1 A. Trace Domain Name Server using packet sniffer and packet analyser.

B. Design and simulate a wired network with duplex links between 3 nodes with CDR over UDP. Set the queue size vary the bandwidth and find the number of packets dropped.

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("TrafficControlExample");

int

  main (int argc, char \*argv[])

  {

    double simulationTime = 10; //seconds

    std::string transportProt = "Udp";

    std::string socketType;

    CommandLine cmd;

    cmd.AddValue ("transportProt", "Transport protocol to use: Tcp, Udp", transportProt);

    cmd.Parse (argc, argv);

    if (transportProt.compare ("Tcp") == 0)

    {

      socketType = "ns3::TcpSocketFactory";

    }

    else

    {

      socketType = "ns3::UdpSocketFactory";

    }

    NodeContainer nodes;

    nodes.Create (3);

    PointToPointHelper pointToPoint;

    pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));

    pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

    pointToPoint.SetQueue ("ns3::DropTailQueue", "MaxSize", StringValue ("1p"));

    NetDeviceContainer devices01;

    devices01 = pointToPoint.Install (nodes.Get(0),nodes.Get(1));

    NetDeviceContainer devices12;

    devices12 = pointToPoint.Install (nodes.Get(1),nodes.Get(2));

    InternetStackHelper stack;

    stack.Install (nodes);

    Ipv4AddressHelper address;

    address.SetBase ("10.1.1.0", "255.255.255.0");

    Ipv4InterfaceContainer interfaces01 = address.Assign (devices01);

    address.SetBase ("10.1.2.0", "255.255.255.0");

    //flow

    Ipv4InterfaceContainer interfaces12 = address.Assign (devices12);

    Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

    uint16\_t port = 7;

    Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

    PacketSinkHelper packetSinkHelper (socketType, localAddress);

    ApplicationContainer sinkApp = packetSinkHelper.Install (nodes.Get (2));

    sinkApp.Start (Seconds (0.0));

    sinkApp.Stop (Seconds (simulationTime + 0.1));

  uint32\_t payloadSize = 1448;

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff (socketType, Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (interfaces12.GetAddress (1), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (nodes.Get (0)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (simulationTime + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (simulationTime + 5));

  Simulator::Run ();

  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  std::cout << std::endl << "\*\*\* Flow monitor statistics \*\*\*" << std::endl;

  std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets

            << " / " << stats[1].txBytes << std::endl;

  std::cout << "  Offered Load: " << stats[1].txBytes \* 8.0 / (stats[1].timeLastTxPacket.GetSeconds () - stats[1].timeFirstTxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets

            << " / " << stats[1].rxBytes << std::endl;

  uint32\_t packetsDroppedByQueueDisc = 0;

  uint64\_t bytesDroppedByQueueDisc = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE\_DISC)

    {

      packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

      bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

    }

  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc

            << " / " << bytesDroppedByQueueDisc << std::endl;

  Simulator::Destroy ();

  return 0;

}

2 A. Trace DHCP using packet sniffer and packet analyser.

B. Design a wired network with ‘n’ nodes to observe the performance of two TCP variants (where one is Reno). Simulate the designed network and observe the network performance.

=> Do by yourself its easy

3 A. Trace Internet Protocol and Internet Control Message Protocol using packet sniffer and packet analyser.

B. Design and simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of packets.

/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

 \* 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

 \*/

#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/network-module.h"

#include "ns3/applications-module.h"

#include "ns3/mobility-module.h"

#include "ns3/csma-module.h"

#include "ns3/internet-module.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/ssid.h"

#include "ns3/flow-monitor-module.h"

// Default Network Topology

//

//   Wifi 10.1.3.0

//                 AP

//  \*    \*    \*    \*

//  |    |    |    |    10.1.1.0

// n5   n6   n7   n0 -------------- n1   n2   n3   n4

//                   point-to-point  |    |    |    |

//                                   ================

//                                     LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("ThirdScriptExample");

int

main (int argc, char \*argv[])

{

  bool verbose = true;

  uint32\_t nCsma = 3;

  uint32\_t nWifi = 3;

  bool tracing = false;

  CommandLine cmd;

  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);

  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);

  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);

  // The underlying restriction of 18 is due to the grid position

  // allocator's configuration; the grid layout will exceed the

  // bounding box if more than 18 nodes are provided.

  if (nWifi > 18)

    {

      std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;

      return 1;

