NS31: * Problem statement: Three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

Expected learning outcome: NS3 basic simulation basics, on-off application (CBR), reading traces through flow monitor and display the network performance

Algorithm:

1. Create a simple 3 node topology using NodeContainer topology helper as no---n1---n2---n3. Use point to point links between two nodes.
   // Network topology
   //
   // 10.1.1.0 10.1.2.0
   // n0 ---------- n1----------n2
   // point-to-point point-to-point
   //
2. Install internet stack on all nodes.
3. Assign IP4 addresses to netdevice containing two nodes at a time
4. Set network address for interfaces
5. Populate the global routing table between all nodes
6. Install a UDP socket instance on Node0 as sender that will connect to Node3 as receiver.
7. Start the UDP application at time 1.0 at rate Rate1
8. Use the ns-3 tracing mechanism to record the network performance.
   // The output will consist of all the traced statistics collected at the network layer (by the flow monitor) and the application layer.
   // Finally, the number of packets dropped by the queuing discipline and
   // the number of packets dropped by the netdevice
9. vary the bandwidth of point-to-point link and observe the performance
10. Use gnuplot/matplotlib to visualise plots of bandwidth vs packet drop.
11. Conclude the performance from graph
12. Perform the above experiment for different topology conncetion.

Steps:

1. Open editor and write the program for the algorithm logic
2. Save in ns3.30/scratch directory
3. Compilation (you should be accessing ns-3.35 directory):
   $./waf --run scratch/filenameWithoutExtention

*/

```
#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;

int main ()
{
```

```
double simulationTime = 10; //seconds
std::string socketType="ns3::UdpSocketFactory";//"ns3::TcpSocketFactory";

NodeContainer nodes;
nodes.Create (3);

PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
p2p.SetQueue ("ns3::DropTailQueue", "MaxSize", StringValue ("1p"));

NetDeviceContainer dev01;
dev01= p2p.Install (nodes.Get(0),nodes.Get(1));
NetDeviceContainer dev12;
dev12= p2p.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 (dev01);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces12 = address.Assign (dev12);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

//Flow
uint16_t port = 7;
Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper psh (socketType, localAddress);
ApplicationContainer sinkApp = psh.Install (nodes.Get (2));
sinkApp.Start (Seconds (0.0));
sinkApp.Stop (Seconds (simulationTime + 0.1));

OnOffHelper onoff (socketType, Ipv4Address::GetAny ());
onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s
ApplicationContainer apps;

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

  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
    {
      Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
      std::cout << "Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress<< std::endl;
      std::cout << "Tx Packets   = " << iter->second.txPackets<< std::endl;
      std::cout << "Rx Packets   = " << iter->second.rxPackets<< std::endl;
      std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;
      std::cout << "Throughput   = " << iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1000000
<< " Kbps"<< std::endl;
    }

  Simulator::Destroy ();

  return 0;
}


/*
Usage of the existing examples: (Lab1.cc--→ example/traffic-control/traffic-control.cc)

*/
```

NS32: /* Simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP agent between n1-n3. Apply relevant applications over TCP and UDP agents by changing the parameters and determine the number of packets sent by TCP/UDP */

```
#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 ("TrafficControlExample");


int
main (int argc, char *argv[])
{
 double simulationTime = 10; //seconds
 std::string transportProt = "Udp";
 std::string socketType;

 CommandLine cmd;
 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 ("5Mbps"));
 pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
//pointToPoint.SetQueue ("ns3::DropTailQueue", "Mode", StringValue ("QUEUE_MODE_PACKETS"),
"MaxPackets", UintegerValue (1));

 NetDeviceContainer devices1 = pointToPoint.Install (nodes.Get(0),nodes.Get(2));
  NetDeviceContainer devices2 = pointToPoint.Install (nodes.Get(1),nodes.Get(2));
  NetDeviceContainer devices3 = pointToPoint.Install (nodes.Get(2),nodes.Get(3));
```

```
 InternetStackHelper stack;
 stack.Install (nodes);


 Ipv4AddressHelper address1;
 address1.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer interfaces1 = address1.Assign (devices1);
Ipv4AddressHelper address2;
 address2.SetBase ("10.1.2.0", "255.255.255.0");
 Ipv4InterfaceContainer interfaces2 = address2.Assign (devices2);
Ipv4AddressHelper address3;
 address3.SetBase ("10.1.3.0", "255.255.255.0");
 Ipv4InterfaceContainer interfaces3 = address3.Assign (devices3);

Ipv4GlobalRoutingHelper::PopulateRoutingTables (); //take from examples/tutorial/third.cc

 //UDP Flow
 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;

 AddressValue remoteAddress (InetSocketAddress (interfaces3.GetAddress (1), port));
 onoff.SetAttribute ("Remote", remoteAddress);
 apps.Add (onoff.Install (nodes.Get (0)));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (simulationTime + 0.1));

//TCP Flow
 uint16_t port_tcp = 9;
 socketType = "ns3::TcpSocketFactory"; //Add this line
 Address localAddress_tcp (InetSocketAddress (Ipv4Address::GetAny (), port_tcp));
 PacketSinkHelper packetSinkHelper_tcp (socketType, localAddress_tcp);
 ApplicationContainer sinkApp_tcp = packetSinkHelper_tcp.Install (nodes.Get (3));

 sinkApp_tcp.Start (Seconds (0.5));
 sinkApp_tcp.Stop (Seconds (simulationTime + 0.1));
```

