ET4394

Wireless Networking

NS3 Assignment Report

April 30, 2016

Delft University of Technology

Authors: Shruthi Kashyap (4518438): s.kashyap@student.tudelft.nl

TU Delft Delft University of Technology

Challenge the future

Contents

1 Introduction

IEEE 802.11 is one of the most widely and rapidly adopted wireless communication standard. It adopts Ethernet style networking to radio links. The CSMA/CA method is used for data transmission in the IEEE 802.11 standard. In this method, the node first senses the medium and checks if it is free before transmitting data. It then waits for an acknowledgment for data reception. If there is a collision and the node does not receive any acknowledgment, it waits for a certain period called the back-off time before sensing the medium again and re-transmitting the packet when the medium is free.

The IEEE 802.11b is a popular standard of wireless communication that operates in the 2.4GHz ISM band and has an indoor coverage area of about 30m. It uses Direct Sequence Spread Spectrum (DSSS) technique with Complementary Code Keying (CCK) modulation scheme and offers throughput up to 11 Mbps using the same 2.4GHz band. However, it is not always possible to achieve this peak data rate in practical conditions because of the presence of propagation loss and interference factor. To improve resiliency, it is possible for the system to adopt to lower data rates like 5.5Mbps, 2Mbps and 1Mbps.

2 Project Description

2.1 Objective

In this project, the performance of a Wifi IEEE 802.11b network is evaluated using network simulation, by varying different network parameters like data rate, payload size, number of nodes, node mobility, control rate algorithm, etc. The effect these parameters have on the throughput of the network is measured and analyzed.

2.2 Hypothesis

The throughput is closely related to the factors such as data rate, payload size, number of nodes, node mobility, control rate algorithm, etc. As the number of nodes in the network increases, the throughput decreases. This happens because the contention in the channel increases leading to decrease in the average throughput per node. This implies that the maximum throughput is achieved when there is only one node in the network.

On the contrary, as the payload size increases, the throughput increases. Also, when the data rate increases, throughput improves as the average number of packets transmitted per second increases.

3 Implementation

In this project, the network simulation is done using the NS3 simulator. It supports IP based wireless model simulation for Wifi in layer 1 to 3 with various routing protocols and other network parameters. The implementation of the simulation can be done using both C++ and python languages.

Figure 1 shows the topology of the IEEE 802.11b network that is chosen to perform the simulation. It is a Wifi 802.11b infrastructure-based topology having one access point and N nodes.

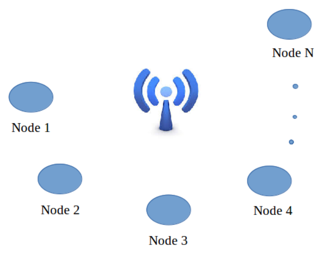


Figure 1: IEEE 802.11b Network Topology

To build this topology, the access point and the nodes need to be created using the NodeContainer class. The 802.11b standard is defined using the WifiHelper. The channel between the nodes and the access point is created using the YansWifiPhyHelper and YansWifiChannelHelper, and the propagation loss and propagation delay are specified.

Then the MAC parameter configuration should be done by specifying the data rate and control rate algorithms. If the data rate and control rate algorithms are not specified, default values of 11Mbps and ConstantRate algorithm respectively, will be taken. This is followed by configuration of SSID and NetDeviceContainer for the nodes and the access point.

Next, the mobility configuration is done to all the nodes and the access point using the wireless mobility model. This affects the layer 1 to 3. The mobility of the access point is made constant and the mobility of other nodes are chosen to be one of the following:

- Constant Mobility Model:The positions of the objects do not change compared to their initial positions.

- Random Direction 2D Mobility Model: The objects move in random direction with speed in a specific direction. When the objects hit a boundary, they wait with some delay and then start moving in new direction with new speed.

- Random Walk 2D Mobility Model: The objects move in random directions with random speeds in a rectangular boundary.

The Internet stack and IP addresses are then configured for each node and the access point using InternetStackHelper and Ipv4AddressHelper. The InternetStackHelper is used for the IP, TCP or UDP aggregation. The UDP traffic is being simulated in this project. The UdpServerHelper and the UdpClientHelper are used to generate the UDP traffic in the network. The simulation time is defined by the ApplicationContainer.

The FlowMonitor is used to measure the performance of the network. It provides end to end flow statistics by tracking the number of packets exchanged in the network. The delay and throughput can be calculated from the measurements obtained and are calculated by the formulas provided below.

$$Throughput = \frac{rxBytes * 8.0}{(timeLastRxPacket - timeFirstTxPacket) * 1024 * NumNodes}(kbps) \tag{1}$$

$$Delay = \frac{delaySum}{rxPackets}(ns) \tag{2}$$

To simulate a more realistic environment, randomness is introduced using the RngSeedManager class. The randomness is created based on timestamp.