    }

  if (verbose)

    {

      LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

      LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

    }

  NodeContainer p2pNodes;

  p2pNodes.Create (2);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer p2pDevices;

  p2pDevices = pointToPoint.Install (p2pNodes);

  NodeContainer csmaNodes;

  csmaNodes.Add (p2pNodes.Get (1));

  csmaNodes.Create (nCsma);

  CsmaHelper csma;

  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));

  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;

  csmaDevices = csma.Install (csmaNodes);

  NodeContainer wifiStaNodes;

  wifiStaNodes.Create (nWifi);

  NodeContainer wifiApNode = p2pNodes.Get (0);

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

  phy.SetChannel (channel.Create ());

  WifiHelper wifi;

  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  WifiMacHelper mac;

  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);

  MobilityHelper mobility;

  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

                                 "MinX", DoubleValue (0.0),

                                 "MinY", DoubleValue (0.0),

                                 "DeltaX", DoubleValue (5.0),

                                 "DeltaY", DoubleValue (10.0),

                                 "GridWidth", UintegerValue (3),

                                 "LayoutType", StringValue ("RowFirst"));

  mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",

                             "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));

  mobility.Install (wifiStaNodes);

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

  mobility.Install (wifiApNode);

  InternetStackHelper stack;

  stack.Install (csmaNodes);

  stack.Install (wifiApNode);

  stack.Install (wifiStaNodes);

  Ipv4AddressHelper address;

  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer p2pInterfaces;

  p2pInterfaces = address.Assign (p2pDevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");

  Ipv4InterfaceContainer csmaInterfaces;

  csmaInterfaces = address.Assign (csmaDevices);

  address.SetBase ("10.1.3.0", "255.255.255.0");

  address.Assign (staDevices);

  address.Assign (apDevices);

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

 uint16\_t port = 7;

  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

  PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", localAddress);

  ApplicationContainer sinkApp = packetSinkHelper.Install (csmaNodes.Get (0));

  sinkApp.Start (Seconds (0.0));

  sinkApp.Stop (Seconds (10 + 0.1));

  uint32\_t payloadSize = 1448;

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff ("ns3::UdpSocketFactory", Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (csmaInterfaces.GetAddress (0), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (csmaNodes.Get (1)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (simulationTime + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (10 + 5));

  Simulator::Run ();

  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  std::cout << std::endl << "\*\*\* Flow monitor statistics \*\*\*" << std::endl;

  std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets

            << " / " << stats[1].txBytes << std::endl;

  std::cout << "  Offered Load: " << stats[1].txBytes \* 8.0 / (stats[1].timeLastTxPacket.GetSeconds () - stats[1].timeFirstTxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets

            << " / " << stats[1].rxBytes << std::endl;

  uint32\_t packetsDroppedByQueueDisc = 0;

  uint64\_t bytesDroppedByQueueDisc = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE\_DISC)

    {

      packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

      bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

    }

  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc

            << " / " << bytesDroppedByQueueDisc << std::endl;

  Simulator::Run ();

  Simulator::Destroy ();

  return 0;

}

Or

/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

 \* 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

 \*/

#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/network-module.h"

#include "ns3/applications-module.h"

#include "ns3/mobility-module.h"

#include "ns3/csma-module.h"

#include "ns3/internet-module.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/ssid.h"

#include "ns3/flow-monitor-module.h"

// Default Network Topology

//

//   Wifi 10.1.3.0

//                 AP

//  \*    \*    \*    \*

//  |    |    |    |    10.1.1.0

// n5   n6   n7   n0 -------------- n1   n2   n3   n4

//                   point-to-point  |    |    |    |

//                                   ================

//                                     LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("ThirdScriptExample");

int

main (int argc, char \*argv[])

{

  bool verbose = true;

  uint32\_t nCsma = 3;

  uint32\_t nWifi = 3;

  bool tracing = false;

  CommandLine cmd;

  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);

  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);

  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);

  // The underlying restriction of 18 is due to the grid position

  // allocator's configuration; the grid layout will exceed the

  // bounding box if more than 18 nodes are provided.

  if (nWifi > 18)

    {

      std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;

      return 1;

    }

  if (verbose)

    {

      LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

      LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

    }

  NodeContainer p2pNodes;

  p2pNodes.Create (2);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer p2pDevices;

  p2pDevices = pointToPoint.Install (p2pNodes);