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

 OnOffHelper onoff_tcp (socketType, Ipv4Address::GetAny ());
 onoff_tcp.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
 onoff_tcp.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
 onoff_tcp.SetAttribute ("PacketSize", UintegerValue (payloadSize));
 onoff_tcp.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s
 ApplicationContainer apps_tcp;

 AddressValue remoteAddress_tcp (InetSocketAddress (interfaces3.GetAddress (1), port_tcp));
 onoff_tcp.SetAttribute ("Remote", remoteAddress_tcp);
 apps_tcp.Add (onoff_tcp.Install (nodes.Get (1)));
 apps_tcp.Start (Seconds (1.5));
 apps_tcp.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 << "  Dropped Packets:   " << stats[1].lostPackets << std::endl;
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
   {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
    NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress);
    NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);

   }

 Simulator::Destroy ();
 return 0;
}


/*
Usage of the existing examples: (Lab2.cc ----------> Lab1.cc--→ example/traffic-control/traffic-
control.cc)

*/
```

Ns3: /*Simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of packets. */

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"

// Default Network Topology
//
// Number of wifi or csma nodes can be increased up to 250
//                          |
//             Rank 0   |   Rank 1
// -------------------------|----------------------------
//   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[])
{
 uint32_t nCsma = 3;
 uint32_t nWifi = 3;
 double simulationTime = 10; //seconds
 std::string socketType = "ns3::UdpSocketFactory";

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

 // Check for valid number of csma or wifi nodes
 // 250 should be enough, otherwise IP addresses
 // soon become an issue
 if (nWifi > 250 || nCsma > 250)
  {
   std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::endl;
   return 1;
  }
```

```
 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;
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);

 /*  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps =
    echoClient.Install (wifiStaNodes.Get (nWifi - 1));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));
 */

 //Flow
  uint16_t port = 7;
  Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
  PacketSinkHelper packetSinkHelper (socketType, localAddress);
```

```
 ApplicationContainer sinkApp = packetSinkHelper.Install (csmaNodes.Get (nCsma));
 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;
 AddressValue remoteAddress (InetSocketAddress (csmaInterfaces.GetAddress (nCsma), port));
 onoff.SetAttribute ("Remote", remoteAddress);
 apps.Add (onoff.Install (wifiStaNodes.Get (nWifi - 1)));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (simulationTime + 0.1));

 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

 Simulator::Stop (Seconds (10.0));

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

 Simulator::Run ();
// Print per flow statistics
 monitor->CheckForLostPackets ();
 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

 for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
   {
 Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
    NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress);
    NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);
    std::cout << "Rx Packets   = " << iter->second.rxPackets<< std::endl;
    std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;
    std::cout << "Throughput   = " << iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1000000
<< " Kbps"<< std::endl;
   }
 Simulator::Destroy ();
 return 0;
}
/*
Usage of the existing examples: (Lab3.cc--→ example/tutorials/third.cc)

*/
```

NS4: //Simulate a wireless network, generate traffic and analyze its performance.
//

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/netanim-module.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace ns3;

int main (int argc, char *argv[])
{
 std::string phyMode ("DsssRate1Mbps"); //Direct Sequence Spread Spectrum (DSSS)
 double rss = -80;  // -dBm

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

 // disable fragmentation for frames below 2200 bytes
 Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue
("2200"));
 // turn off RTS/CTS for frames below 2200 bytes
 Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2200"));
 // Fix non-unicast data rate to be the same as that of unicast
 Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue (phyMode));

 NodeContainer c;
 c.Create (5);

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

 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 (1.0, 1.0, 0.0));
 positionAlloc->Add (Vector (2.0, 1.0, 0.0));
 positionAlloc->Add (Vector (3.0, 2.0, 0.0));
 positionAlloc->Add (Vector (4.0, 3.0, 0.0));
 positionAlloc->Add (Vector (5.0, 2.0, 0.0));
 mobility.SetPositionAllocator (positionAlloc);
 mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");*/
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 (c);

 InternetStackHelper internet;
 internet.Install (c);
 Ipv4AddressHelper ipv4;
 ipv4.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer i = ipv4.Assign (devices);
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
 uint16_t port = 7;
 Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
 PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", localAddress);
 ApplicationContainer sinkApp = packetSinkHelper.Install (c.Get (4));

