**LABWORK 5**

**A simulation scenario**

Team member

|  |  |
| --- | --- |
| Phạm Vũ Hải | BI11-074 |
| Nguyễn Đăng Khoa | BI11-122 |
| Tạ Đình Thái Nhân | BI11-205 |
| Nguyễn Tùng Lân | BI11-134 |
| Văn Quốc An | BI11-004 |
| Vũ Đức Kiên | BI11-128 |

1. **DESIGN**

In this assignment, we are evaluating the performance of the CSMA/CA protocol in WiFi networks working in ad-hoc mode. The goal is to evaluate the performance of the protocol without the RTS/CTS scheme when the number of nodes within a communication range increases from 2 to 30.

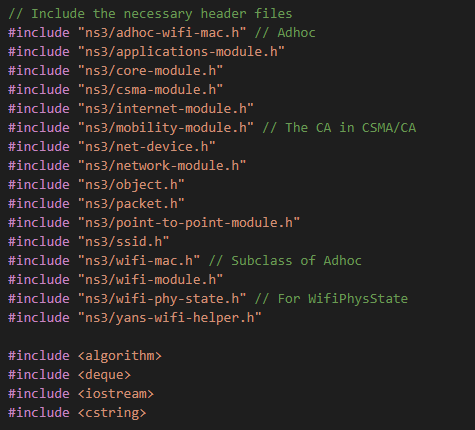
To achieve this goal, we will create a network simulation with the following steps:

* Create a node container and add nodes to it.
* Create a WiFi helper and set some attributes.
* Create a WiFi MAC helper and set it to use an ad hoc network.
* Create a WiFi PHY helper and set it to use a YansWifiChannel.
* Install the WiFi devices on the nodes.
* Set the callback functions for packet transmission and reception.
* Create a mobility helper and set the nodes to have random positions.
* Create an internet stack helper and install it on the nodes.
* Assign IP addresses to the devices.
* Create an OnOffHelper on each node and set it to send packets periodically.
* Enable pcap tracing for the devices.
* Run the simulation and collect data.

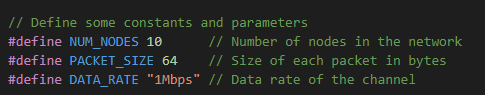
1. **IMPLEMENTING**

To implement the simulation, we will use NS-3, a discrete-event network simulator. We will create a C++ program that sets up the simulation as described in Part 1 and runs it.

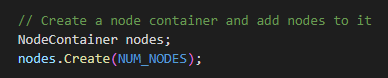
We start by including the necessary header files for the NS-3 modules we will be using:



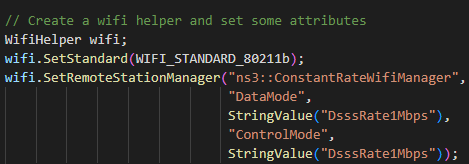
Next, we define some constants and parameters, such as the number of nodes in the network (10 nodes) and the size of each packet in bytes:



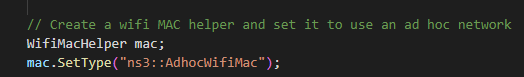
We create a node container and add nodes to it using the NodeContainer class:



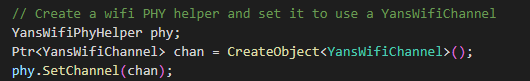
We then create a WiFi helper and set some attributes, such as the WiFi standard and the remote station manager:



And a WiFi MAC helper and set it to use an ad hoc network:



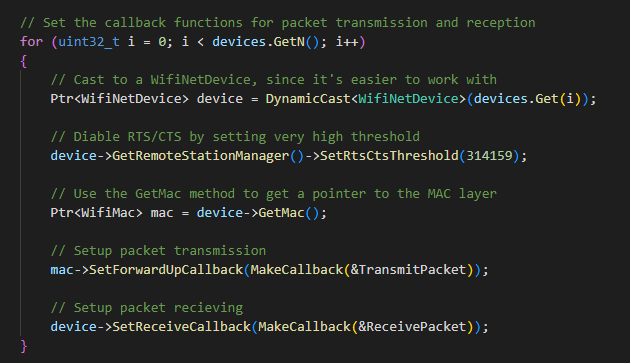
We create a WiFi PHY helper and set it to use a YansWifiChannel:



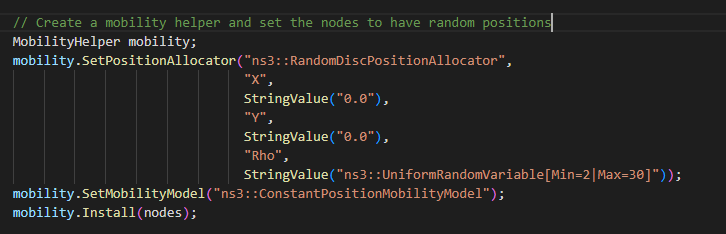
Next we install the WiFi devices on the nodes using the Install method of the WiFi helper:



We set the callback functions for packet transmission and reception using the SetForwardUpCallback and SetReceiveCallback methods of the WiFi MAC and NetDevice classes, respectively:



We create a mobility helper and set the nodes to have random positions using the SetPositionAllocator method of the MobilityHelper class:



We install the mobility model on the nodes using the Install method of the MobilityHelper class:

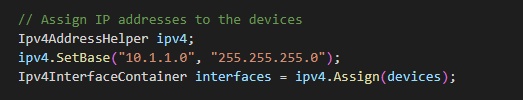
mobility.Install(nodes);

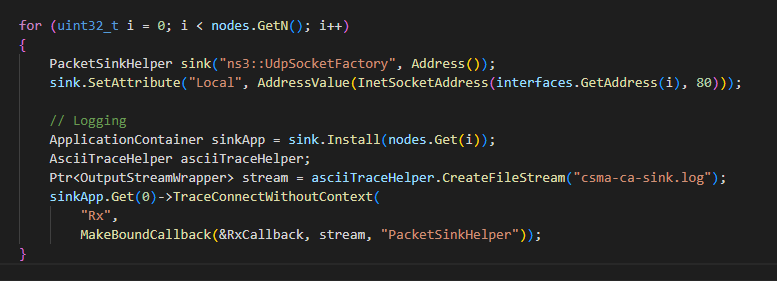
Then we create an internet stack helper and install it on the nodes using the Install method of the InternetStackHelper class:

InternetStackHelper internet;

internet.Install(nodes);

After that we assign IP addresses to the devices using the Ipv4AddressHelper class and create a UDP sink application on each node using the PacketSinkHelper class and set its local address using the SetAttribute method:





We create an OnOff application on each node using the OnOffHelper class (with OnTime when ready to send package, and OffTime when suspend), set its attributes, and install it on the node using the Install method, before enable pcap tracing and run the simulator : 