  NodeContainer csmaNodes;

  csmaNodes.Add (p2pNodes.Get (1));

  csmaNodes.Create (nCsma);

  CsmaHelper csma;

  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));

  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;

  csmaDevices = csma.Install (csmaNodes);

  NodeContainer wifiStaNodes;

  wifiStaNodes.Create (nWifi);

  NodeContainer wifiApNode = p2pNodes.Get (0);

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

  phy.SetChannel (channel.Create ());

  WifiHelper wifi;

  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  WifiMacHelper mac;

  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);

  MobilityHelper mobility;

  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

                                 "MinX", DoubleValue (0.0),

                                 "MinY", DoubleValue (0.0),

                                 "DeltaX", DoubleValue (5.0),

                                 "DeltaY", DoubleValue (10.0),

                                 "GridWidth", UintegerValue (3),

                                 "LayoutType", StringValue ("RowFirst"));

  mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",

                             "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));

  mobility.Install (wifiStaNodes);

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

  mobility.Install (wifiApNode);

  InternetStackHelper stack;

  stack.Install (csmaNodes);

  stack.Install (wifiApNode);

  stack.Install (wifiStaNodes);

  Ipv4AddressHelper address;

  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer p2pInterfaces;

  p2pInterfaces = address.Assign (p2pDevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");

  Ipv4InterfaceContainer csmaInterfaces;

  csmaInterfaces = address.Assign (csmaDevices);

  address.SetBase ("10.1.3.0", "255.255.255.0");

  address.Assign (staDevices);

  address.Assign (apDevices);

  //Flow

  uint16\_t port = 7;

  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

  PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", localAddress);

  ApplicationContainer sinkApp = packetSinkHelper.Install (csmaNodes.Get (1));

  sinkApp.Start (Seconds (0.0));

  sinkApp.Stop (Seconds (10 + 0.1));

  uint32\_t payloadSize = 1448;

  //Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff ("ns3::UdpSocketFactory", Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (csmaInterfaces.GetAddress (1), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (wifiStaNodes.Get (0)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (10 + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (10 + 5));

  Simulator::Run ();

  Simulator::Run ();

  Simulator::Destroy ();

  return 0;

}

4 A. Trace HTTP using packet sniffer and packet analyser.

B. Design and simulate infrastructure less network, generate two traffic flows between nodes and analyse its performance.

/\* -\*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

 \* Copyright (c) 2009 The Boeing Company

 \*

 \* 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

 \*

 \*/

// This script configures two nodes on an 802.11b physical layer, with

// 802.11b NICs in adhoc mode, and by default, sends one packet of 1000

// (application) bytes to the other node.  The physical layer is configured

// to receive at a fixed RSS (regardless of the distance and transmit

// power); therefore, changing position of the nodes has no effect.

//

// There are a number of command-line options available to control

// the default behavior.  The list of available command-line options

// can be listed with the following command:

// ./waf --run "wifi-simple-adhoc --help"

//

// For instance, for this configuration, the physical layer will

// stop successfully receiving packets when rss drops below -97 dBm.

// To see this effect, try running:

//

// ./waf --run "wifi-simple-adhoc --rss=-97 --numPackets=20"

// ./waf --run "wifi-simple-adhoc --rss=-98 --numPackets=20"

// ./waf --run "wifi-simple-adhoc --rss=-99 --numPackets=20"

//

// Note that all ns-3 attributes (not just the ones exposed in the below

// script) can be changed at command line; see the documentation.

//

// This script can also be helpful to put the Wifi layer into verbose

// logging mode; this command will turn on all wifi logging:

//

// ./waf --run "wifi-simple-adhoc --verbose=1"

//

// When you are done, you will notice two pcap trace files in your directory.

// If you have tcpdump installed, you can try this:

//

// tcpdump -r wifi-simple-adhoc-0-0.pcap -nn -tt

//

#include "ns3/command-line.h"

#include "ns3/config.h"

#include "ns3/double.h"

#include "ns3/string.h"

#include "ns3/log.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/mobility-helper.h"

#include "ns3/ipv4-address-helper.h"

#include "ns3/yans-wifi-channel.h"

#include "ns3/mobility-model.h"

#include "ns3/internet-stack-helper.h"

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/traffic-control-module.h"

#include "ns3/flow-monitor-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("WifiSimpleAdhoc");

void ReceivePacket (Ptr<Socket> socket)

{

  while (socket->Recv ())

    {

      NS\_LOG\_UNCOND ("Received one packet!");