 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;
 AddressValue remoteAddress (InetSocketAddress (i.GetAddress (3), port));
//AddressValue remoteAddress (InetSocketAddress (Ipv4Address ("255.255.255.255"), port));
 onoff.SetAttribute ("Remote", remoteAddress);
 apps.Add (onoff.Install (c.Get (0)));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (10 + 0.1));
//Enable Tracing using flowmonitor
 FlowMonitorHelper flowmon;
 Ptr<FlowMonitor> monitor = flowmon.InstallAll();
 Simulator::Stop (Seconds (10.0));
//Add visualization using Netanim
 AnimationInterface anim ("ex4.xml");

 Simulator::Run ();

// Print per flow statistics
 monitor->CheckForLostPackets ();
 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

 for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
   {
 Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
    NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress);
    NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);
    NS_LOG_UNCOND("Rx Packets = " << iter->second.rxPackets);
    NS_LOG_UNCOND("Throughput: " << iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1024  << "
Kbps");
   }

 Simulator::Destroy ();

 return 0;
}
/*
Usage of the existing examples: (Lab4.cc--→ example/wireless/wifi-simple-adhoc.cc)

*/
```

NS5: /* Simulate a transmission of ping message over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion. */

```
// Network topology
//
//      n0    n1   n2   n3    n4     n5
//      |     |    |    |     |      |
//    ===================================
//
//  node n0,n1,n3,n4,n5 pings to node n2
//  node n0 generates protocol 2 (IGMP) to node n3

#include <iostream>
#include <fstream>
#include <string>
#include <cassert>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
//#include "ns3/netanim-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("CsmaPingExample");


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

  CommandLine cmd;
  cmd.Parse (argc, argv);
  Time interPacketInterval = Seconds (1.);
  // Here, we will explicitly create four nodes.
  NS_LOG_UNCOND ("Create nodes.");
  NodeContainer c;
  c.Create (6);

  // connect all our nodes to a shared channel.
  NS_LOG_UNCOND ("Build Topology.");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
  csma.SetDeviceAttribute ("EncapsulationMode", StringValue ("Llc"));
  NetDeviceContainer devs = csma.Install (c);
```

```
 // add an ip stack to all nodes.
 NS_LOG_UNCOND ("Add ip stack.");
 InternetStackHelper ipStack;
 ipStack.Install (c);



 // assign ip addresses
 NS_LOG_UNCOND ("Assign ip addresses.");
 Ipv4AddressHelper ip;
 ip.SetBase ("192.168.1.0", "255.255.255.0");
 Ipv4InterfaceContainer addresses = ip.Assign (devs);

 NS_LOG_UNCOND ("Create Source");

 InetSocketAddress dst = InetSocketAddress (addresses.GetAddress (3));
 OnOffHelper onoff = OnOffHelper ("ns3::UdpSocketFactory", dst);
 onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
 onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
 onoff.SetAttribute ("PacketSize", UintegerValue (1100));
 onoff.SetAttribute ("DataRate", StringValue ("50Mbps"));

 ApplicationContainer apps = onoff.Install (c.Get (0));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (10.0));

 NS_LOG_UNCOND ("Create Sink.");
 PacketSinkHelper sink = PacketSinkHelper ("ns3::UdpSocketFactory", dst);
 apps = sink.Install (c.Get (3));
 apps.Start (Seconds (0.0));
 apps.Stop (Seconds (11.0));

 NS_LOG_UNCOND ("Create pinger");
 V4PingHelper ping = V4PingHelper (addresses.GetAddress (0));
// ping.SetAttribute ("Interval", TimeValue (interPacketInterval));
ping.SetAttribute ("Interval", TimeValue (interPacketInterval));
 NodeContainer pingers;
 pingers.Add (c.Get (3));
 pingers.Add (c.Get (1));
 pingers.Add (c.Get (2));
 pingers.Add (c.Get (4));
 pingers.Add (c.Get (5));
 apps = ping.Install (pingers);
 apps.Start (Seconds (2.0));
 apps.Stop (Seconds (10.0));



//Enable Tracing using flowmonitor
 FlowMonitorHelper flowmon;
 Ptr<FlowMonitor> monitor = flowmon.InstallAll();
```

```
  Simulator::Stop (Seconds (10.0));

//Add visualization using Netanim
//  AnimationInterface anim ("ex5.xml");

  NS_LOG_UNCOND ("Run Simulation.");
  Simulator::Run ();

// Print per flow statistics
  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
    {
  Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
      NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress);
     NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);
     NS_LOG_UNCOND("Rx Packets = " << iter->second.rxPackets);
     std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;
     NS_LOG_UNCOND("Throughput: " << iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1024  << "
Kbps");
    }

  Simulator::Destroy ();
  NS_LOG_UNCOND ("Done.");
}


/*
Usage of the existing examples: (Lab5.cc--→ src/csma/examples/csma-ping.cc)
*/
```