.udpApp[0].file="mn16_udpApp.csv"
**.mn16.udpApp[0].localPort=1030
**.mn16.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
**.mn16.udpApp[0].serverPort=1024
**.mn16.udpApp[0].startTime=uniform(10,12)
```

```
        **.mn17.numUdpApps=1
        **.mn17.udpAppType="UDPVideoStreamCli"
        **.mn17.udpApp[0].file="mn6_udpApp.csv"
        **.mn17.udpApp[0].localPort=1031
        **.mn17.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
    "
        **.mn17.udpApp[0].serverPort=1024
        **.mn17.udpApp[0].startTime=uniform(10,12)

        **.mn21.numUdpApps=1
        **.mn21.udpAppType="UDPVideoStreamCli"
        **.mn21.udpApp[0].file="mn21_udpApp.csv"
        **.mn21.udpApp[0].localPort=1032
        **.mn21.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
    "
        **.mn21.udpApp[0].serverPort=1024
        **.mn21.udpApp[0].startTime=uniform(10,12)

        **.mn22.numUdpApps=1
        **.mn22.udpAppType="UDPVideoStreamCli"
        **.mn22.udpApp[0].file="mn22_udpApp.csv"
        **.mn22.udpApp[0].localPort=1033
        **.mn22.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
    "
        **.mn22.udpApp[0].serverPort=1024
        **.mn22.udpApp[0].startTime=uniform(10,12)

        **.mn23.numUdpApps=1
        **.mn23.udpAppType="UDPVideoStreamCli"
        **.mn23.udpApp[0].file="mn23_udpApp.csv"
        **.mn23.udpApp[0].localPort=1034
        **.mn23.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
    "
        **.mn23.udpApp[0].serverPort=1024
        **.mn23.udpApp[0].startTime=uniform(10,12)

        **.mn25.numUdpApps=1
        **.mn25.udpAppType="UDPVideoStreamCli"
        **.mn25.udpApp[0].file="mn25_udpApp.csv"
        **.mn25.udpApp[0].localPort=1035
```

```
        **.mn25.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
        **.mn25.udpApp[0].serverPort=1024
        **.mn25.udpApp[0].startTime=uniform(10,12)

        **.mn26.numUdpApps=1
        **.mn26.udpAppType="UDPVideoStreamCli"
        **.mn26.udpApp[0].file="mn26_udpApp.csv"
        **.mn26.udpApp[0].localPort=1036
        **.mn26.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
        **.mn26.udpApp[0].serverPort=1024
        **.mn26.udpApp[0].startTime=uniform(10,12)

        **.mn32.numUdpApps=1
        **.mn32.udpAppType="UDPVideoStreamCli"
        **.mn32.udpApp[0].file="mn32_udpApp.csv"
        **.mn32.udpApp[0].localPort=1037
        **.mn32.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
        **.mn32.udpApp[0].serverPort=1024
        **.mn32.udpApp[0].startTime=uniform(10,12)

        **.mn31.numUdpApps=1
        **.mn31.udpAppType="UDPVideoStreamCli"
        **.mn31.udpApp[0].file="mn31_udpApp.csv"
        **.mn31.udpApp[0].localPort=1038
        **.mn31.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
        **.mn31.udpApp[0].serverPort=1024
        **.mn31.udpApp[0].startTime=uniform(10,12)

        **.mn30.numUdpApps=1
        **.mn30.udpAppType="UDPVideoStreamCli"
        **.mn30.udpApp[0].file="mn30_udpApp.csv"
        **.mn30.udpApp[0].localPort=1039
        **.mn30.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
        **.mn30.udpApp[0].serverPort=1024
        **.mn30.udpApp[0].startTime=uniform(10,12)
```

```
**.mn35.numUdpApps=1
**.mn35.udpAppType="UDPVideoStreamCli"
**.mn35.udpApp[0].file="mn35_udpApp.csv"
**.mn35.udpApp[0].localPort=1040
**.mn35.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
**.mn35.udpApp[0].serverPort=1024
**.mn35.udpApp[0].startTime=uniform(10,12)

**.mn36.numUdpApps=1
**.mn36.udpAppType="UDPVideoStreamCli"
**.mn36.udpApp[0].file="mn36_udpApp.csv"
**.mn36.udpApp[0].localPort=1041
**.mn36.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
**.mn36.udpApp[0].serverPort=1024
**.mn36.udpApp[0].startTime=uniform(10,12)

**.mn37.numUdpApps=1
**.mn37.udpAppType="UDPVideoStreamCli"
**.mn37.udpApp[0].file="mn37_udpApp.csv"
**.mn37.udpApp[0].localPort=1042
**.mn37.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439A
"
**.mn37.udpApp[0].serverPort=1024
**.mn37.udpApp[0].startTime=uniform(10,12)

#**.mn45.numUdpApps=1
#**.mn45.udpAppType="UDPVideoStreamCli"
#**.mn45.udpApp[0].file="mn45_udpApp.csv"
#**.mn45.udpApp[0].localPort=1043
#**.mn45.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439
A"
#**.mn45.udpApp[0].serverPort=1024
#**.mn45.udpApp[0].startTime=uniform(10,12)

