Ruchita Jain
Jeevitha Mahankali
Raghava Adarsh Mandarapu

DCN2-Project

```
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-layout-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-apps-module.h"

using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Group11");
int
main (int argc, char *argv[])
{
/* The below value configures the default behavior of global routing. By default, it is disabled.  To
respond to interface events, set to true*/

  Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents", BooleanValue (true));

  /*Allow the user to override any of the defaults and the above Bind ()s at run-time, via command-
line arguments*/
  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

   /*Creating network NodeContainer for nodes, routers, wifi Nodes and csma nodes*/
    uint16_t ncsma = 15;
    NodeContainer routers;
    NodeContainer nodes;
    NodeContainer wifiStaNodes;
    NodeContainer wifiApNode;
    NodeContainer csmanodes1;
    csmanodes1.Create(ncsma);
    routers.Create(3);
    nodes.Create(10);
```

```
    wifiStaNodes.Create(14);
    wifiApNode.Create(1);

   /*Creating Internet Stacks*/
    InternetStackHelper stack;
    stack.Install(routers);
    stack.Install(nodes);
    stack.Install(wifiStaNodes);
    stack.Install(csmanodes1);
    stack.Install(wifiApNode);

    /*Create Csma helper*/
    CsmaHelper csma;

    /*Create Point-to-Point helper*/
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
    p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

    /*Create an Address helper*/
    Ipv4AddressHelper address;

    /* Link between Router0 and Router1*/
    NodeContainer subnet8;                    // Comment this to observe Link Failure
    subnet8.Add(routers.Get(0));
    subnet8.Add(routers.Get(1));

    /*Create Device container to hold net devices installed on each node*/
    NetDeviceContainer subnet8Devices = p2p.Install(subnet8);

    /*Configure the subnet address and mask*/
    address.SetBase("10.172.1.0","255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet8Interfaces = address.Assign(subnet8Devices);

    /* Link between Router0 and Router 2*/
     NodeContainer subnet7;
    subnet7.Add(routers.Get(0));
    subnet7.Add(routers.Get(2));

    /*Create Device container to hold net devices installed on each node*/
    NetDeviceContainer subnet7Devices = p2p.Install(subnet7);

    /*Configure the subnet address and mask*/
    address.SetBase("10.172.2.0","255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet7Interfaces = address.Assign(subnet7Devices);
```

```
    /* Link between Router1 and Router2*/
    NodeContainer subnet9;
    subnet9.Add(routers.Get(1));
    subnet9.Add(routers.Get(2));

    /*Create Device container to hold net devices installed on each node*/
    NetDeviceContainer subnet9Devices = p2p.Install(subnet9);

    /*Configure the subnet address and mask*/
    address.SetBase("10.172.3.0","255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet9Interfaces = address.Assign(subnet9Devices);

    /*Creating star topology*/
     uint32_t nspokes = 15;
    CommandLine cmd;
    cmd.AddValue ("nSpokes", "Number of nodes to place in the star", nspokes);
    cmd.Parse (argc, argv);

    NS_LOG_INFO ("Creating star topology");
    PointToPointHelper pointToPoint;
    pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
    pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
    PointToPointStarHelper star (nspokes, pointToPoint);

    InternetStackHelper internet;
    star.InstallStack (internet);

    star.AssignIpv4Addresses (Ipv4AddressHelper ("192.168.1.0", "255.255.255.0"));

    /*Configuring Subnets*/
   NodeContainer subnet2;
    subnet2.Add(star.GetHub());
    subnet2.Add(routers.Get(0));

    /*Create a device conatiner to hold net devices installed on each node.*/
    NetDeviceContainer subnet2Devices = p2p.Install(subnet2);

    /*Configure the subnet address and mask*/