    }

}

static void GenerateTraffic (Ptr<Socket> socket, uint32\_t pktSize,

                             uint32\_t pktCount, Time pktInterval )

{

  if (pktCount > 0)

    {

      socket->Send (Create<Packet> (pktSize));

      Simulator::Schedule (pktInterval, &GenerateTraffic,

                           socket, pktSize,pktCount - 1, pktInterval);

    }

  else

    {

      socket->Close ();

    }

}

int main (int argc, char \*argv[])

{

  std::string phyMode ("DsssRate1Mbps");

  double rss = -80;  // -dBm

  uint32\_t packetSize = 1000; // bytes

  uint32\_t numPackets = 1;

  double interval = 1.0; // seconds

  bool verbose = false;

  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);

  cmd.AddValue ("rss", "received signal strength", rss);

  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);

  cmd.AddValue ("numPackets", "number of packets generated", numPackets);

  cmd.AddValue ("interval", "interval (seconds) between packets", interval);

  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);

  cmd.Parse (argc, argv);

  // Convert to time object

  Time interPacketInterval = Seconds (interval);

  // Fix non-unicast data rate to be the same as that of unicast

  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",

                      StringValue (phyMode));

  NodeContainer c;

  c.Create (4);

  // The below set of helpers will help us to put together the wifi NICs we want

  WifiHelper wifi;

  if (verbose)

    {

      wifi.EnableLogComponents ();  // Turn on all Wifi logging

    }

  wifi.SetStandard (WIFI\_PHY\_STANDARD\_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();

  // This is one parameter that matters when using FixedRssLossModel

  // set it to zero; otherwise, gain will be added

  wifiPhy.Set ("RxGain", DoubleValue (0) );

  // ns-3 supports RadioTap and Prism tracing extensions for 802.11b

  wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT\_IEEE802\_11\_RADIO);

  YansWifiChannelHelper wifiChannel;

  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

  // The below FixedRssLossModel will cause the rss to be fixed regardless

  // of the distance between the two stations, and the transmit power

  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue (rss));

  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control

  WifiMacHelper wifiMac;

  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",

                                "DataMode",StringValue (phyMode),

                                "ControlMode",StringValue (phyMode));

  // Set it to adhoc mode

  wifiMac.SetType ("ns3::AdhocWifiMac");

  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

  // Note that with FixedRssLossModel, the positions below are not

  // used for received signal strength.

  MobilityHelper mobility;

  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();

  positionAlloc->Add (Vector (0.0, 0.0, 0.0));

  positionAlloc->Add (Vector (5.0, 0.0, 0.0));

 positionAlloc->Add (Vector (10.0, 5.0, 0.0));

  positionAlloc->Add (Vector (15.0, 5.0, 0.0));

  mobility.SetPositionAllocator (positionAlloc);

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

  mobility.Install (c);

  InternetStackHelper internet;

  internet.Install (c);

  Ipv4AddressHelper ipv4;

  NS\_LOG\_INFO ("Assign IP Addresses.");

  ipv4.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer i = ipv4.Assign (devices);

  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");

  Ptr<Socket> recvSink = Socket::CreateSocket (c.Get (0), tid);

  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);

  recvSink->Bind (local);

  recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

  Ptr<Socket> source = Socket::CreateSocket (c.Get (1), tid);

  InetSocketAddress remote = InetSocketAddress (Ipv4Address ("255.255.255.255"), 80);

  source->SetAllowBroadcast (true);

  source->Connect (remote);

  // Tracing

  wifiPhy.EnablePcap ("wifi-simple-adhoc", devices);

  // Output what we are doing

  NS\_LOG\_UNCOND ("Testing " << numPackets  << " packets sent with receiver rss " << rss );

  Simulator::ScheduleWithContext (source->GetNode ()->GetId (),

                                  Seconds (1.0), &GenerateTraffic,

                                  source, packetSize, numPackets, interPacketInterval);

//Flow

  uint16\_t port = 7;

  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

  PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", localAddress);

  ApplicationContainer sinkApp = packetSinkHelper.Install (c.Get (0));

  sinkApp.Start (Seconds (0.0));

  sinkApp.Stop (Seconds (10 + 0.1));

  uint32\_t payloadSize = 1448;

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff ("ns3::UdpSocketFactory", Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (i.GetAddress (0), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (c.Get (1)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (10 + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (10 + 5));