#**.mn46.numUdpApps=1
#**.mn46.udpAppType="UDPVideoStreamCli"
#**.mn46.udpApp[0].file="mn46_udpApp.csv"
#**.mn46.udpApp[0].localPort=1043
```

```
     #**.mn46.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439
A"
     #**.mn46.udpApp[0].serverPort=1024
     #**.mn46.udpApp[0].startTime=uniform(10,12)

     #**.mn47.numUdpApps=1
     #**.mn47.udpAppType="UDPVideoStreamCli"
     #**.mn47.udpApp[0].file="mn47_udpApp.csv"
     #**.mn47.udpApp[0].localPort=1042
     #**.mn47.udpApp[0].serverAddress="50:50:50:50:F67D:AA32:0:439
A"
     #**.mn47.udpApp[0].serverPort=1024
     #**.mn47.udpApp[0].startTime=uniform(10,12)

     **.srv.numUdpApps = 1
     **.srv.udpAppType="UDPAppOnOff"
     **.srv.udpApp[0].startTime=uniform(10,12)
     **.srv.udpApp[0].packetInterval=0.05
     **.srv.udpApp[0].numPackets=uniform(10000,20000)
     **.srv.udpApp[0].packetLength=4400
     **.srv.udpApp[0].average_distrib_on=exponential(1.004) #1.004
     **.srv.udpApp[0].average_distrib_off=exponential(1.587)
#1.587
     **.srv.udpApp[0].local_port=1024
     **.srv.udpApp[0].dest_port=1024
     **.srv.udpApp[0].dest_addresses="50:50:50:50:2580:ebff:fe8c:1
448"

     **.host0.numUdpApps = 1
     **.host0.udpAppType = "UDPSink"
     **.host0.udpApp[0].local_port = 1024

     **.mn7.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439A
"
     **.mn7.pingApp.srcAddr=""
     **.mn7.pingApp.packetSize=56
     **.mn7.pingApp.interval=1
     **.mn7.pingApp.hopLimit=32
     **.mn7.pingApp.count=0
     **.mn7.pingApp.startTime=1
     **.mn7.pingApp.stopTime=300
```

```
**.mn7.pingApp.printPing=false

**.mn8.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439A
"
**.mn8.pingApp.srcAddr=""
**.mn8.pingApp.packetSize=56
**.mn8.pingApp.interval=1
**.mn8.pingApp.hopLimit=32
**.mn8.pingApp.count=0
**.mn8.pingApp.startTime=1
**.mn8.pingApp.stopTime=300
**.mn8.pingApp.printPing=false

**.mn9.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439A
"
**.mn9.pingApp.srcAddr=""
**.mn9.pingApp.packetSize=56
**.mn9.pingApp.interval=1
**.mn9.pingApp.hopLimit=32
**.mn9.pingApp.count=0
**.mn9.pingApp.startTime=1
**.mn9.pingApp.stopTime=300
**.mn9.pingApp.printPing=false

**.mn10.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn10.pingApp.srcAddr=""
**.mn10.pingApp.packetSize=56
**.mn10.pingApp.interval=1
**.mn10.pingApp.hopLimit=32
**.mn10.pingApp.count=0
**.mn10.pingApp.startTime=1
**.mn10.pingApp.stopTime=300
**.mn10.pingApp.printPing=false

**.mn11.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn11.pingApp.srcAddr=""
**.mn11.pingApp.packetSize=56
**.mn11.pingApp.interval=1
**.mn11.pingApp.hopLimit=32
```

```
**.mn11.pingApp.count=0
**.mn11.pingApp.startTime=1
**.mn11.pingApp.stopTime=300
**.mn11.pingApp.printPing=false

**.mn12.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn12.pingApp.srcAddr=""
**.mn12.pingApp.packetSize=56
**.mn12.pingApp.interval=1
**.mn12.pingApp.hopLimit=32
**.mn12.pingApp.count=0
**.mn12.pingApp.startTime=1
**.mn12.pingApp.stopTime=300
**.mn12.pingApp.printPing=false

**.mn13.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn13.pingApp.srcAddr=""
**.mn13.pingApp.packetSize=56
**.mn13.pingApp.interval=1
**.mn13.pingApp.hopLimit=32
**.mn13.pingApp.count=0
**.mn13.pingApp.startTime=1
**.mn13.pingApp.stopTime=300
**.mn13.pingApp.printPing=false

**.mn14.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn14.pingApp.srcAddr=""
**.mn14.pingApp.packetSize=56
**.mn14.pingApp.interval=1
**.mn14.pingApp.hopLimit=32
**.mn14.pingApp.count=0
**.mn14.pingApp.startTime=1
**.mn14.pingApp.stopTime=300
**.mn14.pingApp.printPing=false

**.mn18.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn18.pingApp.srcAddr=""
```

```
**.mn18.pingApp.packetSize=56
**.mn18.pingApp.interval=1
**.mn18.pingApp.hopLimit=32
**.mn18.pingApp.count=0
**.mn18.pingApp.startTime=1
**.mn18.pingApp.stopTime=300
**.mn18.pingApp.printPing=false

**.mn19.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn19.pingApp.srcAddr=""
**.mn19.pingApp.packetSize=56
**.mn19.pingApp.interval=1
**.mn19.pingApp.hopLimit=32
**.mn19.pingApp.count=0
**.mn19.pingApp.startTime=1
**.mn19.pingApp.stopTime=300
**.mn19.pingApp.printPing=false

