Data Communication Networks, Lab #3

Instructor: Dr. MohammadReza Pakravan

MAC Sublayer Simulation Using NS3

1 Section One - Intro

This homework is about the MAC sublayer and its simulation using the NS3 software and pursues two main goals, which will be mentioned below . The MAC sublayer is one of the most fundamental parts of any communication network and is designed to answer one of the intrinsic problems of communications: how can we prevent multiple nodes talking with a single node simultaneously? This problem arises even in the simplest form of communicating, namely verbal communication, where multiple persons talking with an individual can ruin all of the sent messages. Therefore, special care and strong ideas are needed while dealing with this issue. Multiple solutions, like ALOHA, CSMA and so on have been proposed to tackle the problem, and our first main goal in this lab is getting familiar with the simulation techniques and basics of CSMA.

Notice: A question that usually emerges is that why, then, if so important, MAC is a sublayer and not a layer? The answer lies in the nature of the OSI model. It is recommended that you think about and answer this question.

NS3 is a powerful simulaton tool used for testing and analyzing various network protocols and scenarios. It is based on the C++ language and provides the user with a predefined API to design a hypothetical network and modify the important parameters. A background of C programming is needed for an efficient use of the software. Our second goal is to get familiar with the NS3 environment (it doesn't have a GUI though :( ) and some of its functions to gain an overall insight on using it. It is worth mentioning that NS3 is a strong tool and is used extensively all around the world.

As a final note, this lab is designed in a way that will lead you to look for useful functions and the correct way to use them, on your own. One of the most critical skills that an individual, and in particular a network engineer should have is the ability to do extensive reseach about various subjects, and not to lose patience while doing so. The networking world is a vast space full of interesting things, and is growing with a substantial speed, which makes staying up to date a little harder. Therefore, one can state the above mentioned skill as a hidden - but **golden** - goal of this lab (or even this course).

P.S: you can use This link for finding the useful functions. Also, The examples located in the NS3 folder, especially the tutorial folder, can be of great help.

2 Section Two - Getting Familiar With NS3

2.1 Commandline Functions

In this section we will get familiar with the basic functions that are needed in almost every simulation and also the routine procedures that have to be done constantly. It is assumed that you have installed NS3 and Wireshark. A guideline for installing NS3 exists in the net. The installation process can be tricky, so if you haven't installed it yet, do it ASAP to have more time for the excercises.
First, lets see how we can compile and run a program. Open a text editor (I suggest ATOM, a great experience!) and copy the code below, which is a simple program to add two values and print the output:

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


using namespace ns3;
NS_LOG_COMPONENT_DEFINE("Lab #3");

int main(int argc, char *argv[]){
  int a;
  int b;
  CommandLine myCMD;
  myCMD.AddValue("a" , "This is a" , a);
  myCMD.AddValue("b" , "This is b" , b);
  myCMD.Parse(argc , argv);
  std::cout << "a + b = " << a + b;
}
```

Save the file with the *.cc* extension. Now, copy the file to the directory below:

NS3_Directory/Inside_The_NS3_Folder/ns-X.YZ/scratch

Where NS3_Directory is the place where you have installed NS3 and ns-X.YZ is a folder inside the NS3 folder, with X, Y and Z specifying the version you are using. Scratch is the folder where you should put your codes to be able to compile and run them. In my own laptop, this is the directory:

Downloads/ns-allinone-3.26/ns-3.26/scratch

Now open a terminal and *cd* to the directory below:

NS3_Directory/Inside_The_NS3_Folder/ns-X.YZ

copy this command:

```
./waf --run "scratch/file_name --a=2 --b=5"
```

You should see a line in the output, showing the sum of a and b, which is 7. Several important notes must be stated:

- The basic command for building and running is *./waf –run "scratch/File_name"*. File_name should not have the *.cc* extension.
- In this example, the values of a and b are not specified in the code and are passed to the code via the terminal. This is enabled using the command line functions. Search the web to get more

familiar with these functions. Notice also the way the command mentioned above changes to pass the values of a and b. You will need this functionality while testing a similar scenario with different values for a specified set of parameters. Changing the values inside the code can be tiring in the long run.

2.2 Network Nodes

We will continue using the text editor for the rest of the code. It is obvious that the first step of every network simulation is adding a number of nodes. Check the *NodeContainer* class in the NS3 documentation and add two nodes to the network. We will mostly work with point-to-point networks, so we have to set the important parameters in the code. Check the *PointToPointHelper* class in the documentation and derive the useful functions. Instantiate a PointToPointHelper and Set the channel delay and the bitrate of the devices to 5ms and 10Mbps, respectively. Note that the first one is a channel attribute and the second one is a device attribute. The next step is to assign a device to each node and make the devices obey the point-to-point parameters. use the function below to do this:

```
devices = pointToPoint.Install(nodes);
```

where "devices" is an instance of the *NetDeviceContainer* class, "nodes" is an instance of the *NodeContainer* class and "pointToPoint" is a *PointToPointHelper* (It should be clear that you can choose other names for these instances and this is just an example).

2.3 Network Layer Parameters

The network layer is the layer before Data Link layer. The main purpose of this layer is to route the packets from the destination to the source. One of the most popular network layer protocols is the IP. You will gain sufficient knowledge about this layer during the course.
In this example, the nodes need IP addresses to communicate with each other. First of all, check the *InternetStackHelper* class and create an instance of this class. Install this instance on the nodes that you've created in the last part. Then, look for the *ipv4AddressHelper* class, instanciate one and learn how to set a base for the IP addresses. The base is consisted of a base address and a netmask. These are IP address parameters, so if you don't know what they mean, don't worry, although I suggest a little bit of searching. Set the base address to "*st.nu*.1.0" and the netmask to "255.255.255.0", where *stnu* is the last four digits of your student number. You must also create an *Ipv4InterfaceContainer* to hold the IP interfaces that you have assigned to the nodes. You should find a detailed description in the documentation.

2.4 Application

After doing the preceding part, now its time to determine the application with which the nodes will function. There exist several applications in the NS3 environment. We will work with two of them, The *OnOffApplication* and the *UdpEcho*. In the OnOffApplication, a node which works with the application sends a Constant Bit Rate (CBR) stream to a destination at regular times, which is called the *OnTime*. At other times, the node sleeps until it reaches another OnTime. The interval in which the node sleeps is called the *OffTime*. The node alternates back and forth between these two states. in this part, you should check the documentation and learn how to use the *OnOffHelper* class to make a node have the functionality mentioned above. select one of the two nodes that we have in the network to use this application. creating an instance of the OnOffApplication requires two arguments. Use the format below:

```
OnOffHelper myOnOff("ns3::UdpSocketFactory" , Address(InetSocketAddress(IP
    address of the destination node)));
```

You can get the IP address of the destination node by using the *GetAddress* method of the Ipv4InterfaceContainer you have created. Set the packet size, bitrate, OnTime and OffTime to whatever value you want. Notice that you have to set the bitrate and the bitrate that you have set in the first part was for learning to work with the PointToPointHelper class methods and attributes. Use the *ConstantRandomVariable* class for the OnTime and OffTime attributes and keep in mind that this attributes are tricky and you have to be careful, because the commands vary between different releases of NS3, so its better to look for the correct way of using the methods and passing arguments to them based on the NS3 release you are working with. After handling the attributes, use an *ApplicationContainer* to assign this application on the node that you have selected in your mind as the node using the application. Start the ApplicationContainer and then start the simulation and kill it using the commands below:

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

Observe the output and take a snapshot. Answer the following questions:

---

1. What are the numbers that you are seeing in the output?
2. Do you recognize a pattern in the sending times? Explain.
3. Change the attributes of the OnOffHelper and analyze the differences. Verify your answer to the second question.
4. Create an ExponentialRandomVariable object and set the packet size attribute of the OnOffHelper using this object. An easy method of doing this can be found at the documentation. Take an snapshot and show the difference.
5. Name some of the attributes that you can change in the PointToPointHelper class.

Now you should use the UdpEcho application. delete the OnOffHelper part from your code. You should use the *UdpEchoServerHelper* and *UdpEchoClientHelper* classes to make one of your nodes be an UDP echo client and the other one an UDP echo server. Set the *MaxPackets*, *Interval* and *PacketSize* attributes to any value that you want. You should use two ApplicationContainers for your nodes: one for the client and one for the server. You should also determine a start time and a stop time for both of the ApplicationContainers. Start the simulation, take an snapshot and answer the following questions:

---

1. What are the numbers that you are seeing in the output?
2. What is the interval parameter?
3. Increase the channel delay and report the results.
4. Change the packet size attribute and report the results.

2.5 NetAnim and Wireshark

In this part we will focus on enabling outputs in the code to see the network and the packets. We will work with NetAnim and Wireshark. NetAnim is an utility which comes with the installed NS3 but you should install wireshark separately. In the previous sections you have defined and used a PointToPointHelper. You can enable outputs using this class.
NS3 has two types of text outputs: trace files and Pcap files. Trace files contain written information about the events that happen. Pcap files contain similar information, and you can view them using Wireshark. Another useful output type of NS3 is the XML output file, which contains the schematic of the network and the nodes. To enable this kind of output you don't need the PointToPointHelper class.

you need the *AnimationInterface* class. You can view the XML files using NetAnim. To do this simply open a terminal and cd to the NetAnim directory located in the NS3 directory and type ./NetAnim. The output files of the simulation can be found inside the NS3 folder, the same place where the scratch folder is located.
for this part, enable all the above mentioned outputs with this name:

LAB3_StudentID

---

1. Enable all the outputs and take screenshots from all of them. Your ZIP file should contain them all.

---

3 MAC Sublayer

In this part it is assumed that you have fulfilled the tasks mentioned in the previous parts and you have a basic understanding of how NS3 works. We will simulate a network in which a number of nodes are connected to each other via Ethernet, and we will model this using CSMA because we know that Ethernet uses CSMA/CD for avoiding collision in the channel. Create a network with five nodes and connect them using *CsmaHelper*. The first node will be a UdpEchoClient and will send packets to the fifth node, which is an UdpEchoServer. The second node too will be an UdpEchoClient and its corresponding UdpEchoServer will be the fourth node. Enable the XML output and include it in the package that you will upload. You should also upload a snapshot of the output of the simulation, which is printed in the terminal.

---

1. Take some time and dive deeper in the IEEE 802.16 MAC protocol. You will be asked of its properties.
2. A second scenario: the UdpEchoServer corresponding to the second node is the first node. Repeat all the above mentioned steps.

Now we will try to add some WiFi nodes to the topology. We will use the topology depicted in figure 1. The CSMA nodes have nothing new. For adding the WiFi nodes, however, we need more API. Also keep in mind that the leftmost CSMA node and the WiFi access point are connected via a point to point link, which you worked with in the first part. Now, First create a WiFi channel and a WiFi PHY layer. You need the *YansWiFiChannelHelper* and the *YansWiFiPhyHelper* classes. Use their default modes. Use the created channel to the set the channel of the created phy object. Next, create a *WiFiHelper* object and name it *myWiFi*. Then copy this line to your code:

```
1 myWiFi.SetRemoteStationManager ("ns3::AarfWifiManager");
```

The remaining parts of the procedure are not explained here and you should look for them in the net and the documentation and the ns3 examples. You will come across objects like *WiFiMacHelper*. You also have to specify the mobility model of the WiFi nodes. ns3 has different mobility models and you can find sufficient information by searching the net. Also keep in mind that you will need two different node containers for the non-access-point WiFi nodes and the WiFi access point. Their MAC APIs have a slight difference.

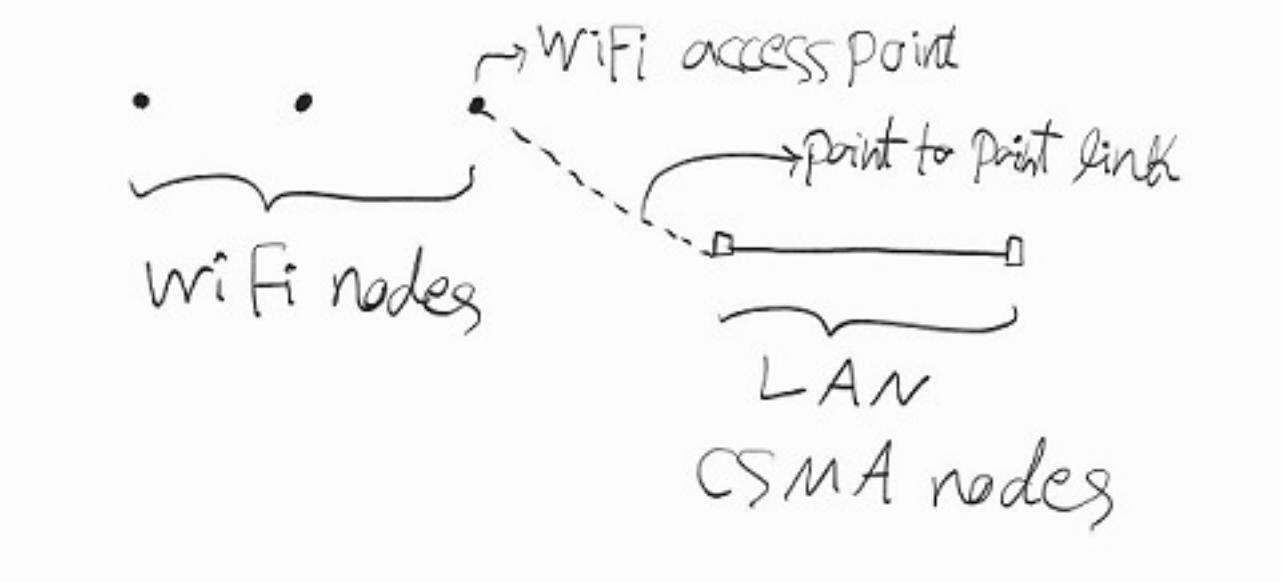

Figure 1: The Topology.

```
WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));
NetDeviceContainer staDevices;
staDevices = myWiFi.Install (phy, mac, wifiStaNodes);


mac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid));
NetDeviceContainer apDevices;
apDevices = myWiFi.Install (phy, mac, wifiApNode);
```

In the code brought above, wifiStaNodes is an object for the non-access-point WiFi nodes and wifiApNode is an object for the WiFi access point. After setting the mobility model, the rest of the code is made of what you have seen in the first part: install a *UdpEchoServer* on the rightmost CSMA node and a *UdpEchoClient* on the leftmost WiFi node. Don't forget to assing IPs to the nodes. The IPs should be in the format explained in the first section. Run the code and do the following:

---

1. Take screenshots from your simulation and include them in your report. The normal ns3 output in the terminal is sufficient.
2. You need to give *SSID* for your WiFi network. What is an SSID? try giving different SSIDs for the non-access-point nodes and the access point and take an screenshot from the output.
3. Change the position change step in the mobility model. Position change is the amount a node's position will change in different mobility models. Increase the change step and the rectangle (the available place for the nodes to move) size. Does this affect the output? Show the results with screenshots.

---

4 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. Feel free to ask any questions in the Telegram group of the course.

Important notice:

As a final note, your professional and academic characteristics are being shaped in the university, during the numerous HWs and projects that you must work on. Therefore, being responsible and trying to tackle challenges with hard work is very crucial. We believe that you can tackle the challenges of this course on your own, although team work and discussion is also supported. It is important for you to put some time on the HWs to do them. All being said, we strictly prohibit any kind of copying which leads to a poor character and no understanding of the course. Copying includes copying from the previous semesters. We hope that you understand our worry.