  Simulator::Run ();

  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  std::cout << std::endl << "\*\*\* Flow monitor statistics \*\*\*" << std::endl;

  std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets

            << " / " << stats[1].txBytes << std::endl;

  std::cout << "  Offered Load: " << stats[1].txBytes \* 8.0 / (stats[1].timeLastTxPacket.GetSeconds () - stats[1].timeFirstTxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets

            << " / " << stats[1].rxBytes << std::endl;

  uint32\_t packetsDroppedByQueueDisc = 0;

  uint64\_t bytesDroppedByQueueDisc = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE\_DISC)

    {

      packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

      bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

    }

  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc

            << " / " << bytesDroppedByQueueDisc << std::endl;

  Simulator::Run ();

  Simulator::Destroy ();

  return 0;

}

5 A. Write a program for congestion control using leaky bucket algorithm.

import java.util.Scanner;

import java.lang.\*;

public class lab7 {

public static void main(String[] args)

{

int i;

int a[]=new int[20];

int buck\_rem=0,buck\_cap=4,rate=3,sent,recv;

Scanner in = new Scanner(System.in);

System.out.println("Enter the number of packets");

int n = in.nextInt();

System.out.println("Enter the packets");

for(i=1;i<=n;i++)

a[i]= in.nextInt();

System.out.println("Clock \t packet size \t accept \t sent \t remaining");

for(i=1;i<=n;i++)

{

if(a[i]!=0)

{

if(buck\_rem+a[i]>buck\_cap)

recv=-1;

else

{

recv=a[i];

buck\_rem+=a[i];

}

}

else

recv=0;

if(buck\_rem!=0)

{

if(buck\_rem<rate)

{   sent=buck\_rem;

buck\_rem=0;

}

else

{

sent=rate;

buck\_rem=buck\_rem-rate;

}

}

else

sent=0;

if(recv==-1)

System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" +  sent +"\t" +buck\_rem);

else

System.out.println(+i+ "\t\t" +a[i] +"\t\t" +recv +"\t" +sent + "\t" +buck\_rem);

}

}

}

B. Design and simulate a wired network with duplex links between 4 nodes with CDR over UDP. Set the queue size vary the bandwidth and find the number of packets dropped.

=> same as 1st one with 4 nodes

/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

 \* Copyright (c) 2015 Universita' degli Studi di Napoli "Federico II"

 \*

 \* 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: Pasquale Imputato <p.imputato@gmail.com>

 \* Author: Stefano Avallone <stefano.avallone@unina.it>

 \*/

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/traffic-control-module.h"

#include "ns3/flow-monitor-module.h"

// This simple example shows how to use TrafficControlHelper to install a

// QueueDisc on a device.

//

// The default QueueDisc is a pfifo\_fast with a capacity of 1000 packets (as in

// Linux). However, in this example, we install a RedQueueDisc with a capacity

// of 10000 packets.

//

// Network topology

//

//       10.1.1.0

// n0 -------------- n1

//    point-to-point

//

// The output will consist of all the traced changes in the length of the RED

// internal queue and in the length of the netdevice queue:

//

//    DevicePacketsInQueue 0 to 1

//    TcPacketsInQueue 7 to 8

//    TcPacketsInQueue 8 to 9

//    DevicePacketsInQueue 1 to 0

//    TcPacketsInQueue 9 to 8

//

// plus some statistics collected at the network layer (by the flow monitor)

// and the application layer. Finally, the number of packets dropped by the

// queuing discipline, the number of packets dropped by the netdevice and

// the number of packets requeued by the queuing discipline are reported.

//

// If the size of the DropTail queue of the netdevice were increased from 1

// to a large number (e.g. 1000), one would observe that the number of dropped

// packets goes to zero, but the latency grows in an uncontrolled manner. This

// is the so-called bufferbloat problem, and illustrates the importance of

// having a small device queue, so that the standing queues build in the traffic

// control layer where they can be managed by advanced queue discs rather than

// in the device layer.