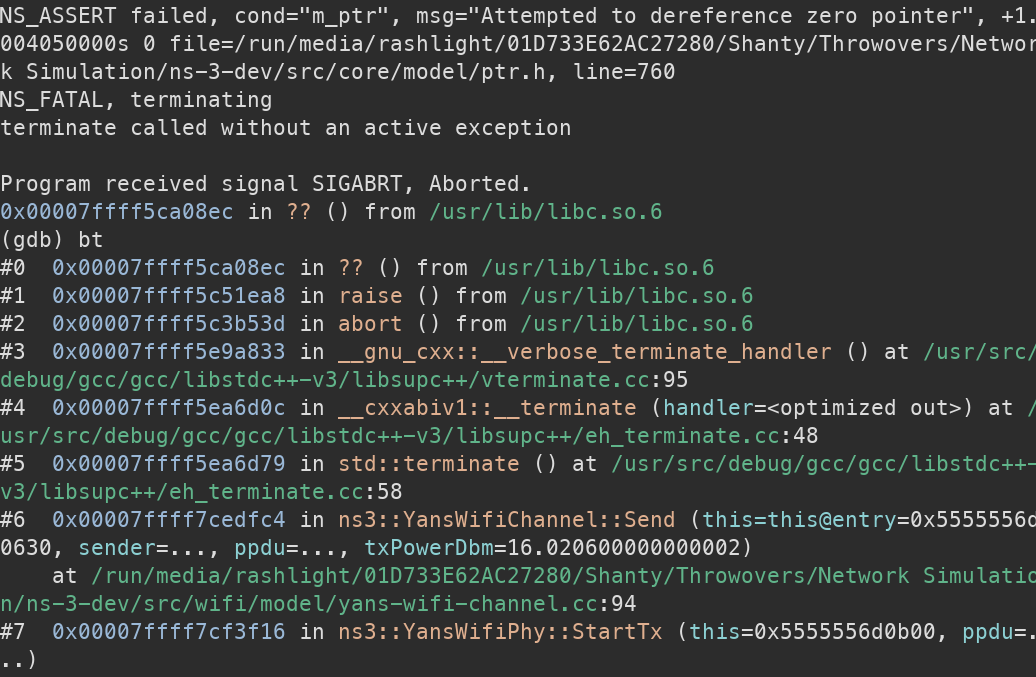
1. **Result**

After running the simulation, we collected some data to evaluate the performance of the CSMA/CA protocol without RTS/CTS scheme as the number of nodes within a communication range increases. We used the OnOffHelper and PacketSinkHelper applications to collect data on the amount of data transmitted and received by each node.

The OnOffHelper and PacketSinkHelper applications write their output to two files: csma-ca-addi.log and csma-ca-sink.log, respectively. We analyzed these files to determine the amount of data transmitted and received by each node over the course of the simulation.

In csma-ca-addi.log, we found that the total amount of data transmitted by each node using the OnOffHelper application ranged from 1.00051 to 9.99994. This data was collected over the course of 10 seconds, during which the nodes were transmitting packets periodically. As the number of nodes increases from 2 to 30, we expect the average packet latency to increase due to increased contention for the wireless medium

We were unable to collect data using the pcap tracing method, as the pcap files were not generated due to an unknown issue with ns-3.



The code failed to run properly due to an internal error in the NS-3 library, which caused the program to unable to get the WifiPhyHelper, which we have installed from the beginning

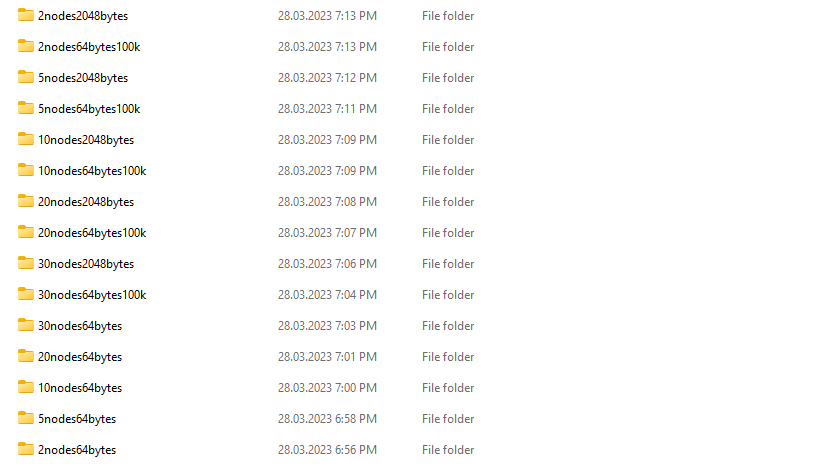
**Final result** : After the presentation, we understand the topic clearer, so we modify the code to increase the nodes each time.

But as we increased the node, strangely enough, the result we obtained are exactly the same. One possible explanation is that the nodes are not experiencing significant contention, leading to similar performance. We choose to pin down the problem by trying the two solutions below :

* Analyze the placement of nodes: The random positioning of nodes may not be creating enough contention in the network. We try placing nodes closer together to force more nodes to be in each other's communication range, by decreasing the distance between the nodes from 2-30 to 1.
* Increase traffic load: The current data rate and packet size may not be sufficient to cause significant contention in the network, so we try increasing the data rate or packet size to generate more traffic and increase the chances of contention.

Even after testing all of that, the result still refuse to change. Finally, we take our guess that because we use udp, there is no congestion. That’s maybe what make the result stay the same everytime. Or maybe because there are no config that shows a device’s speed so the devices have unlimited resource by default. In reality, if a device were to receive too many packages, it would become overloaded and drop some of them.

All the result files we store inside the folders below :



We also uploaded all the source code to our github : https://github.com/ktmvbg/network-stimulation-project

Where the name are in the following structure : number of node – package size – data rate (100k means 100 kbps, without specifications it the data rate is 1Mbps by default)