NS LAB 6 NOVEMBER | Lab Assignment 7

Name: Maloth Aditya
Roll No: 120CS0124

MANET

CODE:

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2011 University of Kansas
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Justin Rohrer <rohrej@ittc.ku.edu>
 *
 * James P.G. Sterbenz <jpgs@ittc.ku.edu>, director
 * ResiliNets Research Group  http://wiki.ittc.ku.edu/resilinets
 * Information and Telecommunication Technology Center (ITTC)
 * and Department of Electrical Engineering and Computer Science
 * The University of Kansas Lawrence, KS USA.
 *
 * Work supported in part by NSF FIND (Future Internet Design) Program
 * under grant CNS-0626918 (Postmodern Internet Architecture),
 * NSF grant CNS-1050226 (Multilayer Network Resilience Analysis and
Experimentation on GENI),
 * US Department of Defense (DoD), and ITTC at The University of Kansas.
 */

/*
 * This example program allows one to run ns-3 DSDV, AODV, or OLSR under
 * a typical random waypoint mobility model.
 *
 * By default, the simulation runs for 200 simulated seconds, of which
 * the first 50 are used for start-up time.  The number of nodes is 50.
 * Nodes move according to RandomWaypointMobilityModel with a speed of
 * 20 m/s and no pause time within a 300x1500 m region.  The WiFi is
 * in ad hoc mode with a 2 Mb/s rate (802.11b) and a Friis loss model.
 * The transmit power is set to 7.5 dBm.
 *
```

NS LAB 6 NOVEMBER | Lab Assignment 7

```
 * It is possible to change the mobility and density of the network by
 * directly modifying the speed and the number of nodes.  It is also
 * possible to change the characteristics of the network by changing
 * the transmit power (as power increases, the impact of mobility
 * decreases and the effective density increases).
 *
 * By default, OLSR is used, but specifying a value of 2 for the protocol
 * will cause AODV to be used, and specifying a value of 3 will cause
 * DSDV to be used.
 *
 * By default, there are 10 source/sink data pairs sending UDP data
 * at an application rate of 2.048 Kb/s each.    This is typically done
 * at a rate of 4 64-byte packets per second.  Application data is
 * started at a random time between 50 and 51 seconds and continues
 * to the end of the simulation.
 *
 * The program outputs a few items:
 * - packet receptions are notified to stdout such as:
 *   <timestamp> <node-id> received one packet from <src-address>
 * - each second, the data reception statistics are tabulated and output
 *   to a comma-separated value (csv) file
 * - some tracing and flow monitor configuration that used to work is
 *   left commented inline in the program
 */

#include <fstream>
#include <iostream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/flow-monitor-helper.h"

using namespace ns3;
using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("manet-routing-compare");

class RoutingExperiment
{
public:
  RoutingExperiment ();
```

```
  void Run (int nSinks, double txp, std::string CSVfileName);
  //static void SetMACParam (ns3::NetDeviceContainer & devices,
  //                         int slotDistance);
  std::string CommandSetup (int argc, char **argv);

private:
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node> node);
  void ReceivePacket (Ptr<Socket> socket);
  void CheckThroughput ();

  uint32_t port;
  uint32_t bytesTotal;
  uint32_t packetsReceived;

  std::string m_CSVfileName;
  int m_nSinks;
  std::string m_protocolName;
  double m_txp;
  bool m_traceMobility;
  uint32_t m_protocol;
};

RoutingExperiment::RoutingExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
    m_CSVfileName ("manet-routing.output.csv"),
    m_traceMobility (false),
    m_protocol (2) // AODV
{
}

static inline std::string
PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet, Address
senderAddress)
{
  std::ostringstream oss;

  oss << Simulator::Now ().GetSeconds () << " " << socket->GetNode ()-
>GetId ();

  if (InetSocketAddress::IsMatchingType (senderAddress))
    {
      InetSocketAddress addr = InetSocketAddress::ConvertFrom
(senderAddress);
      oss << " received one packet from " << addr.GetIpv4 ();
    }
  else
    {
```

