Name: Wahia Tasnim **ID:** IT-16029

Experiment N0: 03

Experiment Name : TCP and Router Queues

Objective :

1. Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.
2. Add drop tail queues of size QueueSize5 and QueueSize6 to Node5 and Node6, respectively.
3. Install a TCP socket instance on Node1 that will connect to Node3.
4. Install a TCP socket instance on Node2 that will connect to Node3.
5. Install a TCP socket instance on Node2 that will connect to Node4.
6. Start Node1--Node3 flow at time 1s, then measure it's throughput. How long does it take to fill link's entire capacity?
7. Start Node2--Node3 and Node2--Node4 flows at time 15s, measure their throughput.
8. Measure packet loss and cwnd size, and plot graphs throughput/time, cwnd/time and packet loss/time for each of the flows.
9. Plot graph throughput/cwnd and packet loss/cwnd for the first flow.

Source Code:

```
#include <fstream>

#include <iostream>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/mobility-module.h"
```

```
#include "ns3/wifi-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-module.h"

using namespace ns3;
using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("manet-routing-compare");

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

private:
```

```
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node>
node);

  void ReceivePacket (Ptr<Socket> socket);

  void CheckThroughput ();



  uint32_t port;

  uint32_t bytesTotal;

  uint32_t packetsReceived;



  std::string m_CSVfileName;

  int m_nSinks;

  std::string m_protocolName;

  double m_txp;

  bool m_traceMobility;

  uint32_t m_protocol;

};



RoutingExperiment::RoutingExperiment ()

  : port (9),

    bytesTotal (0),

    packetsReceived (0),

    m_CSVfileName ("manet-routing.output.csv"),

    m_traceMobility (false),
```

```
    m_protocol (2) // AODV

{

}



static inline std::string

PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet,
Address senderAddress)

{

  std::ostringstream oss;



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



  if (InetSocketAddress::IsMatchingType (senderAddress))

    {

      InetSocketAddress addr = InetSocketAddress::ConvertFrom
(senderAddress);

      oss << " received one packet from " << addr.GetIpv4 ();

    }

  else

    {

      oss << " received one packet!";

    }

  return oss.str ();
```

```
}


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


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


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

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


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


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

```
  sink->SetRecvCallback (MakeCallback
(&RoutingExperiment::ReceivePacket, this));



  return sink;

}



std::string

RoutingExperiment::CommandSetup (int argc, char **argv)

{

  CommandLine cmd;

  cmd.AddValue ("CSVfileName", "The name of the CSV output file
name", m_CSVfileName);

  cmd.AddValue ("traceMobility", "Enable mobility tracing",
m_traceMobility);

  cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR",
m_protocol);

  cmd.Parse (argc, argv);

  return m_CSVfileName;

}



int

main (int argc, char *argv[])

{

  RoutingExperiment experiment;

  std::string CSVfileName = experiment.CommandSetup (argc,argv);
```

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

  int nSinks = 10;
  double txp = 7.5;

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

void
RoutingExperiment::Run (int nSinks, double txp, std::string
CSVfileName)
{
  Packet::EnablePrinting ();
```

```
  m_nSinks = nSinks;
  m_txp = txp;
  m_CSVfileName = CSVfileName;

  int nWifis = 20;

  double TotalTime = 120.0;
  std::string rate ("2048bps");
  std::string phyMode ("DsssRate11Mbps");
  std::string tr_name ("manet-routing-compare");
  int nodeSpeed = 20; //in m/s
  int nodePause = 0; //in s
  m_protocolName = "protocol";
uint32_t SentPackets = 0;
uint32_t ReceivedPackets = 0;
uint32_t LostPackets = 0;

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

  //Set Non-unicastMode rate to unicast mode
```

```
  Config::SetDefault
("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue
(phyMode));


  NodeContainer adhocNodes;

  adhocNodes.Create (nWifis);


  // setting up wifi phy and channel using helpers

  WifiHelper wifi;

  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);


  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();

  YansWifiChannelHelper wifiChannel;

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

  wifiChannel.AddPropagationLoss
("ns3::FriisPropagationLossModel");

  wifiPhy.SetChannel (wifiChannel.Create ());


  // Add a mac and disable rate control

  WifiMacHelper wifiMac;

  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",

                                "DataMode",StringValue (phyMode),

                                "ControlMode",StringValue (phyMode));
```

