Data Networks
Lab 3: Experimenting Network Layer with NS3

Dr. Mohammad Reza Pakravan

Due on 5 June, 2020

Introduction

The network layer is concerned with getting packets from the source all the way to the destination. Getting to the destination may require making many hops at intermediate routers along the way. This function clearly contrasts with that of the data link layer, which has the more modest goal of just moving frames from one end of a wire to the other. To achieve its goals, the network layer must know about the topology of the communication subnet (i.e., the set of all routers) and choose appropriate paths through it. It must also take care to choose routes to avoid overloading some of the communication lines and routers while leaving others idle. It is up to the network layer to deal with them. A router in the network needs to be able to look at a packet's destination address and then determine the output port which is the best choice to get the packet to that destination. The router makes this decision by consulting a forwarding table. Routing algorithms are required to build the routing tables and hence forwarding tables. The basic problem of routing is to find the lowest-cost path between any two nodes, where the cost of a path equals the sum of the costs of all the edges that make up the path. Routing is achieved in most practical networks by running routing protocols among the nodes. The protocols provide a distributed, dynamic way to solve the problem of finding the lowest-cost path in the presence of link and node failures, and changing edge costs. In this exercise, you will see some NS3 features to simulate routing.

Routing in Fixed wired Networks

In this part we want to run a simulation to investigate the routing algorithm in the following topology. The topology is the same as the example, the sink tree of which is calculated in the class based on Dijkstra's shortest path algorithm.

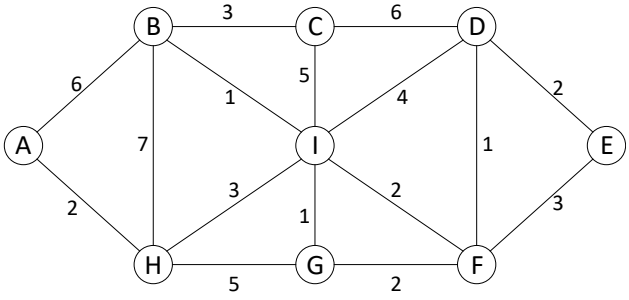

Figure 1: Network Topology

Initialization

As the first step you have to include the modules you need as follows:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/netanim-module.h"
#include "ns3/applications-module.h"
#include "ns3/animation-interface.h"
#include "ns3/point-to-point-layout-module.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/ipv4-list-routing-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor-module.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
```

The next line enables you in showing some messages while running the code:

```
NS_LOG_COMPONENT_DEFINE ("Lab3_part1");
```

At the beginning of the main function, you should declare some parameters for your simulation as follows:

```
uint32_t PacketSize = 512; // bytes
std::string DataRate ("1Mbps");
uint16_t num_Nodes = 9;
uint16_t UDPport = 9;
bool tracing = false;
```

"CommandLine" object allows the user to override any of the defaults, including the above parameters, at run-time:

```
CommandLine cmd;
cmd.AddValue ("PacketSize", "size of application packet sent", PacketSize);
cmd.AddValue ("DataRate", "rate of pacekts sent", DataRate);
cmd.AddValue ("tracing", "turn on ascii and pcap tracing", tracing);
cmd.Parse (argc, argv);
```

The following lines override the default values of the OnOffApplication application used in data generation:

```
Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue(PacketSize));
Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue (DataRate));
Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",BooleanValue(true));
```

Finally, you should enable Packet Meta data for animator and the name of file for animation output:

```
ns3::PacketMetadata::Enable();
std::string animFile = "lab3_part1.xml" ;
```

Network Topology

In the next step, you have to create the network topology. Network nodes are stored in a container class called NodeContainer.

```
NodeContainer nodes;
nodes.Create (num_Nodes);
```

Now, you should group nodes to make links between them.

```
NodeContainer AB = NodeContainer (nodes.Get (0), nodes.Get (1));
NodeContainer BC = NodeContainer (nodes.Get (1), nodes.Get (2));
NodeContainer CD = NodeContainer (nodes.Get (2), nodes.Get (3));
NodeContainer DE = NodeContainer (nodes.Get (3), nodes.Get (4));
NodeContainer EF = NodeContainer (nodes.Get (4), nodes.Get (5));
NodeContainer FG = NodeContainer (nodes.Get (5), nodes.Get (6));
NodeContainer GH = NodeContainer (nodes.Get (6), nodes.Get (7));
NodeContainer HA = NodeContainer (nodes.Get (7), nodes.Get (0));
NodeContainer BH = NodeContainer (nodes.Get (1), nodes.Get (7));
NodeContainer DF = NodeContainer (nodes.Get (3), nodes.Get (5));
NodeContainer BI = NodeContainer (nodes.Get (1), nodes.Get (8));
NodeContainer CI = NodeContainer (nodes.Get (2), nodes.Get (8));
NodeContainer DI = NodeContainer (nodes.Get (3), nodes.Get (8));
NodeContainer FI = NodeContainer (nodes.Get (5), nodes.Get (8));
NodeContainer GI = NodeContainer (nodes.Get (6), nodes.Get (8));
NodeContainer HI = NodeContainer (nodes.Get (7), nodes.Get (8));
```

Now you should assign bandwidth and delay to each link.