**.mn20.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn20.pingApp.srcAddr=""
**.mn20.pingApp.packetSize=56
**.mn20.pingApp.interval=1
**.mn20.pingApp.hopLimit=32
**.mn20.pingApp.count=0
**.mn20.pingApp.startTime=1
**.mn20.pingApp.stopTime=300
**.mn20.pingApp.printPing=false

**.mn24.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn24.pingApp.srcAddr=""
**.mn24.pingApp.packetSize=56
**.mn24.pingApp.interval=1
**.mn24.pingApp.hopLimit=32
**.mn24.pingApp.count=0
**.mn24.pingApp.startTime=1
**.mn24.pingApp.stopTime=300
**.mn24.pingApp.printPing=false
```

```
     **.mn27.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
     **.mn27.pingApp.srcAddr=""
     **.mn27.pingApp.packetSize=56
     **.mn27.pingApp.interval=1
     **.mn27.pingApp.hopLimit=32
     **.mn27.pingApp.count=0
     **.mn27.pingApp.startTime=1
     **.mn27.pingApp.stopTime=300
     **.mn27.pingApp.printPing=false

     **.mn28.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
     **.mn28.pingApp.srcAddr=""
     **.mn28.pingApp.packetSize=56
     **.mn28.pingApp.interval=1
     **.mn28.pingApp.hopLimit=32
     **.mn28.pingApp.count=0
     **.mn28.pingApp.startTime=1
     **.mn28.pingApp.stopTime=300
     **.mn28.pingApp.printPing=false

     **.mn29.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
     **.mn29.pingApp.srcAddr=""
     **.mn29.pingApp.packetSize=56
     **.mn29.pingApp.interval=1
     **.mn29.pingApp.hopLimit=32
     **.mn29.pingApp.count=0
     **.mn29.pingApp.startTime=1
     **.mn29.pingApp.stopTime=300
     **.mn29.pingApp.printPing=false

     **.mn33.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
     **.mn33.pingApp.srcAddr=""
     **.mn33.pingApp.packetSize=56
     **.mn33.pingApp.interval=1
     **.mn33.pingApp.hopLimit=32
     **.mn33.pingApp.count=0
     **.mn33.pingApp.startTime=1
```

```
**.mn33.pingApp.stopTime=300
**.mn33.pingApp.printPing=false

**.mn34.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn34.pingApp.srcAddr=""
**.mn34.pingApp.packetSize=56
**.mn34.pingApp.interval=1
**.mn34.pingApp.hopLimit=32
**.mn34.pingApp.count=0
**.mn34.pingApp.startTime=1
**.mn34.pingApp.stopTime=300
**.mn34.pingApp.printPing=false

**.mn38.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn38.pingApp.srcAddr=""
**.mn38.pingApp.packetSize=56
**.mn38.pingApp.interval=1
**.mn38.pingApp.hopLimit=32
**.mn38.pingApp.count=0
**.mn38.pingApp.startTime=1
**.mn38.pingApp.stopTime=300
**.mn38.pingApp.printPing=false

**.mn39.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:439
A"
**.mn39.pingApp.srcAddr=""
**.mn39.pingApp.packetSize=56
**.mn39.pingApp.interval=1
**.mn39.pingApp.hopLimit=32
**.mn39.pingApp.count=0
**.mn39.pingApp.startTime=1
**.mn39.pingApp.stopTime=300
**.mn39.pingApp.printPing=false

#**.mn40.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
#**.mn40.pingApp.srcAddr=""
#**.mn40.pingApp.packetSize=56
#**.mn40.pingApp.interval=1
```

```
    #**.mn40.pingApp.hopLimit=32
    #**.mn40.pingApp.count=0
    #**.mn40.pingApp.startTime=1
    #**.mn40.pingApp.stopTime=300
    #**.mn40.pingApp.printPing=false

    #**.mn41.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
    #**.mn41.pingApp.srcAddr=""
    #**.mn41.pingApp.packetSize=56
    #**.mn41.pingApp.interval=1
    #**.mn41.pingApp.hopLimit=32
    #**.mn41.pingApp.count=0
    #**.mn41.pingApp.startTime=1
    #**.mn41.pingApp.stopTime=300
    #**.mn41.pingApp.printPing=false

    #**.mn42.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
    #**.mn42.pingApp.srcAddr=""
    #**.mn42.pingApp.packetSize=56
    #**.mn42.pingApp.interval=1
    #**.mn42.pingApp.hopLimit=32
    #**.mn42.pingApp.count=0
    #**.mn42.pingApp.startTime=1
    #**.mn42.pingApp.stopTime=300
    #**.mn42.pingApp.printPing=false

    #**.mn43.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
    #**.mn43.pingApp.srcAddr=""
    #**.mn43.pingApp.packetSize=56
    #**.mn43.pingApp.interval=1
    #**.mn43.pingApp.hopLimit=32
    #**.mn43.pingApp.count=0
    #**.mn43.pingApp.startTime=1
    #**.mn43.pingApp.stopTime=300
    #**.mn43.pingApp.printPing=false