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("TrafficControlExample");

int

main (int argc, char \*argv[])

{

  double simulationTime = 10; //seconds

  std::string transportProt = "Udp";

  std::string socketType;

  CommandLine cmd;

  cmd.AddValue ("transportProt", "Transport protocol to use: Tcp, Udp", transportProt);

  cmd.Parse (argc, argv);

  if (transportProt.compare ("Tcp") == 0)

    {

      socketType = "ns3::TcpSocketFactory";

    }

  else

    {

      socketType = "ns3::UdpSocketFactory";

    }

  NodeContainer nodes;

  nodes.Create (4);

  PointToPointHelper pointToPoint;

  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));

  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  pointToPoint.SetQueue ("ns3::DropTailQueue", "MaxSize", StringValue ("1p"));

  NetDeviceContainer devices01;

  devices01 = pointToPoint.Install (nodes.Get(0),nodes.Get(1));

 NetDeviceContainer devices12;

  devices12 = pointToPoint.Install (nodes.Get(1),nodes.Get(2));

 NetDeviceContainer devices23;

  devices23 = pointToPoint.Install (nodes.Get(2),nodes.Get(3));

  InternetStackHelper stack;

  stack.Install (nodes);

  Ipv4AddressHelper address;

  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces01 = address.Assign (devices01);

address.SetBase ("10.1.2.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces12 = address.Assign (devices12);

address.SetBase ("10.1.3.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces23 = address.Assign (devices23);

  //Flow

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  uint16\_t port = 7;

  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

  PacketSinkHelper packetSinkHelper (socketType, localAddress);

  ApplicationContainer sinkApp = packetSinkHelper.Install (nodes.Get (3));

  sinkApp.Start (Seconds (0.0));

  sinkApp.Stop (Seconds (simulationTime + 0.1));

  uint32\_t payloadSize = 1448;

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff (socketType, Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (interfaces12.GetAddress (1), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (nodes.Get (0)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (simulationTime + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (simulationTime + 5));

  Simulator::Run ();

  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  std::cout << std::endl << "\*\*\* Flow monitor statistics \*\*\*" << std::endl;

  std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets

            << " / " << stats[1].txBytes << std::endl;

  std::cout << "  Offered Load: " << stats[1].txBytes \* 8.0 / (stats[1].timeLastTxPacket.GetSeconds () - stats[1].timeFirstTxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets

            << " / " << stats[1].rxBytes << std::endl;

  uint32\_t packetsDroppedByQueueDisc = 0;

  uint64\_t bytesDroppedByQueueDisc = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE\_DISC)

    {

      packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

      bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

    }

  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc

            << " / " << bytesDroppedByQueueDisc << std::endl;

  Simulator::Destroy ();

  return 0;

}

or

OR

/\* -\*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

 \* Copyright (c) 2009 The Boeing Company

 \*

 \* 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

 \*

 \*/

// This script configures two nodes on an 802.11b physical layer, with

// 802.11b NICs in adhoc mode, and by default, sends one packet of 1000

// (application) bytes to the other node.  The physical layer is configured

// to receive at a fixed RSS (regardless of the distance and transmit

// power); therefore, changing position of the nodes has no effect.

//

// There are a number of command-line options available to control

// the default behavior.  The list of available command-line options

// can be listed with the following command:

// ./waf --run "wifi-simple-adhoc --help"

//

// For instance, for this configuration, the physical layer will

// stop successfully receiving packets when rss drops below -97 dBm.

// To see this effect, try running:

//

// ./waf --run "wifi-simple-adhoc --rss=-97 --numPackets=20"

// ./waf --run "wifi-simple-adhoc --rss=-98 --numPackets=20"

// ./waf --run "wifi-simple-adhoc --rss=-99 --numPackets=20"

//

// Note that all ns-3 attributes (not just the ones exposed in the below

// script) can be changed at command line; see the documentation.

//

// This script can also be helpful to put the Wifi layer into verbose

// logging mode; this command will turn on all wifi logging:

//

// ./waf --run "wifi-simple-adhoc --verbose=1"

//

// When you are done, you will notice two pcap trace files in your directory.

// If you have tcpdump installed, you can try this:

//

// tcpdump -r wifi-simple-adhoc-0-0.pcap -nn -tt

//

#include "ns3/command-line.h"

#include "ns3/config.h"

#include "ns3/double.h"

#include "ns3/string.h"

#include "ns3/log.h"

#include "ns3/yans-wifi-helper.h"

#include "ns3/mobility-helper.h"

#include "ns3/ipv4-address-helper.h"

#include "ns3/yans-wifi-channel.h"

#include "ns3/mobility-model.h"

#include "ns3/internet-stack-helper.h"

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/traffic-control-module.h"

#include "ns3/flow-monitor-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("WifiSimpleAdhoc");

void ReceivePacket (Ptr<Socket> socket)