4 Results and Analysis

The following cases are simulated and analyzed:

- Network Performance versus Payload Size
- Network Performance versus Data Rate
- Network Performance versus Node Mobility
- Network Performance versus Control Rate Algorithm

The waf, a python based framework, is used for building and running the simulation.

4.1 Network Performance versus Payload Size

The payload size is the actual size of the data that is carried by the packet during transmission. The maximum payload size allowed by the UDP is 1472 bytes. The variation in the performance for different payload sizes of 1472 bytes, 1024 bytes and 512 bytes are measured. The simulation is carried out for 1 to 5 nodes and the average throughput obtained is plotted as shown in the figure 2.

It can be observed that for every node, as the payload size increases, the average throughput also increases. This is because when the payload size is small, the number of bytes that are transmitted within the simulation period will be small, consequently the number of successfully received bytes will also be small. To achieve maximum throughput, the payload size should be kept at its maximum value of 1472 bytes in case of UDP traffic. Beyond this, the compiler will give an error.

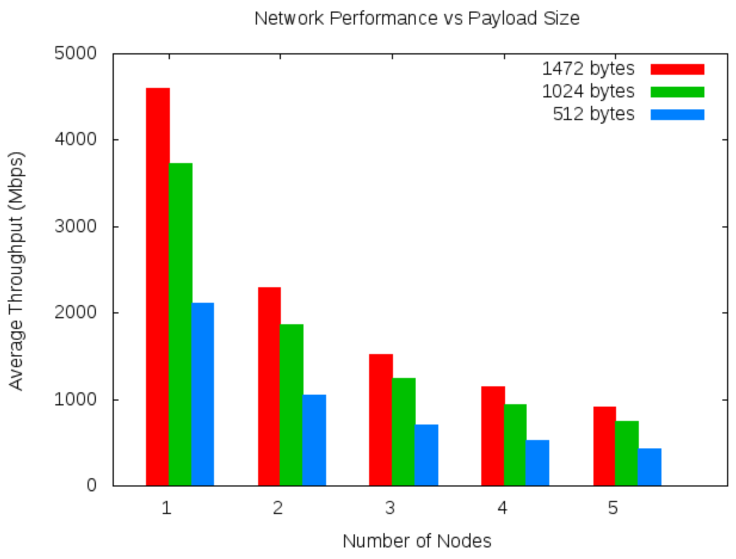


Figure 2: Average throughput for different payload sizes

4.2 Network Performance versus Data Rate

The IEEE 802.11b standard supports four data rates - 11Mbps, 5.5Mbps, 2Mbps and 1Mbps. The simulation is performed for nodes varying from 1 to 5 with different data rates. The average throughput in each case is measured and plotted as shown in figure 3. The figure also shows how the number of nodes affects the performance of the network.

The figure 3 shows that as the number of nodes in the network increases, the throughput decreases exponentially. The maximum throughput is achieved when there is only one node in the network. Also, the achieved throughput is the highest for the data rate of 11Mbps. It can be observed that as the number of nodes increases, the difference between the throughput achieved with different data rates becomes less significant.

4.3 Network Performance versus Node Mobility

The network has been simulated with three different mobility models - Constant Mobility Model, Random Direction 2D Mobility Model and Random Walk 2D Mobility Model. The number of nodes is taken as 5. The average throughput and delay are measured as shown in figure 4.

The figure shows that all the models give almost the same performance. The Constant Mobility Model achieves slightly higher performance than the other models. The reason is that the nodes in Constant Mobility Model are fixed, so the average throughput mostly remains the same throughout the simulation. Similarly the values of delay obtained in all the models are also almost the same.