    #**.mn44.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
```

```
    #**.mn44.pingApp.srcAddr=""
    #**.mn44.pingApp.packetSize=56
    #**.mn44.pingApp.interval=1
    #**.mn44.pingApp.hopLimit=32
    #**.mn44.pingApp.count=0
    #**.mn44.pingApp.startTime=1
    #**.mn44.pingApp.stopTime=300
    #**.mn44.pingApp.printPing=false

    #**.mn48.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
    #**.mn48.pingApp.srcAddr=""
    #**.mn48.pingApp.packetSize=56
    #**.mn48.pingApp.interval=1
    #**.mn48.pingApp.hopLimit=32
    #**.mn48.pingApp.count=0
    #**.mn48.pingApp.startTime=1
    #**.mn48.pingApp.stopTime=300
    #**.mn48.pingApp.printPing=false

    #**.mn49.pingApp.destAddr="3011:AAAA:33BB:2211:F67D:AA32:0:43
9A"
    #**.mn49.pingApp.srcAddr=""
    #**.mn49.pingApp.packetSize=56
    #**.mn49.pingApp.interval=1
    #**.mn49.pingApp.hopLimit=32
    #**.mn49.pingApp.count=0
    #**.mn49.pingApp.startTime=1
    #**.mn49.pingApp.stopTime=300
    #**.mn49.pingApp.printPing=false

    [Run 1]
    output-vector-file = Tese1.vec
    output-scalar-file = Tese1.sca
    **.host0.udpApp[0].file="clientOnOff_1.csv"
    **.srv.udpApp[0].file="srv_1.csv"
    **.cn1.udpApp[0].file="cn1_1.csv"
```

8.25.
Arquivo WirelessIPv6HIPTese.ned

```
    import
        "WirelessHIPHost6",
        "NetworkHIPLayer",
        "WirelessAP",
        "WirelessAP2",
        "WirelessAPWithEth",
        "Router6",
        "RVS_HIP",
        "StandardHostHIP6";

    channel ethernetCable
        delay 4.0e-3;
    endchannel

    channel InternetCable
        delay 20.0e-3;
    endchannel

    module HandoverTeseNetwork
        parameters:
            playgroundSizeX: numeric const,
            playgroundSizeY: numeric const;
        submodules:
            host0: WirelessHIPHost6;
                display:
"p=58,346;i=device/wifilaptop;r=,,#707070";
            mn0: WirelessHIPHost6;
                display:
"p=34,412;i=device/wifilaptop;r=,,#707070";
            mn1: WirelessHIPHost6;
                display:
"p=332,400;i=device/wifilaptop;r=,,#707070";
            mn2: WirelessHIPHost6;
                display:
"p=596,354;i=device/wifilaptop;r=,,#707070";
            mn3: WirelessHIPHost6;
                display:
"p=762,392;i=device/wifilaptop;r=,,#707070";
```

```
            mn4: WirelessHIPHost6;
                display:
"p=106,348;i=device/wifilaptop;r=,,#707070";
            mn5: WirelessHIPHost6;
                display:
"p=454,388;i=device/wifilaptop;r=,,#707070";
            mn6: WirelessHIPHost6;
                display:
"p=612,384;i=device/wifilaptop;r=,,#707070";
            mn7: WirelessHIPHost6;
                display:
"p=808,410;i=device/wifilaptop;r=,,#707070";
            mn8: WirelessHIPHost6;
                display:
"p=246,352;i=device/wifilaptop;r=,,#707070";
            mn9: WirelessHIPHost6;
                display:
"p=418,348;i=device/wifilaptop;r=,,#707070";
            mn10: WirelessHIPHost6;
                display:
"p=366,336;i=device/wifilaptop;r=,,#707070";
            mn11: WirelessHIPHost6;
                display:
"p=566,392;i=device/wifilaptop;r=,,#707070";
            mn12: WirelessHIPHost6;
                display:
"p=678,394;i=device/wifilaptop;r=,,#707070";
            mn13: WirelessHIPHost6;
                display:
"p=200,388;i=device/wifilaptop;r=,,#707070";
            mn14: WirelessHIPHost6;
                display:
"p=308,364;i=device/wifilaptop;r=,,#707070";
            mn15: WirelessHIPHost6;
                display:
"p=834,380;i=device/wifilaptop;r=,,#707070";
            mn16: WirelessHIPHost6;
                display:
"p=596,304;i=device/wifilaptop;r=,,#707070";
            mn17: WirelessHIPHost6;
```