{

  while (socket->Recv ())

    {

      NS\_LOG\_UNCOND ("Received one packet!");

    }

}

static void GenerateTraffic (Ptr<Socket> socket, uint32\_t pktSize,

                             uint32\_t pktCount, Time pktInterval )

{

  if (pktCount > 0)

    {

      socket->Send (Create<Packet> (pktSize));

      Simulator::Schedule (pktInterval, &GenerateTraffic,

                           socket, pktSize,pktCount - 1, pktInterval);

    }

  else

    {

      socket->Close ();

    }

}

int main (int argc, char \*argv[])

{

  std::string phyMode ("DsssRate1Mbps");

  double rss = -80;  // -dBm

  uint32\_t packetSize = 1000; // bytes

  uint32\_t numPackets = 1;

  double interval = 1.0; // seconds

  bool verbose = false;

  CommandLine cmd;

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);

  cmd.AddValue ("rss", "received signal strength", rss);

  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);

  cmd.AddValue ("numPackets", "number of packets generated", numPackets);

  cmd.AddValue ("interval", "interval (seconds) between packets", interval);

  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbose);

  cmd.Parse (argc, argv);

  // Convert to time object

  Time interPacketInterval = Seconds (interval);

  // Fix non-unicast data rate to be the same as that of unicast

  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",

                      StringValue (phyMode));

  NodeContainer c;

  c.Create (4);

  // The below set of helpers will help us to put together the wifi NICs we want

  WifiHelper wifi;

  if (verbose)

    {

      wifi.EnableLogComponents ();  // Turn on all Wifi logging

    }

  wifi.SetStandard (WIFI\_PHY\_STANDARD\_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();

  // This is one parameter that matters when using FixedRssLossModel

  // set it to zero; otherwise, gain will be added

  wifiPhy.Set ("RxGain", DoubleValue (0) );

  // ns-3 supports RadioTap and Prism tracing extensions for 802.11b

  wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT\_IEEE802\_11\_RADIO);

  YansWifiChannelHelper wifiChannel;

  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

  // The below FixedRssLossModel will cause the rss to be fixed regardless

  // of the distance between the two stations, and the transmit power

  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue (rss));

  wifiPhy.SetChannel (wifiChannel.Create ());

  // Add a mac and disable rate control

  WifiMacHelper wifiMac;

  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",

                                "DataMode",StringValue (phyMode),

                                "ControlMode",StringValue (phyMode));

  // Set it to adhoc mode

  wifiMac.SetType ("ns3::AdhocWifiMac");

  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

  // Note that with FixedRssLossModel, the positions below are not

  // used for received signal strength.

  MobilityHelper mobility;

  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();

  positionAlloc->Add (Vector (0.0, 0.0, 0.0));

  positionAlloc->Add (Vector (5.0, 0.0, 0.0));

  positionAlloc->Add (Vector (15.0, 5.0, 0.0));

  positionAlloc->Add (Vector (20.0, 5.0, 0.0));

  mobility.SetPositionAllocator (positionAlloc);

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

  mobility.Install (c);

  InternetStackHelper internet;

  internet.Install (c);

  Ipv4AddressHelper ipv4;

  NS\_LOG\_INFO ("Assign IP Addresses.");

  ipv4.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer i = ipv4.Assign (devices);

  uint16\_t port = 7;

  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

  PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", localAddress);

  ApplicationContainer sinkApp = packetSinkHelper.Install (c.Get (0));

  sinkApp.Start (Seconds (0.0));

  sinkApp.Stop (Seconds (10 + 0.1));

  uint32\_t payloadSize = 1448;

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  OnOffHelper onoff ("ns3::UdpSocketFactory", Ipv4Address::GetAny ());

  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

  onoff.SetAttribute ("PacketSize", UintegerValue (payloadSize));

  onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

  ApplicationContainer apps;

  InetSocketAddress rmt (i.GetAddress (0), port);

  rmt.SetTos (0xb8);

  AddressValue remoteAddress (rmt);

  onoff.SetAttribute ("Remote", remoteAddress);

  apps.Add (onoff.Install (c.Get (1)));

  apps.Start (Seconds (1.0));

  apps.Stop (Seconds (10 + 0.1));

  FlowMonitorHelper flowmon;

  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop (Seconds (10 + 5));

  Simulator::Run ();