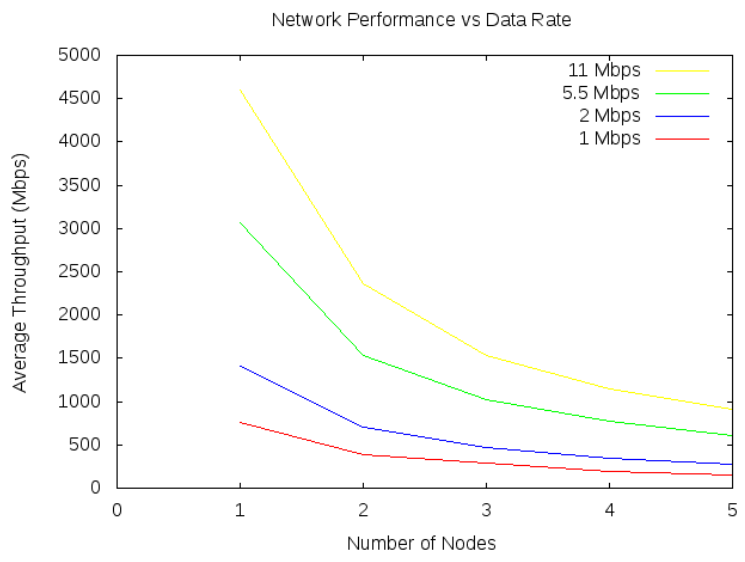
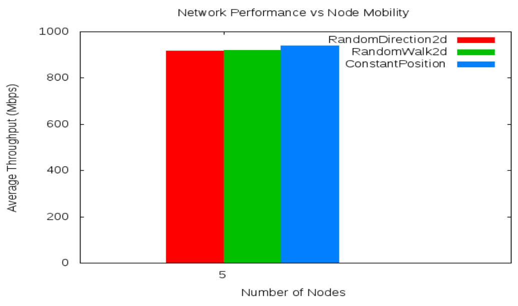
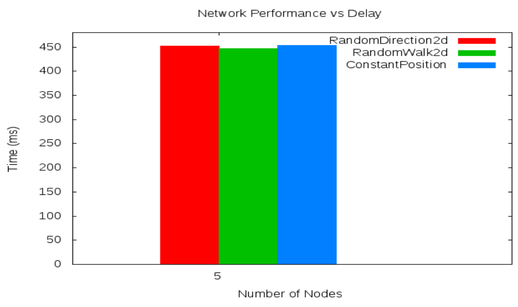


Figure 3: Average throughput for different data rates



(a) Average throughput for different mobility models



(b) Average delay in different mobility models

Figure 4

4.4 Network Performance versus Control Rate Algorithm

The control rate algorithm in NS3 is used to manage the rate of data transmission. The available control rate algorithms in NS3 are Aarf, Arf, Cara, Onoe, Rraa, ConstantRate and Minstrel. The ConstantRate, Aarf, Arf and Cara have been simulated. The results are as shown in the figure 5.

From the figure it can be observed that the ConstantRate algorithm provides the maximum throughput followed by Aarf and Arf. The Cara provides the lowest throughput compared to other algorithms. The ConstantRate algorithm always provides the same data rate as specified by the user, hence the throughput will always be predictable and high compared to other algorithms. Whereas, the Aarf and the Arf algorithms vary the data rate dynamically based on the number of successful transmissions. This leads to a slight lowering of throughput.

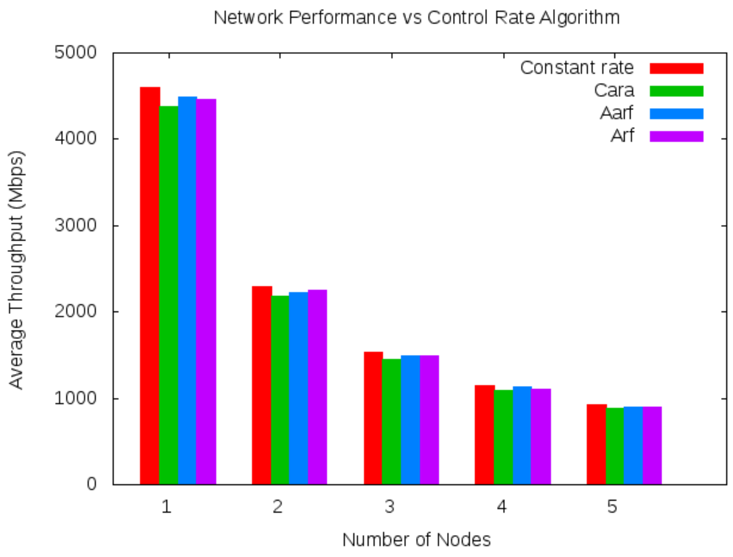


Figure 5: Average throughput for different control rate algorithms

5 Conclusion

The simulation is performed for Wifi 802.11b infrastructure-based network having a maximum of 5 nodes with UDP traffic. The simulation time is set to 10s. the performance of the network is measured by varying the payload size, data rate, number of nodes, node mobility and control rate algorithms.From the results of the simulation, it can be concluded that,

- Maximum performance is obtained with a payload size of 1472 bytes with UDP protocol. As the payload size increases, the throughput of the network also increases.