```
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));
NetDeviceContainer dAB = p2p.Install (AB);
```

Do it for other links with the values like link AB.

Network Layer Parameters

In this part, you will configure the network parameters. First, you have to install IPv4 on the network via using InternetStackHelper. This helper uses static routing as its first priority algorithm for routing; therefore, we redefine the priority:

```
NS_LOG_INFO("Setting routing protocols");
Ipv4StaticRoutingHelper staticRouting;
Ipv4GlobalRoutingHelper globalRouting;
Ipv4ListRoutingHelper list;
list.Add(staticRouting,0);
list.Add(globalRouting,10);
// Install network stacks on the nodes
InternetStackHelper internet;
internet.SetRoutingHelper(list);
internet.Install(nodes);
```

Now, you should assign IP to each node for routing and also assign the metric of each link to it.

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer iAB = ipv4.Assign (dAB);
iAB.SetMetric(0,6);
iAB.SetMetric(1,6);
ipv4.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer iBC = ipv4.Assign (dBC);
iBC.SetMetric(0,3);
iBC.SetMetric(1,3);
ipv4.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer iCD = ipv4.Assign (dCD);
iCD.SetMetric(0,6);
iCD.SetMetric(1,6);
ipv4.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer iDE = ipv4.Assign (dDE);
iDE.SetMetric(0,2);
iDE.SetMetric(1,2);
ipv4.SetBase ("10.1.5.0", "255.255.255.0");
Ipv4InterfaceContainer iEF = ipv4.Assign (dEF);
iEF.SetMetric(0,3);
iEF.SetMetric(1,3);
ipv4.SetBase ("10.1.6.0", "255.255.255.0");
Ipv4InterfaceContainer iFG = ipv4.Assign (dFG);
iFG.SetMetric(0,2);
iFG.SetMetric(1,2);
ipv4.SetBase ("10.1.7.0", "255.255.255.0");
Ipv4InterfaceContainer iGH = ipv4.Assign (dGH);
iGH.SetMetric(0,5);
iGH.SetMetric(1,5);
ipv4.SetBase ("10.1.8.0", "255.255.255.0");
Ipv4InterfaceContainer iHA = ipv4.Assign (dHA);
iHA.SetMetric(0,2);
iHA.SetMetric(1,2);
ipv4.SetBase ("10.1.9.0", "255.255.255.0");
Ipv4InterfaceContainer iBH = ipv4.Assign (dBH);
iBH.SetMetric(0 ,7);
iBH.SetMetric(1,7);
ipv4.SetBase ("10.1.10.0", "255.255.255.0");
Ipv4InterfaceContainer iDF = ipv4.Assign (dDF);
iDF.SetMetric(0,1);
iDF.SetMetric(1,1);
ipv4.SetBase ("10.1.11.0", "255.255.255.0");
Ipv4InterfaceContainer iBI = ipv4.Assign (dBI);
iBI.SetMetric(0,1);
iBI.SetMetric(1,1);
ipv4.SetBase ("10.1.12.0", "255.255.255.0");
Ipv4InterfaceContainer iCI = ipv4.Assign (dCI);
iCI.SetMetric(0,5);
```

```
iCI.SetMetric(1,5);
ipv4.SetBase ("10.1.13.0", "255.255.255.0");
Ipv4InterfaceContainer iDI = ipv4.Assign (dDI);
iDI.SetMetric(0,4);
iDI.SetMetric(1,4);
ipv4.SetBase ("10.1.14.0", "255.255.255.0");
Ipv4InterfaceContainer iFI = ipv4.Assign (dFI);
iFI.SetMetric(0,2);
iFI.SetMetric(1,2);
ipv4.SetBase ("10.1.15.0", "255.255.255.0");
Ipv4InterfaceContainer iGI = ipv4.Assign (dGI);
iGI.SetMetric(0,1);
iGI.SetMetric(1,1);
ipv4.SetBase ("10.1.16.0", "255.255.255.0");
Ipv4InterfaceContainer iHI = ipv4.Assign (dHI);
iHI.SetMetric(0,3);
iHI.SetMetric(1,3);
```