    address.SetBase("10.1.1.0", "255.255.255.0");

    /*Create an interface container to hold the ipv4 interface create and assign IP addresses to
interface*/
    Ipv4InterfaceContainer subnet2Interfaces = address.Assign(subnet2Devices);
```

```
  /*Configuring CSMA*/
     NodeContainer subnet3;
    subnet3.Add(routers.Get(1));
    subnet3.Add(nodes.Get(0));

    /*Create Device container to hold net devices installed on each node*/
    NetDeviceContainer subnet3Devices = csma.Install(subnet3);

    /*Configure the subnet address and mask*/
    address.SetBase("192.16.2.0", "255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet3Interfaces = address.Assign(subnet3Devices);

    /*creating subnets*/
    NodeContainer subnet4devices;
    for(int i=1; i<15; i++){
       subnet4devices.Add(csmanodes1.Get(i));
    }
    NodeContainer subnet4;
    subnet4.Add(subnet4devices);
    subnet4.Add(nodes.Get(0));

    /*Create Device container to hold net devices installed on each node*/
    NetDeviceContainer subnet4Devices = csma.Install(subnet4);

    /*Configure the subnet address and mask*/
    address.SetBase("10.3.1.0", "255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet4Interfaces = address.Assign(subnet4Devices);

    /*Creating Wifi nodes*/
       NodeContainer subnet5;
       subnet5.Add(routers.Get(2));
       subnet5.Add(wifiApNode.Get(0));

    /*Create a device conatiner to hold net devices installed on each node.*/
    NetDeviceContainer subnet5Devices = p2p.Install(subnet5);

    /*Configure the subnet address and mask*/
    address.SetBase("10.4.1.0", "255.255.255.0");

    /*Create a interface container to hold ipv4 interfaces created and assign IP address to each
interface*/
    Ipv4InterfaceContainer subnet5Interfaces = address.Assign(subnet5Devices);
```

```
    NodeContainer wifisubnet6;
    wifisubnet6.Add(wifiStaNodes);
    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);
    address.SetBase ("172.16.1.0", "255.255.255.0");
    address.Assign (staDevices);
    address.Assign (apDevices);

/*UDP Echo Client-Server Application*/

/*Between LAN1 and LAN3*/
   UdpEchoServerHelper echoServer1(22);
    ApplicationContainer serverApps1 = echoServer1.Install (star.GetSpokeNode(5));
    serverApps1.Start (Seconds (2.0));
    serverApps1.Stop (Seconds (10.0));
```

```
    UdpEchoClientHelper echoClient1 (star.GetSpokeIpv4Address(5), 22);
    echoClient1.SetAttribute ("MaxPackets", UintegerValue (1));
    echoClient1.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
    echoClient1.SetAttribute ("PacketSize", UintegerValue (1024));

    ApplicationContainer clientApps1 = echoClient1.Install (wifiStaNodes.Get(8));
    clientApps1.Start (Seconds (3.0));
    clientApps1.Stop (Seconds (10.0));

/*UDP Echo Client-Server Application 2*/
/*Between LAN1 and LAN2*/

    UdpEchoServerHelper echoServer (9);
    ApplicationContainer serverApps = echoServer.Install (star.GetSpokeNode(1));
    serverApps.Start (Seconds (5.0));
    serverApps.Stop (Seconds (10.0));


    UdpEchoClientHelper echoClient (star.GetSpokeIpv4Address(1), 9);
    echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
    echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
    echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

    ApplicationContainer clientApps = echoClient.Install (csmanodes1.Get(6));
    clientApps.Start (Seconds (6.0));
    clientApps.Stop (Seconds (10.0));




/*ON OFF application*/

/*Create a packet sink on the star node 3 to receive packets.*/
   uint16_t port = 50000;
    Address LocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
    PacketSinkHelper packetSinkHelper1 ("ns3::TcpSocketFactory", LocalAddress);
    ApplicationContainer App = packetSinkHelper1.Install (star.GetHub());
    App.Start (Seconds (2.0));
    App.Stop (Seconds (4.0));

    //
    // Create OnOff applications to send TCP to the node 3, from spoke node 7.