- With more number of nodes in the network, the performance degrades and the average throughput per node reduces. The throughput decreases exponentially with the increase in number of nodes.

- The higher the data rate, the higher will be the performance. The maximum throughput is achieved with a data rate of 11Mbps.

- The Constant Mobility Model has slightly better performance than the Random Direction 2D Mobility Model and Random Walk 2D Mobility Model. The delay incurred is almost the same among all the models considered.

- The ConstantRate algorithm gives the highest throughput among the selected control rate algorithms. The Aarf and Arf algorithms also show good performance. The Cara algorithm has the least performance among all.

6 References

1. https://www.nsnam.org/documentation/
2. https://www.nsnam.org/docs/models/html/flow-monitor.html
3. 802.11 Wireless Networks, The Definitive Guide - Matthew S. Gast

7 Appendix

File: throughput.cc

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/random-variable-stream.h"
#include "ns3/gnuplot.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Throughput Measurement");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nWifi = 0;
  bool tracing = true;
  double startTime = 0.0;
  double stopTime = 10.0;
  uint32_t payloadSize = 0;
  uint32_t maxPacket = 10000;
  uint32_t rate = 0;
  uint32_t cRateAlgo = 0;
  uint32_t mobMod = 0;
  StringValue dataRate;

  // Create randomness based on time
  time_t timex;
  time(&timex);
  RngSeedManager::SetSeed(timex);
  RngSeedManager::SetRun(1);

  CommandLine cmd;
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("rate", "Data rate", rate);
  cmd.AddValue ("payloadSize", "Size of the payloadSize", payloadSize);
  cmd.AddValue ("cRateAlgo", "Control rate alogrithm used", cRateAlgo);
  cmd.AddValue ("mobMod", "Mobility model used", mobMod);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);
```

```
// Checking if valid number of csma or wifi nodes are specified
if (nWifi > 250 )
{
  std::cout << "Too many wifi nodes. Cannot create more than 250." << std::endl;
  return 1;
}

// Set default number of nodes if not specified
if (nWifi == 0)
  nWifi = 5;

// Set default payloadSize if not specified
if (payloadSize == 0)
 payloadSize = 1472;

// Set data rate
switch(rate)
{
  case 11:
    dataRate = StringValue("DsssRate11Mbps");
    std::cout << "11" << "\t";
    break;
  case 5:
    dataRate = StringValue("DsssRate5_5Mbps");
    std::cout << "5.5" << "\t";
    break;
  case 2:
    dataRate = StringValue("DsssRate2Mbps");
    std::cout << "2" << "\t";
    break;
  case 1:
    dataRate = StringValue("DsssRate1Mbps");
    std::cout << "1" << "\t";
    break;
  default:
    dataRate = StringValue("DsssRate11Mbps");
    std::cout << "11" << "\t";
    break;
}

std::cout << nWifi << "\t";
std::cout << payloadSize << "\t";

if (verbose)
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}
```

```
// Creating nodes
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode;
wifiApNode.Create (1);

//std::cout << "The nodes and access point are created" << std::endl;
NS_LOG_INFO("The nodes and access point are created");

// Creating channel
YansWifiChannelHelper channel;
channel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
channel.AddPropagationLoss ("ns3::FriisPropagationLossModel");

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.Set ("RxGain", DoubleValue (0));
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
//std::cout << "The channel is created" << std::endl;
NS_LOG_INFO("The channel is created");

// Configuring MAC parameters
// Set control rate algorithm
//1 : ConstantRate
//2 : Cara
//3 : Aarf
//4 : Arf
switch(cRateAlgo)
{
  case 1:
    std::cout << "ConstantRate" << "\t";
    wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", dataRate,
    "ControlMode", dataRate);
    break;
  case 2:
    std::cout << "Cara" << "\t";
    wifi.SetRemoteStationManager ("ns3::CaraWifiManager");
    break;
  case 3:
    wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
    std::cout << "Aarf" << "\t";
    break;
  case 4:
    std::cout << "Arf" << "\t";
    wifi.SetRemoteStationManager ("ns3::ArfWifiManager");
    break;
  default:
    std::cout << "ConstantRate" << "\t";
    wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", dataRate, "Control
    break;
}

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
//std::cout << "The control rate is configured" << std::endl;
NS_LOG_INFO("The control rate is configured");

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac","Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
//std::cout << "SSID, apdevice and stadevices are configured" << std::endl;
NS_LOG_INFO("SSID, apdevice and stadevices are configured");

// Configuring mobility of nodes
MobilityHelper mobility;