```
                display:
"p=338,314;i=device/wifilaptop;r=,,#707070";
            mn18: WirelessHIPHost6;
                display:
"p=94,392;i=device/wifilaptop;r=,,#707070";
            mn19: WirelessHIPHost6;
                display:
"p=418,298;i=device/wifilaptop;r=,,#707070";
            mn20: WirelessHIPHost6;
                display:
"p=768,340;i=device/wifilaptop;r=,,#707070";
            mn21: WirelessHIPHost6;
                display:
"p=522,364;i=device/wifilaptop;r=,,#707070";
            mn22: WirelessHIPHost6;
                display:
"p=666,350;i=device/wifilaptop;r=,,#707070";
            mn23: WirelessHIPHost6;
                display:
"p=164,368;i=device/wifilaptop;r=,,#707070";
            mn24: WirelessHIPHost6;
                display:
"p=288,344;i=device/wifilaptop;r=,,#707070";
            mn25: WirelessHIPHost6;
                display:
"p=182,304;i=device/wifilaptop;r=,,#707070";
            mn26: WirelessHIPHost6;
                display:
"p=720,380;i=device/wifilaptop;r=,,#707070";
            mn27: WirelessHIPHost6;
                display:
"p=270,406;i=device/wifilaptop;r=,,#707070";
            mn28: WirelessHIPHost6;
                display:
"p=234,308;i=device/wifilaptop;r=,,#707070";
            mn29: WirelessHIPHost6;
                display:
"p=518,398;i=device/wifilaptop;r=,,#707070";
            mn30: WirelessHIPHost6;
                display:
"p=514,316;i=device/wifilaptop;r=,,#707070";
```

```
            mn31: WirelessHIPHost6;
                display:
"p=474,300;i=device/wifilaptop;r=,,#707070";
            mn32: WirelessHIPHost6;
                display:
"p=578,278;i=device/wifilaptop;r=,,#707070";
            mn33: WirelessHIPHost6;
                display:
"p=92,288;i=device/wifilaptop;r=,,#707070";
            mn34: WirelessHIPHost6;
                display:
"p=208,344;i=device/wifilaptop;r=,,#707070";
            mn35: WirelessHIPHost6;
                display:
"p=132,254;i=device/wifilaptop;r=,,#707070";
            mn36: WirelessHIPHost6;
                display:
"p=632,346;i=device/wifilaptop;r=,,#707070";
            mn37: WirelessHIPHost6;
                display:
"p=494,350;i=device/wifilaptop;r=,,#707070";
            mn38: WirelessHIPHost6;
                display:
"p=184,258;i=device/wifilaptop;r=,,#707070";
            mn39: WirelessHIPHost6;
                display:
"p=700,332;i=device/wifilaptop;r=,,#707070";
            rvs: RVS_HIP;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display: "p=130,50;i=device/router;r=,,#707070";
            map: Router6;
                gatesizes:
                    ethIn[5],
                    ethOut[5];
                display: "p=324,138;i=device/router;r=,,#707070";
            ir1: Router6;
                gatesizes:
                    ethIn[3],
                    ethOut[3];
```

```
                display: "p=168,226;i=device/router;r=,,#707070";
            ir2: Router6;
                gatesizes:
                    ethIn[3],
                    ethOut[3];
                display: "p=528,218;i=device/router;r=,,#707070";
            srv: StandardHostHIP6;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=578,50;i=device/server_l;r=,,#707070";
            cn1: StandardHostHIP6;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=288,50;i=device/server_l;r=,,#707070";
            ap1: WirelessAPWithEth;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=50,364;i=device/accesspoint;r=,,#707070";
            ar1: Router6;
                gatesizes:
                    ethIn[2],
                    ethOut[2];
                display: "p=50,296;i=device/router;r=,,#707070";
            ap2: WirelessAPWithEth;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=300,364;i=device/accesspoint;r=,,#707070";
            ar2: Router6;
                gatesizes:
                    ethIn[2],
                    ethOut[2];
                display: "p=300,296;i=device/router;r=,,#707070";
            ap3: WirelessAPWithEth;
```

```
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=550,356;i=device/accesspoint;r=,,#707070";
            ar3: Router6;
                gatesizes:
                    ethIn[2],
                    ethOut[2];
                display: "p=550,296;i=device/router;r=,,#707070";
            ap4: WirelessAPWithEth;
                gatesizes:
                    ethIn[1],
                    ethOut[1];
                display:
"p=800,364;i=device/accesspoint;r=,,#707070";
            ar4: Router6;
                gatesizes:
                    ethIn[2],
                    ethOut[2];
                display: "p=800,296;i=device/router;r=,,#707070";
            channelcontrol: ChannelControl;
                parameters:
                    playgroundSizeX = playgroundSizeX,
                    playgroundSizeY = playgroundSizeY;
                display: "p=60,50;i=misc/sun";
        connections nocheck:
            srv.ethIn[0] <-- InternetCable <-- map.ethOut[0];
            srv.ethOut[0] --> InternetCable --> map.ethIn[0];

            map.ethIn[1] <-- InternetCable <-- rvs.ethOut[0];
            map.ethOut[1] --> InternetCable --> rvs.ethIn[0];

            map.ethIn[2] <-- ethernetCable <-- ir1.ethOut[0];
            map.ethOut[2] --> ethernetCable --> ir1.ethIn[0];

            map.ethIn[3] <-- ethernetCable <-- ir2.ethOut[0];
            map.ethOut[3] --> ethernetCable --> ir2.ethIn[0];

            map.ethIn[4] <-- InternetCable <-- cn1.ethOut[0];
            map.ethOut[4] --> InternetCable --> cn1.ethIn[0];
```