```
      oss << " received one packet!";
    }
  return oss.str ();
}

void
RoutingExperiment::ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  Address senderAddress;
  while ((packet = socket->RecvFrom (senderAddress)))
    {
      bytesTotal += packet->GetSize ();
      packetsReceived += 1;
      NS_LOG_UNCOND (PrintReceivedPacket (socket, packet, senderAddress));
    }
}

void
RoutingExperiment::CheckThroughput ()
{
  double kbs = (bytesTotal * 8.0) / 1000;
  bytesTotal = 0;

  std::ofstream out (m_CSVfileName.c_str (), std::ios::app);

  out << (Simulator::Now ()).GetSeconds () << ","
      << kbs << ","
      << packetsReceived << ","
      << m_nSinks << ","
      << m_protocolName << ","
      << m_txp << ""
      << std::endl;

  out.close ();
  packetsReceived = 0;
  Simulator::Schedule (Seconds (1.0), &RoutingExperiment::CheckThroughput,
this);
}

Ptr<Socket>
RoutingExperiment::SetupPacketReceive (Ipv4Address addr, Ptr<Node> node)
{
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (node, tid);
  InetSocketAddress local = InetSocketAddress (addr, port);
  sink->Bind (local);
  sink->SetRecvCallback (MakeCallback (&RoutingExperiment::ReceivePacket,
this));
```

```
  return sink;
}

std::string
RoutingExperiment::CommandSetup (int argc, char **argv)
{
  CommandLine cmd (__FILE__);
  cmd.AddValue ("CSVfileName", "The name of the CSV output file name",
m_CSVfileName);
  cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
  cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
  cmd.Parse (argc, argv);
  return m_CSVfileName;
}

int
main (int argc, char *argv[])
{
  RoutingExperiment experiment;
  std::string CSVfileName = experiment.CommandSetup (argc,argv);

  //blank out the last output file and write the column headers
  std::ofstream out (CSVfileName.c_str ());
  out << "SimulationSecond," <<
  "ReceiveRate," <<
  "PacketsReceived," <<
  "NumberOfSinks," <<
  "RoutingProtocol," <<
  "TransmissionPower" <<
  std::endl;
  out.close ();

  int nSinks = 10;
  double txp = 7.5;

  experiment.Run (nSinks, txp, CSVfileName);
}

void
RoutingExperiment::Run (int nSinks, double txp, std::string CSVfileName)
{
  //CommandLine cmd;
  //cmd.AddValue("CSVfileName","The name of the CSV output file name",
m_CSVfileName);
  //cmd.AddValue("traceMobility","Enable mobility tracing", m_traceMobility);
  //cmd.AddValue("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
  Packet::EnablePrinting ();
  m_nSinks = nSinks;
```

```
  m_txp = txp;
  m_CSVfileName = CSVfileName;

  int nWifis = 50;

  double TotalTime = 200.0;
  std::string rate ("2048bps");
  std::string phyMode ("DsssRate11Mbps");
  std::string tr_name ("manet-routing-compare");
  int nodeSpeed = 20; //in m/s
  int nodePause = 0; //in s
  m_protocolName = "protocol";

  Config::SetDefault  ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
  Config::SetDefault ("ns3::OnOffApplication::DataRate",  StringValue (rate));

  //Set Non-unicastMode rate to unicast mode
  Config::SetDefault
("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue (phyMode));

  NodeContainer adhocNodes;
  adhocNodes.Create (nWifis);

  // setting up wifi phy and channel using helpers
  WifiHelper wifi;
  wifi.SetStandard (WIFI_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy;
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay
("ns3::ConstantSpeedPropagationDelayModel");
  wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));

  wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
  wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));

  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac,
adhocNodes);

  MobilityHelper mobilityAdhoc;
  int64_t streamIndex = 0; // used to get consistent mobility across scenarios
```

```
  ObjectFactory pos;
  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
  pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|
Max=300.0]"));
  pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|
Max=1500.0]"));

