NS3 Lab Manual

Prerequisites:
Student must have gone through following section of NS3 Tutorial

Basic script on NS3

https://www.nsnam.org/docs/tutorial/html/conceptual-overview.html

Walkthrough of Simple TCP Example

https://www.nsnam.org/docs/tutorial/html/tracing.html#walkthrough-fifth-cc

Walk through of TCP, UDP and Routing Protocols

You have been provided the simulator script to use in this lab: **tcp-net.cc**. You can place this script in any location, say in the scratch folder or the examples folder, and then run the script as follows

The "tcp-net.cc" simulation script is fairly straightforward, especially if you have looked at the examples in the ns-3 tutorial. It simulates topolgy of 8 nodes in a network. Data is transferred over TCP on the link. If you look at the first few lines in the main function in the script, you will see the following few lines.

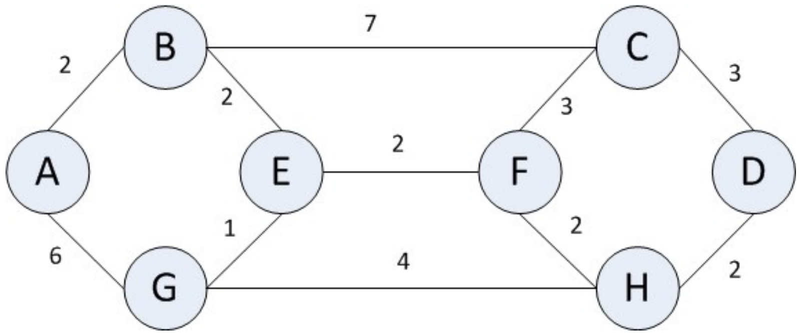
```
std::string tcp_variant = "TcpNewReno";
std::string bandwidth = "5Mbps";
std::string delay = "5ms";
double error_rate = 0.000001;
int queuesize = 10; //packets
int simulation_time = 10; //seconds
```

The above code configures the following parameters. You can change these values as needed in the exercises below.

- ➢ Link bandwidth between the two nodes. Default is 5 Mbps.
- ➢ One way delay of the link. Default is 5 milliseconds.
- ➢ Loss rate of packets on the link. Default is 0.000001. This covers losses other than those that occur due to buffer drops.
- ➢ Queue size of the buffer at node 0. Default is 10 packets.
- ➢ The TCP variant to use. Default is TCP NewReno.
- ➢ Simulation time. Default is 10 seconds

Analyzing simulation results

You will need to run simulations by varying the parameters above, and analyze the output files above to answer several questions about TCP. To understand what is happening with TCP in the simulation, it will be helpful to look at the following metrics.



Network Topology

To run script go to $NS3_HOME\ns-allinone-3.19\ns-3.19 and run

./waf --run tcp-net (note: run without .cc)

Here below different example scenarios are given, for each scenario analyze Animated Output of Flows, Evolution of the sender's cwnd over time, Throughput of flows

Scenario1 (Default): Single TCP flow from A to D

1 TCP flow from A to D

Scenario 2: Two Different TCP flow from Source A to Destination D

Note: Uncomment lines from “flow number 2 starts here” to “flow number 2 ends here”

1 TCP flow from A to D

1 TCP flow from A to D

Scenario 3: Two flows (1 TCP + 1 UDP)

Note: Uncomment lines from “flow number 3 starts here” to “flow number 3 ends here”