```
        ap2.ethIn[0] <-- ethernetCable <-- ar2.ethOut[0];
        ap2.ethOut[0] --> ethernetCable --> ar2.ethIn[0];

        ir1.ethIn[2] <-- ethernetCable <-- ar2.ethOut[1];
        ir1.ethOut[2] --> ethernetCable --> ar2.ethIn[1];

        ap1.ethIn[0] <-- ethernetCable <-- ar1.ethOut[0];
        ap1.ethOut[0] --> ethernetCable --> ar1.ethIn[0];

        ar1.ethIn[1] <-- ethernetCable <-- ir1.ethOut[1];
        ar1.ethOut[1] --> ethernetCable --> ir1.ethIn[1];

        ap3.ethOut[0] --> ethernetCable --> ar3.ethIn[0];
        ap3.ethIn[0] <-- ethernetCable <-- ar3.ethOut[0];

        ar3.ethIn[1] <-- ethernetCable <-- ir2.ethOut[1];
        ar3.ethOut[1] --> ethernetCable --> ir2.ethIn[1];

        ap4.ethOut[0] --> ethernetCable --> ar4.ethIn[0];
        ap4.ethIn[0] <-- ethernetCable <-- ar4.ethOut[0];

        ar4.ethIn[1] <-- ethernetCable <-- ir2.ethOut[2];
        ar4.ethOut[1] --> ethernetCable --> ir2.ethIn[2];