  Ptr<PositionAllocator> taPositionAlloc = pos.Create ()-
>GetObject<PositionAllocator> ();
  streamIndex += taPositionAlloc->AssignStreams (streamIndex);

  std::stringstream ssSpeed;
  ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" << nodeSpeed
<< "]";
  std::stringstream ssPause;
  ssPause << "ns3::ConstantRandomVariable[Constant=" << nodePause <<
"]";
  mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                                  "Speed", StringValue (ssSpeed.str ()),
                                  "Pause", StringValue (ssPause.str ()),
                                  "PositionAllocator", PointerValue (taPositionAlloc));
  mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
  mobilityAdhoc.Install (adhocNodes);
  streamIndex += mobilityAdhoc.AssignStreams (adhocNodes, streamIndex);
  NS_UNUSED (streamIndex); // From this point, streamIndex is unused

  AodvHelper aodv;
  OlsrHelper olsr;
  DsdvHelper dsdv;
  DsrHelper dsr;
  DsrMainHelper dsrMain;
  Ipv4ListRoutingHelper list;
  InternetStackHelper internet;

  switch (m_protocol)
    {
    case 1:
      list.Add (olsr, 100);
      m_protocolName = "OLSR";
      break;
    case 2:
      list.Add (aodv, 100);
      m_protocolName = "AODV";
      break;
    case 3:
      list.Add (dsdv, 100);
      m_protocolName = "DSDV";
      break;
```

```
    case 4:
      m_protocolName = "DSR";
      break;
    default:
      NS_FATAL_ERROR ("No such protocol:" << m_protocol);
    }

  if (m_protocol < 4)
    {
      internet.SetRoutingHelper (list);
      internet.Install (adhocNodes);
    }
  else if (m_protocol == 4)
    {
      internet.Install (adhocNodes);
      dsrMain.Install (dsr, adhocNodes);
    }

  NS_LOG_INFO ("assigning ip address");

  Ipv4AddressHelper addressAdhoc;
  addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer adhocInterfaces;
  adhocInterfaces = addressAdhoc.Assign (adhocDevices);

  OnOffHelper onoff1 ("ns3::UdpSocketFactory",Address ());
  onoff1.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1.0]"));
  onoff1.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0.0]"));

  for (int i = 0; i < nSinks; i++)
    {
      Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i),
adhocNodes.Get (i));

      AddressValue remoteAddress (InetSocketAddress
(adhocInterfaces.GetAddress (i), port));
      onoff1.SetAttribute ("Remote", remoteAddress);

      Ptr<UniformRandomVariable> var =
CreateObject<UniformRandomVariable> ();
      ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i + nSinks));
      temp.Start (Seconds (var->GetValue (100.0,101.0)));
      temp.Stop (Seconds (TotalTime));
    }