1 TCP flow from A to D

1 UDP flow from G to D

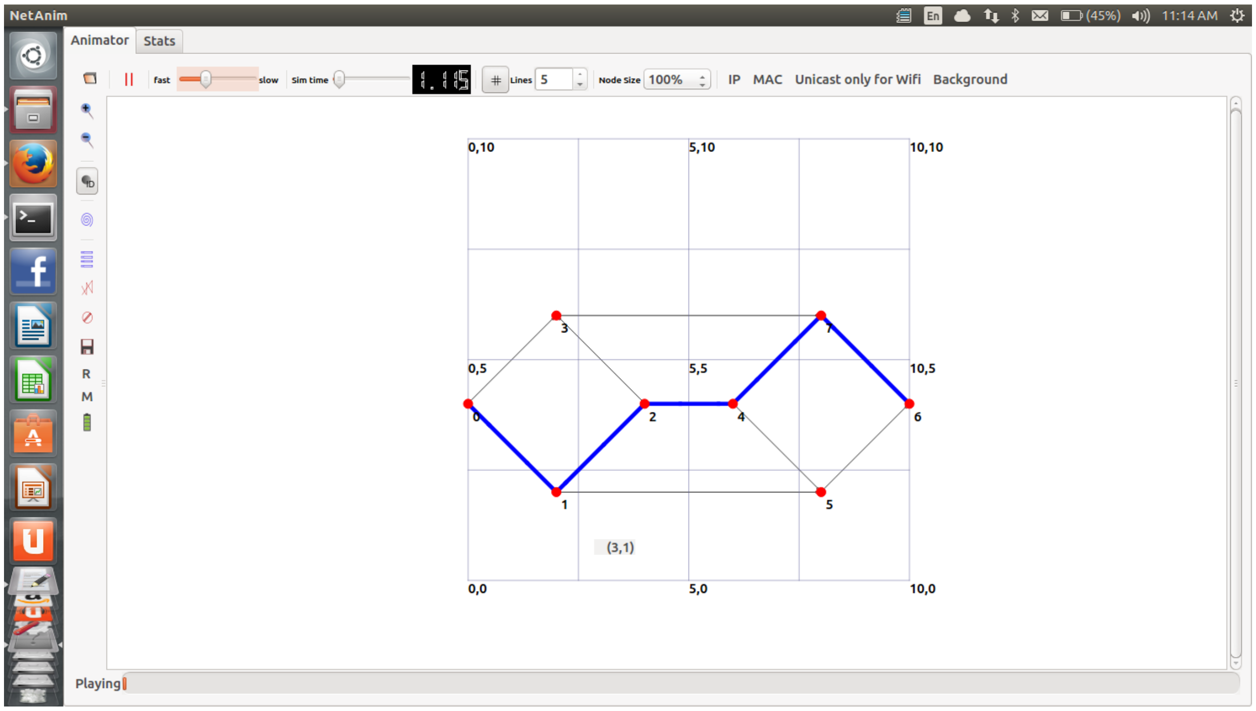
Animated Output of Flows

After successful running follow below steps

1. go to Ns-Allinone root directory and then go to netanim directory
2. cd netanim-3.104/
3. ./NetAnim

Select XML file for tracing which is named as tcp-net.xml in your root directory ($USER_HOME\ns3\ns-allinone-3.19\ns-3.19)

Analysis Result on NetAnim Animator tool

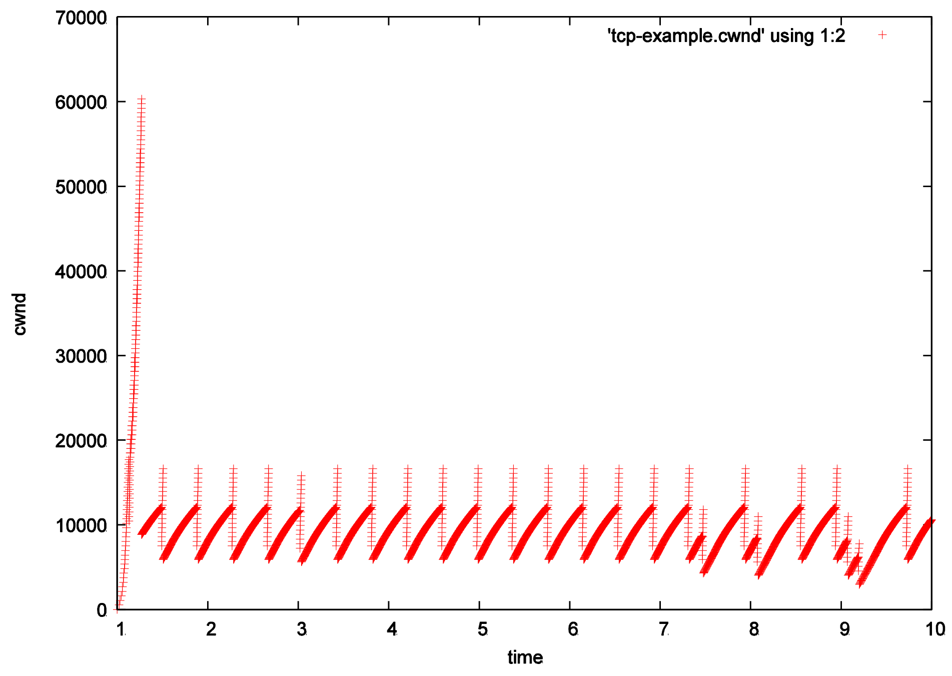


Evolution of the sender's cwnd over time.