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

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



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

  NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac,
adhocNodes);



  MobilityHelper mobilityAdhoc;

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



  ObjectFactory pos;

  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");

  pos.Set ("X", StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));

  pos.Set ("Y", StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));



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

  streamIndex += taPositionAlloc->AssignStreams (streamIndex);



  std::stringstream ssSpeed;

  ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" <<
nodeSpeed << "]";

  std::stringstream ssPause;
```

```
  ssPause << "ns3::ConstantRandomVariable[Constant=" <<
nodePause << "]";

  mobilityAdhoc.SetMobilityModel
("ns3::RandomWaypointMobilityModel",

                          "Speed", StringValue (ssSpeed.str ()),

                          "Pause", StringValue (ssPause.str ()),

                          "PositionAllocator", PointerValue
(taPositionAlloc));

  mobilityAdhoc.SetPositionAllocator (taPositionAlloc);

  mobilityAdhoc.Install (adhocNodes);

  streamIndex += mobilityAdhoc.AssignStreams (adhocNodes,
streamIndex);

  NS_UNUSED (streamIndex); // From this point, streamIndex is
unused



  AodvHelper aodv;

  OlsrHelper olsr;

  DsdvHelper dsdv;

  DsrHelper dsr;

  DsrMainHelper dsrMain;

  Ipv4ListRoutingHelper list;

  InternetStackHelper internet;



  switch (m_protocol)

    {
```

```
  case 1:
    list.Add (olsr, 100);
    m_protocolName = "OLSR";
    break;
  case 2:
    list.Add (aodv, 100);
    m_protocolName = "AODV";
    break;
  case 3:
    list.Add (dsdv, 100);
    m_protocolName = "DSDV";
    break;
  case 4:
    m_protocolName = "DSR";
    break;
  default:
    NS_FATAL_ERROR ("No such protocol:" << m_protocol);
  }

if (m_protocol < 4)
  {
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
```

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

 NS_LOG_INFO ("assigning ip address");

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

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

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

```
    AddressValue remoteAddress (InetSocketAddress
(adhocInterfaces.GetAddress (i), port));

    onoff1.SetAttribute ("Remote", remoteAddress);



    Ptr<UniformRandomVariable> var =
CreateObject<UniformRandomVariable> ();

    ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i +
nSinks));

    temp.Start (Seconds (var->GetValue (100.0,101.0)));

    temp.Stop (Seconds (TotalTime));

   }



  std::stringstream ss;

  ss << nWifis;

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



  std::stringstream ss2;

  ss2 << nodeSpeed;

  std::string sNodeSpeed = ss2.str ();



  std::stringstream ss3;

  ss3 << nodePause;

  std::string sNodePause = ss3.str ();
```

```
  std::stringstream ss4;

  ss4 << rate;

  std::string sRate = ss4.str ();



  //NS_LOG_INFO ("Configure Tracing.");

  //tr_name = tr_name + "_" + m_protocolName +"_" + nodes +
"nodes_" + sNodeSpeed + "speed_" + sNodePause + "pause_" + sRate
+ "rate";



  //AsciiTraceHelper ascii;

  //Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream (
(tr_name + ".tr").c_str());

  //wifiPhy.EnableAsciiAll (osw);

  AsciiTraceHelper ascii;

  MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name +
".mob"));



  //Ptr<FlowMonitor> flowmon;

  //FlowMonitorHelper flowmonHelper;

  //flowmon = flowmonHelper.InstallAll ();



FlowMonitorHelper flowmon;