// Set mobility model
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
     "MinX", DoubleValue (0.0),
     "MinY", DoubleValue (0.0),
     "DeltaX", DoubleValue (10.0),
     "DeltaY", DoubleValue (10.0),
     "GridWidth", UintegerValue (5),
     "LayoutType", StringValue ("RowFirst"));

//1 : RandomDirection2d
//2 : RandomWalk2d
//3 : ConstantPosition
switch(mobMod)
{
  case 1:
    std::cout << "RandomDirection2d" << "\t";
    mobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel", "Bounds",
    RectangleValue (Rectangle (-10, 10, -10, 10)), "Speed",
    StringValue ("ns3::ConstantRandomVariable[Constant=3]"), "Pause",
    StringValue ("ns3::ConstantRandomVariable[Constant=0.4]"));
    break;
  case 2:
    std::cout << "RandomWalk2d" << "\t";
    mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
    "Bounds", RectangleValue (Rectangle (-1000, 1000, -1000, 1000)),
    "Distance", ns3::DoubleValue (300.0));
    break;
  case 3:
    std::cout << "ConstantPosition" << "\t";
    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
    break;
  default:
    std::cout << "RandomWalk2d" << "\t";
    break;
}

mobility.Install (wifiStaNodes);

// Configuring mobility of AP device
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
//std::cout << "Mobility of nodes configured" << std::endl;
NS_LOG_INFO("Mobility of nodes configured");

// Adding internet stack
InternetStackHelper stack;
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

// Configuring IP addresses
Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer Stainterfaces = address.Assign (staDevices);
Ipv4InterfaceContainer Apinterfaces = address.Assign (apDevices);

//for(unsigned int i = 0; i < nWifi; i++)
//{
//  std::cout << "Node" << i+1 << "IP Address = " << Stainterfaces.GetAddress(i) << std::endl;
//}

//std::cout << "IP Address of AP = " << Apinterfaces.GetAddress(0) << std::endl;
//std::cout << "Internet Stack and IP addresses are configured" << std::endl;

// Creating traffic generator
ApplicationContainer serverApps;
UdpServerHelper udpServer (4001); //port 4001
serverApps = udpServer.Install (wifiStaNodes.Get (0));
serverApps.Start (Seconds(startTime));
serverApps.Stop (Seconds(stopTime));
```

```
UdpClientHelper udpClient (Apinterfaces.GetAddress (0), 4001); //port 4001
udpClient.SetAttribute ("MaxPackets", UintegerValue (maxPacket));
udpClient.SetAttribute ("Interval", TimeValue (Time ("0.002"))); //packets
udpClient.SetAttribute ("PacketSize", UintegerValue (payloadSize));

ApplicationContainer clientApps = udpClient.Install (wifiStaNodes.Get (0));
clientApps.Start (Seconds(startTime));
clientApps.Stop (Seconds(stopTime+5));
//std::cout << "UDP traffic is generated" << std::endl;
NS_LOG_INFO("UDP traffic is generated");

// Using FlowMonitor to measure throughput and delay
FlowMonitorHelper fMonitor;
Ptr<FlowMonitor> monitorPtr = fMonitor.InstallAll();

Simulator::Stop (Seconds(stopTime+2));
Simulator::Run ();

monitorPtr->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> flowClassifier =
DynamicCast<Ipv4FlowClassifier> (fMonitor.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> flowStats = monitorPtr->GetFlowStats ();

double intrThrghpt = 0.0;
Time delay;
double timeDiff = 0.0;
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = flowStats.begin ();
i != flowStats.end (); ++i)
{
  intrThrghpt = 0.0;
  //Ipv4FlowClassifier::FiveTuple tuple = flowClassifier->FindFlow (i->first);
  //std::cout << "Flow " << i->first  << " (" <<
  tuple.sourceAddress << " -> " << tuple.destinationAddress << ")\n";
  //std::cout << i->second.txBytes << "\t";
  //std::cout << i->second.rxBytes << "\t";

  timeDiff = i->second.timeLastRxPacket.GetSeconds()
  - i->second.timeFirstTxPacket.GetSeconds();
  if (timeDiff > 0)
    intrThrghpt = i->second.rxBytes  *  8.0  / timeDiff / 1024 / nWifi;

  std::cout << intrThrghpt  << "\t";

  delay = i->second.delaySum / i->second.rxPackets;
  std::cout << delay.GetMilliSeconds() << "\n";
}

if (tracing == true)
```

```
  {
    phy.EnablePcap ("third", apDevices.Get (0));
  }

  Simulator::Destroy ();
  return 0;
}
```