The cwnd trace file, which is generated as part of the simulation output, is written whenever cwnd changes. So simply plotting all the values in the cwnd trace file is enough for you to visualize what happened to the cwnd during the simulation. You may use any graph plotting software. For example, here is a simple gnuplot script, say, "example.gpp", that generates a cwnd vs time plot.

```
set term postscript eps color
set output 'cwnd.eps'
set ylabel 'cwnd'
set xlabel 'time'
plot 'tcp-net.cwnd' using 1:2
```

Now, I store the 5 lines above in a file called "example.gp", and I store the cwnd trace file generated by ns-3 called "**tcp-net.cwnd**" also in the same folder. Then I run "gnuplot example.gp" at the command line. Then the graph "cwnd.eps" will be created in the same folder showing the cwnd vs time plot. **Note**: When you run more than one TCP flows then per flow one “.cwnd “ file will be generated and for each file you need to plot Cwnd vs. time graph.



Throughput of flows

When you run script output of script describes the flow based throughput.

For example when you run Scenario1 output of script will be like below

```
Flow 0 (10.1.1.1:49153 -> 10.1.10.2:8080) [Flow from Sender to Receiver]
 Tx Bytes: 9704044
 Rx Bytes: 9672940
 Throughput: 3.81006 Mbps
Flow 1 (10.1.10.2:8080 -> 10.1.1.1:49153) ) [Flow from receiver to Sender]
 Tx Bytes: 353320
 Rx Bytes: 353320
 Throughput: 0.139317 Mbps [Throughput is very less because receiver only send acknowledgement of sender]
```

The average throughput can also be easily obtained by opening the respective node's pcap file.

tcp-net-6-1.pcap // file contains info about packets through node 6, interface 1

(Tcp-net-6-1.pcap file in wireshark, and *checking out the "Summary" tab or "Conversations" tab under the "Statistics" menu to see various statistics.* (For more info about tracing go to https://www.nsnam.org/docs/tutorial/html/tracing.html)

To check for throughput graph go to Statistics->TCP Stream Graph-> Throughput Graph

Analyze and answer following question

1. Run the simulation with the default parameters and analyze for the following questions
   - What is the average throughput of the TCP transfer? What is the maximum expected value of throughput? Is the achieved throughput approximately equal to the maximum expected value?
   - How many times did the TCP algorithm reduce the cwnd, and why?

2. Start with the default config. Change the link bandwidth to 50Mbps (from 5 Mbps).
   - What is the average throughput of the TCP transfer? What is the maximum expected value of throughput? Is the achieved throughput approximately equal to the maximum expected value? If it is not, explain the reason for the difference.

- What other parameters in the simulation (amongst the ones exposed to you above) can you change to make sure that the throughput is close to the maximum expected value, for this link bandwidth? (Try out a few different simulations, and see what gets you close to the maximum.)

3. Start with the default config. Change the TCP variant to Tahoe, Reno, and NewReno (New Reno is used by default in NS3).
   - What difference do you observe in the TCP throughput across variants?
   - Pick a sample cwnd graph, one for each variant, and see the differences during the various stages (slow start, congestion avoidance, and fast recovery phases) for each variant. You can use the loss rate parameter to inject more or less losses as needed, to clearly identify the difference between the variants.

4. Start with the default config. And try to change the link weight of any link (e.g. F-H make it 6)
   - What difference do you observe in routing of packets?
   - What difference do you observe in throughput?

5. Analyze TCP behavior with Short flows and large flows
   - Use some flow running for small duration of time say(5 sec) and other flow running for large duration of time and analyze congestion window, throughput.

Equal-cost multi-path routing in NS3

Ecmp in NS3 was originally implemented in NS3 for per-packet basis. However later on, patches for per-flow ECMP routing were also enabled.
In this lab, we just simulate ECMP routing on per-packet basis.