Finally, you have to add the following code to initialize routing database and set up the routing tables in the nodes:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Building Flows

After doing the preceding part, now it is time to determine the application with which the nodes will function. You should install UDP application on node A and E. First, use the following code:

```
PacketSinkHelper UDPsink ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), UDPport))
ApplicationContainer App;
NodeContainer SourceNode = NodeContainer (nodes.Get (0));
NodeContainer SinkNode = NodeContainer (nodes.Get (4));
```

To Create a UDP packet sink to receive these packets use the following codes:

```
App = UDPsink.Install (SinkNode);
App.Start (Seconds (0.0));
App.Stop (Seconds (10.0));
Address E_Address(InetSocketAddress(iEF.GetAddress (0), UDPport));
```

To Create a UDP packet source to send these packets type the following codes:

```
OnOffHelper UDPsource ("ns3::UdpSocketFactory", E_Address);
UDPsource.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
UDPsource.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
UDPsource.Install(SourceNode);
App.Start (Seconds (1.0));
App.Stop (Seconds (10.0));
```

Starting The Simulation

You should just add the following two lines to your code before running it.

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

After everything is done, you have to destroy the simulation using the following command

```
Simulator::Destroy();
```

NetAnim, Wireshark and other analysis

You should determine your nodes places for NetAnim. Do it with these codes:

```
AnimationInterface anim (animFile);
Ptr<Node> n = nodes.Get (0);
anim.SetConstantPosition (n, 0, 20);
n = nodes.Get(1);
anim.SetConstantPosition(n,10,10);
n = nodes.Get(2);
anim.SetConstantPosition(n,20,0);
n = nodes.Get(3);
anim.SetConstantPosition(n,30,10);
n = nodes.Get(4);
anim.SetConstantPosition(n,40,20);
n = nodes.Get(5);
anim.SetConstantPosition(n,30,30);
n = nodes.Get(6);
anim.SetConstantPosition(n,20,40);
n = nodes.Get(7);
anim.SetConstantPosition(n,10,30);
n = nodes.Get(8);
anim.SetConstantPosition(n,20,20);
```

You can use the following code to generate the xml file used in netAnim.

```
if (tracing == true)
{
AsciiTraceHelper ascii;
p2p.EnableAsciiAll (ascii.CreateFileStream ("Lab3_part1.tr"));
p2p.EnablePcapAll ("Lab3_part1");
}
```

To print routing tables you should add the following codes:

```
Ptr<OutputStreamWrapper> stream1 = Create<OutputStreamWrapper> ("Table1", std::ios::out);
Ipv4GlobalRoutingHelper helper2;
helper2.PrintRoutingTableAllAt(Seconds(2.0),stream1);
```

Questions

Do the simulation and compare the result with what calculated in the class. Now change the metrics of links AB and BI to 1 and 2, respectively. Again do the simulation and investigate whether the path for sending packets from A to E has been changed or not. Change the metric of link AH to this number: $\left\lfloor \frac{\text{Two last digits of your student number}}{10} \right\rfloor + 3$, for example if your student number is 88223987, the metric of link AB will be: $\left\lfloor \frac{87}{10} \right\rfloor + 3 = 11$. Again do the simulation and find the path by means of which A sends its packets to E.
Now, we want to evaluate the algorithm when a node sets down. We can make a node set down by these codes (You have to set down the links connected to this node):

```
Ptr<Node> node1=nodes.Get(8);
Ptr<Ipv4> ipv41=node1->GetObject<Ipv4>();
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,1);
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,2);
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,3);
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,4);
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,5);
Simulator::Schedule(Seconds(3),&Ipv4::SetDown,ipv41,6);
```

And to check the routing tables you can print the routing tables after this alternation by the following codes:

```
Ptr<OutputStreamWrapper> stream2 = Create<OutputStreamWrapper> ("Table2", std::ios::out);
helper2.PrintRoutingTableAllAt(Seconds(4.0),stream2);
```

Now check the tables to find the new sink tree after the alternation. You can use NetAnim to see the routing process easily.
In all above parts, you should take some snapshots from NetAnim to compare the results obtained by simulation with what you calculate manually by the method mentioned in the class.

What Should I Do?

You must upload anything that you've been asked to upload and you should also answer the questions in your report. Your report should be complete and you should fully explain the API that you have used in your code. Make a folder for each section and put code and output files of each section in its corresponding folder. Compress all files and rename the compressed file to STUDENT_ID_LAB3.zip.
If you have any questions regarding the problem statement or understanding the concept, feel free to ask in cw forum.