    display: "b=883,437";
endmodule

network handoverTeseNetwork : HandoverTeseNetwork
endnetwork
```

8.26. Arquivo testeTese.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE netconf SYSTEM "../../../Etc/netconf2.dtd">

<netconf>

  <local node="host0" routePackets="off">
```

```
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn0" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn1" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn2" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn3" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn4" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn5" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>
```

```
      <local node="mn6" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn7" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn8" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn9" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn10" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn11" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn12" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>
```

```
       <local node="mn13" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn14" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn15" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn16" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn17" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn18" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn19" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
```

```
      </local>

      <local node="mn20" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn21" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn22" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn23" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn24" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn25" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn26" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
```

```
        </interface>
      </local>

       <local node="mn27" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn28" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn29" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn30" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn31" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn32" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn33" routePackets="off">
```

```
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn34" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn35" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn36" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn37" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

       <local node="mn38" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>

      <local node="mn39" routePackets="off">
        <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
        </interface>
      </local>
```

```
       <local node="mn40" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

       <local node="mn41" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

       <local node="mn42" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

       <local node="mn43" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn44" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

       <local node="mn45" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn46" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>
```

```
        <local node="mn47" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn48" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

        <local node="mn49" routePackets="off">
         <interface name="wlan" AdvSendAdvertisements="off"
HostDupAddrDetectTransmits="1">
         </interface>
       </local>

       <local node="srv" routePackets="off">
         <interface name="eth0" AdvSendAdvertisements="off">

<inetAddr>3011:BBBB:3333:6666:ac24:aff:fe11:bba</inetAddr>
         </interface>
         <route>
           <routeEntry routeIface="eth0"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3011:BBBB:3333:6666:34aa:cb19:0:ee28"/>
         </route>
       </local>

       <local node="rvs" routePackets="off">
         <interface name="eth0" AdvSendAdvertisements="off">
           <inetAddr>3018:FFFF:0:c0:532:ff4a:fe59:1256</inetAddr>
         </interface>
         <route>
           <routeEntry routeIface="eth0"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3018:FFFF:0:c0:77a4:ee32:45bb:6754"/>
         </route>
       </local>
```

```
       <local node="cn1" routePackets="off">
         <interface name="eth0" AdvSendAdvertisements="off">

<inetAddr>3011:AAAA:33BB:2211:F67D:AA32:0:439A</inetAddr>
         </interface>
         <route>
           <routeEntry routeIface="eth0"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3011:AAAA:33BB:2211:F67D:23AE:FF21:9672"/>
         </route>
       </local>

       <!--<local node="cn2" routePackets="off">
         <interface name="eth0" AdvSendAdvertisements="off">
           <inetAddr>3011:CA23:2222:EF45:4567:0:0:AAE8</inetAddr>
         </interface>
       </local> -->

      <local node="map" routePackets="on">
         <interface name="eth0" AdvSendAdvertisements="on"> <!--
srv -->

<inetAddr>3011:BBBB:3333:6666:34aa:cb19:0:ee28</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3011:BBBB:3333:6666:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
         </interface>
         <interface name="eth1" AdvSendAdvertisements="on"> <!--
rvs -->
           <inetAddr>3018:FFFF:0:c0:77a4:ee32:45bb:6754</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:0:c0:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
         </interface>
         <interface name="eth2" AdvSendAdvertisements="on"> <!--
ir1 -->

<inetAddr>3018:FFFF:1:459a:2e39:fff9:54ae:9863</inetAddr>
```

```
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:1:459a:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <interface name="eth3" AdvSendAdvertisements="on"> <!--
ir2 -->

<inetAddr>3018:FFFF:2:542b:54ae:9ea4:dd32:ff23</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:2:542b:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <interface name="eth4" AdvSendAdvertisements="on"> <!--
cn1 -->

<inetAddr>3011:AAAA:33BB:2211:F67D:23AE:FF21:9672</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3011:AAAA:33BB:2211:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
         <!-- cn2 --> <!-- <interface name="eth5"
AdvSendAdvertisements="on">
          <inetAddr>3011:CA23:2222:EF45:0:0:32FE:27A8</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3011:CA23:2222:EF45:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface> -->
        <route>
          <routeEntry routeIface="eth2"
routeDestination="3018:FFFF:0:F0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:3256"/>
          <routeEntry routeIface="eth2"
routeDestination="3018:FFAA:0:E1:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:3256"/>
          <routeEntry routeIface="eth2"
routeDestination="3018:FFFF:0:A0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:3256"/>
```

```
          <routeEntry routeIface="eth2"
routeDestination="3012:AAAA:0:12B4:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:3256"/>
          <routeEntry routeIface="eth3"
routeDestination="3013:AAAA:2AFF:5555:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:65a4"/>
          <routeEntry routeIface="eth3"
routeDestination="3013:BBBB:23A:8888:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:65a4"/>
          <routeEntry routeIface="eth3"
routeDestination="3019:CCCC:0:127B:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:65a4"/>
          <routeEntry routeIface="eth3"
routeDestination="3019:CCCC:0:D0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:65a4"/>
        </route>
       </local>

      <local node="ir1" routePackets="on">
         <interface name="eth0" AdvSendAdvertisements="on"> <!--
map -->

<inetAddr>3018:FFFF:1:459a:2e39:fff9:54ae:3256</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:1:459a:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
         </interface>
         <interface name="eth1" AdvSendAdvertisements="on"> <!--
ar1 -->
          <inetAddr>3018:FFFF:0:f0:127b:c0ff:a546:231a</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:0:f0:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
         </interface>
         <interface name="eth2" AdvSendAdvertisements="on"> <!--
ar2 -->
          <inetAddr>3018:FFAA:0:e1:dd27:ef2a:43fe:159a</inetAddr>
          <AdvPrefixList>
```

```
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFAA:0:e1:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <route>
          <routeEntry routeIface="eth0"
routeDestination="3011:BBBB:3333:6666:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:0:C0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:2:542B:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3013:AAAA:2AFF:5555:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3013:BBBB:23A:8888:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3019:CCCC:0:127B:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3019:CCCC:0:D0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <routeEntry routeIface="eth0"
routeDestination="3011:AAAA:33BB:2211:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/>
          <!-- <routeEntry routeIface="eth0"
routeDestination="3011:CA23:2222:EF45:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:1:459a:2e39:fff9:54ae:9863"/> -->
          <routeEntry routeIface="eth1"
routeDestination="3018:FFFF:0:A0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:0:f0:127b:c0ff:fe2e:7212"/>
          <routeEntry routeIface="eth2"
routeDestination="3012:AAAA:0:12B4:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFAA:0:e1:dd27:3421:f0e3:2211"/>
        </route>
       </local>
```

```
      <local node="ar1" routePackets="on">
         <interface name="eth0" AdvSendAdvertisements="on">  <!--
ap1 -->
           <inetAddr>3018:FFFF:0:a0:127b:c0ff:2364:ee23</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:0:a0:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
         </interface>
         <interface name="eth1" AdvSendAdvertisements="on"> <!--
ir1 -->
           <inetAddr>3018:FFFF:0:f0:127b:c0ff:fe2e:7212</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:0:f0:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
         </interface>
         <route>
           <routeEntry routeIface="eth1"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3018:FFFF:0:f0:127b:c0ff:a546:231a"/>
         </route>
       </local>

      <local node="ar2" routePackets="on">
          <interface name="eth0" AdvSendAdvertisements="on"> <!--
ap2 -->

<inetAddr>3012:AAAA:0:12b4:fe25:bb4:33aa:d235</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3012:AAAA:0:12b4:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
         </interface>
         <interface name="eth1" AdvSendAdvertisements="on"> <!--
ir1 -->
           <inetAddr>3018:FFAA:0:e1:dd27:3421:f0e3:2211</inetAddr>
           <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFAA:0:e1:0:0:0:0/64</AdvPrefix>
           </AdvPrefixList>
```

```
        </interface>
        <route>
          <routeEntry routeIface="eth1"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3018:FFAA:0:e1:dd27:ef2a:43fe:159a"/>
        </route>
       </local>

      <local node="ir2" routePackets="on">
         <interface name="eth0" AdvSendAdvertisements="on"> <!--
map -->

<inetAddr>3018:FFFF:2:542b:54ae:9ea4:dd32:65a4</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:2:542b:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <interface name="eth1" AdvSendAdvertisements="on"> <!--
ar3 -->
          <inetAddr>3019:cccc:0:127b:fe2e:c0ff:0:34ef</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3019:cccc:0:127b:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <interface name="eth2" AdvSendAdvertisements="on"> <!--
ar4 -->

<inetAddr>3013:AAAA:2aff:5555:563a:123a:bc34:3521</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3013:AAAA:2aff:555:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <route>
          <routeEntry routeIface="eth0"
routeDestination="3011:BBBB:3333:6666:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
```

```
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:0:C0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:1:459A:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:0:F0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFAA:0:E1:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3018:FFFF:0:A0:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3012:AAAA:0:12B4:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <routeEntry routeIface="eth0"
routeDestination="3011:AAAA:33BB:2211:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/>
          <!-- <routeEntry routeIface="eth0"
routeDestination="3011:CA23:2222:EF45:0:0:0:0" routePrefix="64"
routeNextHop="3018:FFFF:2:542b:54ae:9ea4:dd32:ff23"/> -->
          <routeEntry routeIface="eth2"
routeDestination="3013:BBBB:23A:8888:0:0:0:0" routePrefix="64"
routeNextHop="3013:AAAA:2aff:5555:563a:123a:bc34:8954"/>
          <routeEntry routeIface="eth1"
routeDestination="3019:CCCC:0:D0:0:0:0:0" routePrefix="64"
routeNextHop="3019:cccc:0:127b:fe2e:c0ff:0:7212"/>
        </route>
      </local>

     <local node="ar3" routePackets="on">
        <interface name="eth0" AdvSendAdvertisements="on">  <!--
ap3 -->
          <inetAddr>3019:cccc:0:d0:653e:bb26:aa0:3ff7</inetAddr>
          <AdvPrefixList>
     <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3019:cccc:0:d0:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
```

```
        </interface>
        <interface name="eth1" AdvSendAdvertisements="on"> <!--
ir2 -->
          <inetAddr>3019:cccc:0:127b:fe2e:c0ff:0:7212</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3018:FFFF:0:f0:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <route>
          <routeEntry routeIface="eth1"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3019:cccc:0:127b:fe2e:c0ff:0:34ef"/>
        </route>
       </local>
      <local node="ar4" routePackets="on">
          <interface name="eth0" AdvSendAdvertisements="on"> <!--
ap4 -->
     <inetAddr>3013:BBBB:23a:8888:bc34:8954:ee32:9f2c</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3013:BBBB:23a:8888:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <interface name="eth1" AdvSendAdvertisements="on"> <!--
ir2 -->
<inetAddr>3013:AAAA:2aff:5555:563a:123a:bc34:8954</inetAddr>
          <AdvPrefixList>
      <AdvPrefix AdvOnLinkFlag="on"
AdvRtrAddrFlag="on">3013:AAAA:2aff:5555:0:0:0:0/64</AdvPrefix>
          </AdvPrefixList>
        </interface>
        <route>
          <routeEntry routeIface="eth1"
routeDestination="0:0:0:0:0:0:0:0" routePrefix="0"
routeNextHop="3013:AAAA:2aff:5555:563a:123a:bc34:3521"/>
        </route>
       </local>
     </netconf>
```