  std::stringstream ss;
  ss << nWifis;
```

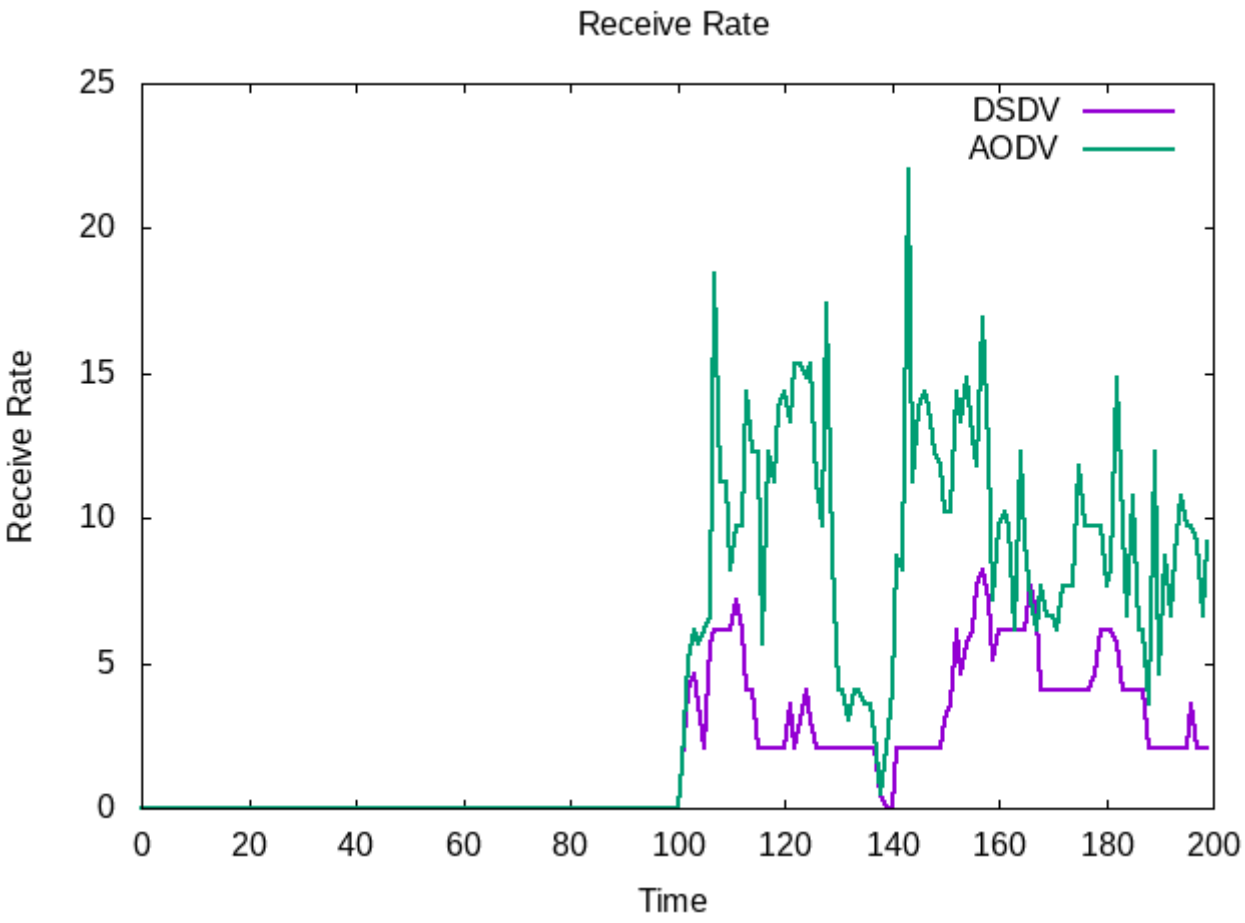
```
  std::string nodes = ss.str ();

  std::stringstream ss2;
  ss2 << nodeSpeed;
  std::string sNodeSpeed = ss2.str ();

  std::stringstream ss3;
  ss3 << nodePause;
  std::string sNodePause = ss3.str ();

  std::stringstream ss4;
  ss4 << rate;
  std::string sRate = ss4.str ();

  //NS_LOG_INFO ("Configure Tracing.");
  //tr_name = tr_name + "_" + m_protocolName +"_" + nodes + "nodes_" +
sNodeSpeed + "speed_" + sNodePause + "pause_" + sRate + "rate";

  //AsciiTraceHelper ascii;
  //Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream ( (tr_name +
".tr").c_str());
  //wifiPhy.EnableAsciiAll (osw);
  AsciiTraceHelper ascii;
  //MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name + ".mob"));
  MobilityHelper::EnableAsciiAll (ascii.CreateFileStream ("lab7.tr"));

  Ptr<FlowMonitor> flowmon;
  FlowMonitorHelper flowmonHelper;
  flowmon = flowmonHelper.InstallAll ();


  NS_LOG_INFO ("Run Simulation.");

  CheckThroughput ();

  Simulator::Stop (Seconds (TotalTime));
  Simulator::Run ();

  flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(), false, false);