ecmp-routing.cc

```
// This script illustrates the behavior of equal-cost multipath routing
// (ECMP) with Ipv4 global routing, across three equal-cost paths.
// Network topology:
//          n2
//         /  \        all links: point-to-point
//        /    \
//n0--n1-n3-n5----n6
//        \    /
//         \  /
//          n4
//Observe 3 equal cost paths:     n1-n2-n3          n1-n3-n5          n1-n4-n5
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("EcmpGlobalRoutingExample");
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);
  Config::SetDefault ("ns3::Ipv4GlobalRouting::RandomEcmpRouting",                BooleanValue(true));
          // Per-packet ECMP routing is eanbled.

  NS_LOG_INFO ("Create nodes.");
  NodeContainer c;
  c.Create (7);
  NodeContainer n0n1 = NodeContainer (c.Get (0), c.Get (1));
  NodeContainer n1n2 = NodeContainer (c.Get (1), c.Get (2));
  NodeContainer n1n3 = NodeContainer (c.Get (1), c.Get (3));
  NodeContainer n1n4 = NodeContainer (c.Get (1), c.Get (4));
  NodeContainer n2n5 = NodeContainer (c.Get (2), c.Get (5));
  NodeContainer n3n5 = NodeContainer (c.Get (3), c.Get (5));
  NodeContainer n4n5 = NodeContainer (c.Get (4), c.Get (5));
  NodeContainer n5n6 = NodeContainer (c.Get (5), c.Get (6));
```

```
InternetStackHelper internet;
internet.Install (c);

NS_LOG_INFO ("Create channels.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
NetDeviceContainer d1d3 = p2p.Install (n1n3);
NetDeviceContainer d1d4 = p2p.Install (n1n4);
NetDeviceContainer d2d5 = p2p.Install (n2n5);
NetDeviceContainer d3d5 = p2p.Install (n3n5);
NetDeviceContainer d4d5 = p2p.Install (n4n5);
NetDeviceContainer d5d6 = p2p.Install (n5n6);

NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.0.1.0", "255.255.255.0");
ipv4.Assign (d0d1);
ipv4.SetBase ("10.1.2.0", "255.255.255.0");
ipv4.Assign (d1d2);
ipv4.SetBase ("10.1.3.0", "255.255.255.0");
ipv4.Assign (d1d3);
ipv4.SetBase ("10.1.4.0", "255.255.255.0");
ipv4.Assign (d1d4);
ipv4.SetBase ("10.2.5.0", "255.255.255.0");
ipv4.Assign (d2d5);
ipv4.SetBase ("10.3.5.0", "255.255.255.0");
ipv4.Assign (d3d5);
ipv4.SetBase ("10.4.5.0", "255.255.255.0");
ipv4.Assign (d4d5);
ipv4.SetBase ("10.5.6.0", "255.255.255.0");
ipv4.Assign (d5d6);

NS_LOG_INFO ("Populate routing tables.");
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

NS_LOG_INFO ("Create Applications.");
uint16_t port = 9;
OnOffHelper onoff ("ns3::UdpSocketFactory",
               InetSocketAddress (Ipv4Address ("10.5.6.2"), port));
onoff.SetConstantRate (DataRate ("100kbps"));
onoff.SetAttribute ("PacketSize", UintegerValue (500));
ApplicationContainer apps;
for (uint32_t i = 0; i < 10; i++)
  {
    apps.Add (onoff.Install (c.Get (0)));
  }
apps.Add (onoff.Install (c.Get (1)));
```

```
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (5.0));
  PacketSinkHelper sink ("ns3::UdpSocketFactory",
                 Address (InetSocketAddress (Ipv4Address::GetAny (), port)));
  sink.Install (c.Get (6));

  // Trace different interfaces (namely 2,3,4) in node 1
  p2p.EnablePcap ("ecmp-global-routing", 1, 2);
  p2p.EnablePcap ("ecmp-global-routing", 1, 3);
  p2p.EnablePcap ("ecmp-global-routing", 1, 4);

  AnimationInterface anim ("ecmp_routing.xml");
  anim.SetConstantPosition(c.Get(0),2.0,1.0);
  anim.SetConstantPosition(c.Get(1),3.0,1.0);
  anim.SetConstantPosition(c.Get(2),4.0,0.0);
  anim.SetConstantPosition(c.Get(3),4.0,1.0);
  anim.SetConstantPosition(c.Get(4),4.0,2.0);
  anim.SetConstantPosition(c.Get(5),5.0,1.0);
  anim.SetConstantPosition(c.Get(6),6.0,1.0);

  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
}
```

Running this file:

1 Copy this file in folder *ns-allinone-3.23/ns-3.23/src/internet/examples/*
2 Use terminal to run following commands:

```
cd ns-allinone-3.23/ns-3.23/
cp src/internet/examples/ecmp-routing.cc scratch/ecmp-routing.cc
./waf –run scratch/ecmp-routing(Note: omit '.cc')
```

Simulation of this file:

```
cd ns-allinone-3.19/netanim-3.106/
./NetAnim
```

Now open ecmp_routing.xml contained in folder *ns-allinone-3.23/ns-3.23/*
Run simulation.
Observe that since we have 3 equal cost paths from node 1 to destination, therefore the packets at node 1 get splitted into 3 paths.

Tracing nodes using pcap files:

You can see the trace files generated in folder ns-allinone-3.23/ns-3.23/ using wireshark:

```
ecmp-global-routing-1-2.pcap     // file contains info about packets through node 1, interface 2
ecmp-global-routing-1-3.pcap     // file contains info about packets through node 1, interface 3
ecmp-global-routing-1-4.pcap     // file contains info about packets through node 1, interface 4
```

Observe the packets using filter eg ip.src == 10.2.2.1 and various other fields in packet header.