8.27. Arquivo rvs.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE netconf SYSTEM "../../../Etc/netconf2.dtd">

<netconf>
  <local node="host0">
    <interface name="wlan" HostDupAddrDetectTransmits="0">
      <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
      <hipAddr>50:50:50:50:2580:ebff:fe8c:1448</hipAddr>
    </interface>
  </local>
  <local node="srv">
    <interface name="eth0" HostDupAddrDetectTransmits="0">

<inetAddr>3011:BBBB:3333:6666:ac24:aff:fe11:bba</inetAddr>
      <hipAddr>50:50:50:50:ac24:aff:fe11:bba</hipAddr>
    </interface>
  </local>
  <local node="mn0">
    <interface name="wlan" HostDupAddrDetectTransmits="0">
      <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
      <hipAddr>50:50:50:50:AE12:3579:F13B:B421</hipAddr>
    </interface>
  </local>
  <local node="mn1">
    <interface name="wlan" HostDupAddrDetectTransmits="0">
      <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
      <hipAddr>50:50:50:50:1234:261:EE18:5148</hipAddr>
    </interface>
  </local>
  <local node="mn2">
    <interface name="wlan" HostDupAddrDetectTransmits="0">
      <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
      <hipAddr>50:50:50:50:1111:ABCD:2468:EF0A</hipAddr>
    </interface>
  </local>
  <local node="mn3">
    <interface name="wlan" HostDupAddrDetectTransmits="0">
      <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
```

```
            <hipAddr>50:50:50:50:5634:EF29:AA76:E456</hipAddr>
          </interface>
        </local>
        <local node="mn4">
          <interface name="wlan" HostDupAddrDetectTransmits="0">
            <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
            <hipAddr>50:50:50:50:0:439A:F67D:AA32</hipAddr>
          </interface>
        </local>
        <local node="mn5">
          <interface name="wlan" HostDupAddrDetectTransmits="0">
            <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
            <hipAddr>50:50:50:50:AA32:FF19:3728:E27B</hipAddr>
          </interface>
        </local>
        <local node="mn6">
          <interface name="wlan" HostDupAddrDetectTransmits="0">
            <inetAddr>0:0:0:0:0:0:0:0</inetAddr>
            <hipAddr>50:50:50:50:F67D:B340:2288:AA23</hipAddr>
          </interface>
        </local>
        <local node="cn1">
          <interface name="eth0">

<inetAddr>3011:AAAA:33BB:2211:F67D:AA32:0:439A</inetAddr>
            <hipAddr>50:50:50:50:F67D:AA32:0:439A</hipAddr>
          </interface>
        </local>
        <local node="rvs">
          <interface name="eth0" HostDupAddrDetectTransmits="0">
            <inetAddr>3018:FFFF:0:c0:532:ff4a:fe59:1256</inetAddr>
            <hipAddr>50:50:50:50:532:ff4a:fe59:1256</hipAddr>
          </interface>
        </local>
      </netconf>
```