  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  std::cout << std::endl << "\*\*\* Flow monitor statistics \*\*\*" << std::endl;

  std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets

            << " / " << stats[1].txBytes << std::endl;

  std::cout << "  Offered Load: " << stats[1].txBytes \* 8.0 / (stats[1].timeLastTxPacket.GetSeconds () - stats[1].timeFirstTxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets

            << " / " << stats[1].rxBytes << std::endl;

  uint32\_t packetsDroppedByQueueDisc = 0;

  uint64\_t bytesDroppedByQueueDisc = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE\_DISC)

    {

      packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

      bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE\_DISC];

    }

  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc

            << " / " << bytesDroppedByQueueDisc << std::endl;

  uint32\_t packetsDroppedByNetDevice = 0;

  uint64\_t bytesDroppedByNetDevice = 0;

  if (stats[1].packetsDropped.size () > Ipv4FlowProbe::DROP\_QUEUE)

    {

      packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP\_QUEUE];

      bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP\_QUEUE];

    }

  std::cout << "  Packets/Bytes Dropped by NetDevice:   " << packetsDroppedByNetDevice

            << " / " << bytesDroppedByNetDevice << std::endl;

  std::cout << "  Throughput: " << stats[1].rxBytes \* 8.0 / (stats[1].timeLastRxPacket.GetSeconds () - stats[1].timeFirstRxPacket.GetSeconds ()) / 1000000 << " Mbps" << std::endl;

  std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds () / stats[1].rxPackets << std::endl;

  std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds () / (stats[1].rxPackets - 1) << std::endl;

  auto dscpVec = classifier->GetDscpCounts (1);

  for (auto p : dscpVec)

    {

      std::cout << "  DSCP value:   0x" << std::hex << static\_cast<uint32\_t> (p.first) << std::dec

                << "  count:   "<< p.second << std::endl;

    }

  Simulator::Destroy ();

}

6 A. Write a client-server program using TCP/IP sockets in which client requests for a file by sending the file name to the server, and the server sends back the contents of the requested file if present.

Server.java :

import java.net.\*;

import java.io.\*;

public class TCPS

{

public static void main(String[] args) throws Exception

{

ServerSocket sersock=new ServerSocket(4000);

System.out.println("Server ready for connection");

Socket sock=sersock.accept();

System.out.println("Connection Is successful and waiting for chatting");

InputStream istream=sock.getInputStream();

BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));

String fname=fileRead.readLine();

BufferedReader ContentRead=new BufferedReader(new FileReader(fname));

OutputStream ostream=sock.getOutputStream();

PrintWriter pwrite=new PrintWriter(ostream,true);

String str;

while((str=ContentRead.readLine())!=null){

pwrite.println(str);

}

sock.close();

sersock.close();

pwrite.close();

fileRead.close();

ContentRead.close();

}

}

Client.java:

import java.net.\*;

import java.io.\*;

public class TCPC

{

public static void main(String[] args) throws Exception

{

Socket sock=new Socket("127.0.01",4000);

System.out.println("Enter the filename");

BufferedReader keyRead=new BufferedReader(new InputStreamReader(System.in));

String fname=keyRead.readLine();

OutputStream ostream=sock.getOutputStream();

PrintWriter pwrite=new PrintWriter(ostream,true);

pwrite.println(fname);

InputStream istream=sock.getInputStream();

BufferedReader socketRead=new BufferedReader(new InputStreamReader(istream));

String str;

while((str=socketRead.readLine())!=null)

{

System.out.println(str);

}

pwrite.close();

socketRead.close();

keyRead.close();

}

}

B. Design a wired network with ‘n’ nodes to observe the performance of two TCP variants (where one is Reno). Simulate the designed network and observe the network performance.

=> same as 2nd one

7 A. Trace Domain Name Server using packet sniffer and packet analyser.

B. Design and simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of packets.

=> same as 3rd one

8 A. Trace Internet Protocol and Internet Control Message Protocol using packet sniffer and packet analyser.

B. Design and simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of packets.

=> same as 3rd one

9 A. Trace DHCP using packet sniffer and packet analyser.

B. Design and simulate a wired network with duplex links between 3 nodes with CDR over UDP. Set the queue size vary the bandwidth and find the number of packets dropped.

=> same as 1st one

10 A Trace Hypertext Transfer Protocol using packet sniffer and packet analyser.

B Design and simulate infrastructure less network, generate two traffic flows between nodes and analyse its performance.

=> same as 4th one