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

```
  NS_LOG_INFO ("Run Simulation.");


  CheckThroughput ();


  Simulator::Stop (Seconds (TotalTime));

  Simulator::Run ();

int j=0;

float AvgThroughput = 0;

Time Jitter;

Time Delay;


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);
```

```
NS_LOG_UNCOND("Src Addr" <<t.sourceAddress << "Dst Addr
"<< t.destinationAddress);

NS_LOG_UNCOND("Sent Packets=" <<iter->second.txPackets);

NS_LOG_UNCOND("Received Packets =" <<iter-
>second.rxPackets);

NS_LOG_UNCOND("Lost Packets =" <<iter->second.txPackets-iter-
>second.rxPackets);

NS_LOG_UNCOND("Packet delivery ratio =" <<iter-
>second.rxPackets*100/iter->second.txPackets << "%");

NS_LOG_UNCOND("Packet loss ratio =" << (iter->second.txPackets-
iter->second.rxPackets)*100/iter->second.txPackets << "%");

NS_LOG_UNCOND("Delay =" <<iter->second.delaySum);

NS_LOG_UNCOND("Jitter =" <<iter->second.jitterSum);

NS_LOG_UNCOND("Throughput =" <<iter->second.rxBytes *
8.0/(iter->second.timeLastRxPacket.GetSeconds()-iter-
>second.timeFirstTxPacket.GetSeconds())/1024<<"Kbps");



SentPackets = SentPackets +(iter->second.txPackets);

ReceivedPackets = ReceivedPackets + (iter->second.rxPackets);

LostPackets = LostPackets + (iter->second.txPackets-iter-
>second.rxPackets);

AvgThroughput = AvgThroughput + (iter->second.rxBytes * 8.0/(iter-
>second.timeLastRxPacket.GetSeconds()-iter-
>second.timeFirstTxPacket.GetSeconds())/1024);

Delay = Delay + (iter->second.delaySum);

Jitter = Jitter + (iter->second.jitterSum);



j = j + 1;
```

```
}


AvgThroughput = AvgThroughput/j;

NS_LOG_UNCOND("--------Total Results of the simulation----------
"<<std::endl);

NS_LOG_UNCOND("Total sent packets  =" << SentPackets);

NS_LOG_UNCOND("Total Received Packets =" <<
ReceivedPackets);

NS_LOG_UNCOND("Total Lost Packets =" << LostPackets);

NS_LOG_UNCOND("Packet Loss ratio =" <<
((LostPackets*100)/SentPackets)<< "%");

NS_LOG_UNCOND("Packet delivery ratio =" <<
((ReceivedPackets*100)/SentPackets)<< "%");

NS_LOG_UNCOND("Average Throughput =" << AvgThroughput<<
"Kbps");

NS_LOG_UNCOND("End to End Delay =" << Delay);

NS_LOG_UNCOND("End to End Jitter delay =" << Jitter);

NS_LOG_UNCOND("Total Flod id " << j);

monitor->SerializeToXmlFile("manet-routing.xml", true, true);


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


  Simulator::Destroy ();

}
```

Output:

```
--------Total Results of the simulation----------

Total sent packets  =26214
Total Received Packets =23269
Total Lost Packets =2945
Packet Loss ratio =11%
Packet delivery ratio =88%
Average Throughput =1.2971Kbps
End to End Delay =+1710040723906.0ns
End to End Jitter delay =+1333943260221.0ns
Total Flod id 1713
```

For protocol = 1,

```
--------Total Results of the simulation----------

Total sent packets  =3561
Total Received Packets =2815
Total Lost Packets =746
Packet Loss ratio =20%
Packet delivery ratio =79%
Average Throughput =2.0544Kbps
End to End Delay =+48059179115.0ns
End to End Jitter delay =+48778645715.0ns
Total Flod id 10
```

Conclusion:

From this lab, we learned about Queues, packet drops and their effect on congestion window size.

For this , we first create a simple dumbell Topology of six node using point to point links. Then we install the TCP socket among node1-node3,node2-node3 and node2-node4. After that we measure the throughput of node1—node3 flow at time 1s and node2—node3 flows at time 15s. At last we plot the graphical throughput.