  Simulator::Destroy ();
}
```

OUTPUT:

a. Receive Rate

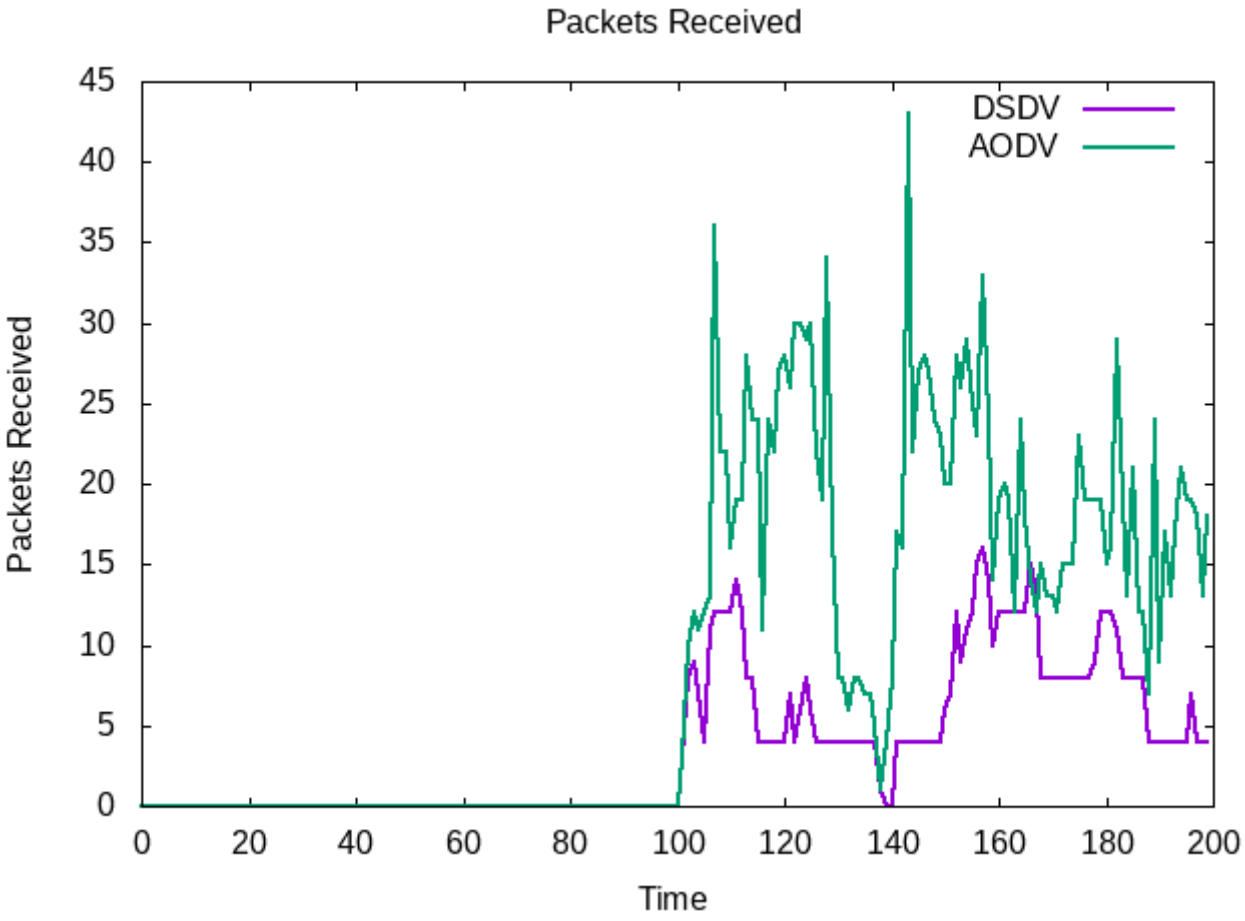
NS LAB 6 NOVEMBER | Lab Assignment 7

NS LAB 6 NOVEMBER | Lab Assignment 7



b. Packets Received

NS LAB 6 NOVEMBER | Lab Assignment 7



Based on the above observations, it is concluded that AODV is the best protocol.