    //
    OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
    onOffHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
    onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
```

```
    ApplicationContainer spokeApps;

    AddressValue remoteAddress1 (InetSocketAddress (star.GetHubIpv4Address (7), port));
    onOffHelper.SetAttribute ("Remote", remoteAddress1);
    spokeApps.Add (onOffHelper.Install (star.GetSpokeNode(7)));

    spokeApps.Start (Seconds (2.0));
    spokeApps.Stop (Seconds (4.0));

/*ON OFF application*/

/* Create a packet sink on the star node 10 to receive packets.*/
     uint16_t port1 = 12;
    Address LocalAddress1 (InetSocketAddress (Ipv4Address::GetAny (), port));
    PacketSinkHelper packetSinkHelper2 ("ns3::TcpSocketFactory", LocalAddress1);
    ApplicationContainer App1 = packetSinkHelper2.Install (star.GetHub());
    App1.Start (Seconds (7.0));
    App1.Stop (Seconds (9.0));

    /* Create OnOff applications to send TCP to the node 5, from spoke node 10.*/
    OnOffHelper onOffHelper1 ("ns3::TcpSocketFactory", Address ());
    onOffHelper1.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
    onOffHelper1.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
   ApplicationContainer spokeApps1;
    AddressValue remoteAddress2 (InetSocketAddress (star.GetHubIpv4Address (10), port1));
    onOffHelper1.SetAttribute ("Remote", remoteAddress1);
    spokeApps1.Add (onOffHelper1.Install (star.GetSpokeNode(10)));
    spokeApps1.Start (Seconds (2.0));
    spokeApps1.Stop (Seconds (4.0));

/*Broadcast*/
       uint16_t port2 = 20;
      NS_LOG_INFO ("Create Applications.");
     OnOffHelper onoff3 ("ns3::UdpSocketFactory", Address (InetSocketAddress (Ipv4Address
           ("255.255.255.255"), port2)));
     onoff3.SetConstantRate (DataRate ("500kb/s"));
     ApplicationContainer app3 = onoff3.Install (csmanodes1.Get(0));
     app3.Start (Seconds (4.0));
     app3.Stop (Seconds (6.0));

     /*Creating optional sink packet*/

     PacketSinkHelper sink ("ns3::UdpSocketFactory",Address (InetSocketAddress
(Ipv4Address::GetAny (), port2)));
     app3=sink.Install (wifiStaNodes.Get(0));

     for(int i=1; i<15; i++){
        app3.Add(sink.Install (csmanodes1.Get(i)));
     }
```

```
      app3.Start (Seconds (4.0));
     app3.Stop (Seconds (6.0));

/*Creating Routing table*/
   NS_LOG_INFO ("Enable static global routing.");
   /*Turn on global static routing so we can actually be routed across the star */
   Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
   Simulator::Stop (Seconds (20.0));

   /* Trace routing tables */
   Ipv4GlobalRoutingHelper g;
   Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("dynamic-global-
routing.routes", std::ios::out);
   g.PrintRoutingTableAllAt (Seconds (6), routingStream);

/*NetAnim*/
   AnimationInterface anim ("dummy.xml");
   star.BoundingBox (5, 5, 10, 10);
   anim.SetConstantPosition(csmanodes1.Get(0), 0.0, 30.0);
   anim.SetConstantPosition(wifiApNode.Get(0), 40.0, 40.0);
   anim.SetConstantPosition(star.GetHub(), 500.0, 0.0);
   anim.SetConstantPosition(routers.Get(0), 30.0, 40.0 );
   anim.SetConstantPosition(routers.Get(1), 40.0, 50.0 );
   anim.SetConstantPosition(routers.Get(2), 50.0, 60.0 );
    for(int i=7; i>=0; i--){
      anim.SetConstantPosition(csmanodes1.Get(i), (7-i)*2, 12.0);
   }


   for(int i=14; i>7; i--){
      anim.SetConstantPosition(csmanodes1.Get(i), (14-i)*2, 13.0);
   }

   for(int i=0; i<14; i++){
      anim.SetConstantPosition(wifiStaNodes.Get(i), (120-i)*2, (120-i)*2);
   }
   /*Throughput*/
   FlowMonitorHelper flowmon;
   Ptr<FlowMonitor> monitor = flowmon.InstallAll();

   /* Calculate Throughput using Flowmonitor*/
  Simulator::Stop (Seconds (100.0));
   NS_LOG_INFO ("Run Simulation.");
   Simulator::Run ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier
());
 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
 for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end ();
++i)
   {
     Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
```

```
//std::cout << "Flow " << i->first  << " (" << t.sourceAddress << " -> " << t.destinationAddress <<
")\n";
     if ((/*t.sourceAddress=="10.1.1.0" && */t.destinationAddress == "192.168.6.2"))
     {
        std::cout  << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";
        std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
        std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
          std::cout << "  THROUGHPUT=: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024/1024
<< " Mbps\n";
     }
      if ((/*t.sourceAddress=="10.1.1.0" && */t.destinationAddress == "192.168.2.2"))
     {
        std::cout  << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";
        std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
        std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
          std::cout << "  THROUGHPUT=: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024/1024
<< " Mbps\n";
     }

  }

  //std::cout << "The ip of alternate Router : "  << subnet7Interfaces.GetAddress(0) << std::endl;
//uncomment for link failure
  std::cout << "The ip of Router : "  << subnet8Interfaces.GetAddress (0) << std::endl;
//comment for link failure
   monitor->SerializeToXmlFile("lab-1.flowmon", true, true);
     Simulator::Destroy ();
    NS_LOG_INFO ("Done.");
     Simulator::Stop (Seconds (10.0));
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
}
```