

# Lab Report

Course Title: Computer Networks Lab

Autumn 2024  
Section:7BF

Lab No: 6

**Name of Labwork:** Statistical Analysis of Random Routing Network

Student's ID : C213246  
Date of Performance : 23.10.2024  
Date of Submission : 28.10.2024  
Team Name : ProtocolPros

Marks

:

## Name of Labwork: Statistical Analysis of Random Routing Network.

### 1. Introduction:

In this lab, we aim to design and evaluate a random routing network where packets travel from a source to a destination through a series of nodes. Each packet counts the number of hops it takes to reach the destination, providing data to analyze the network's efficiency. Statistical measures like mean and standard deviation of hop counts allow for evaluating the network's randomness and consistency. This study is useful in understanding how packets traverse dynamically routed networks, which is essential for designing efficient communication systems.

### 2. Constructing Network(NED):

The network is constructed using a .ned file where we define the structure and connectivity of nodes. In this case, the network, lab\_random\_routing, consists of 10 nodes that can forward messages randomly to neighboring nodes. The connectivity is implemented using DelayChannel, which simulates network delay. The parameters are set to allow for adjustable packet forwarding.

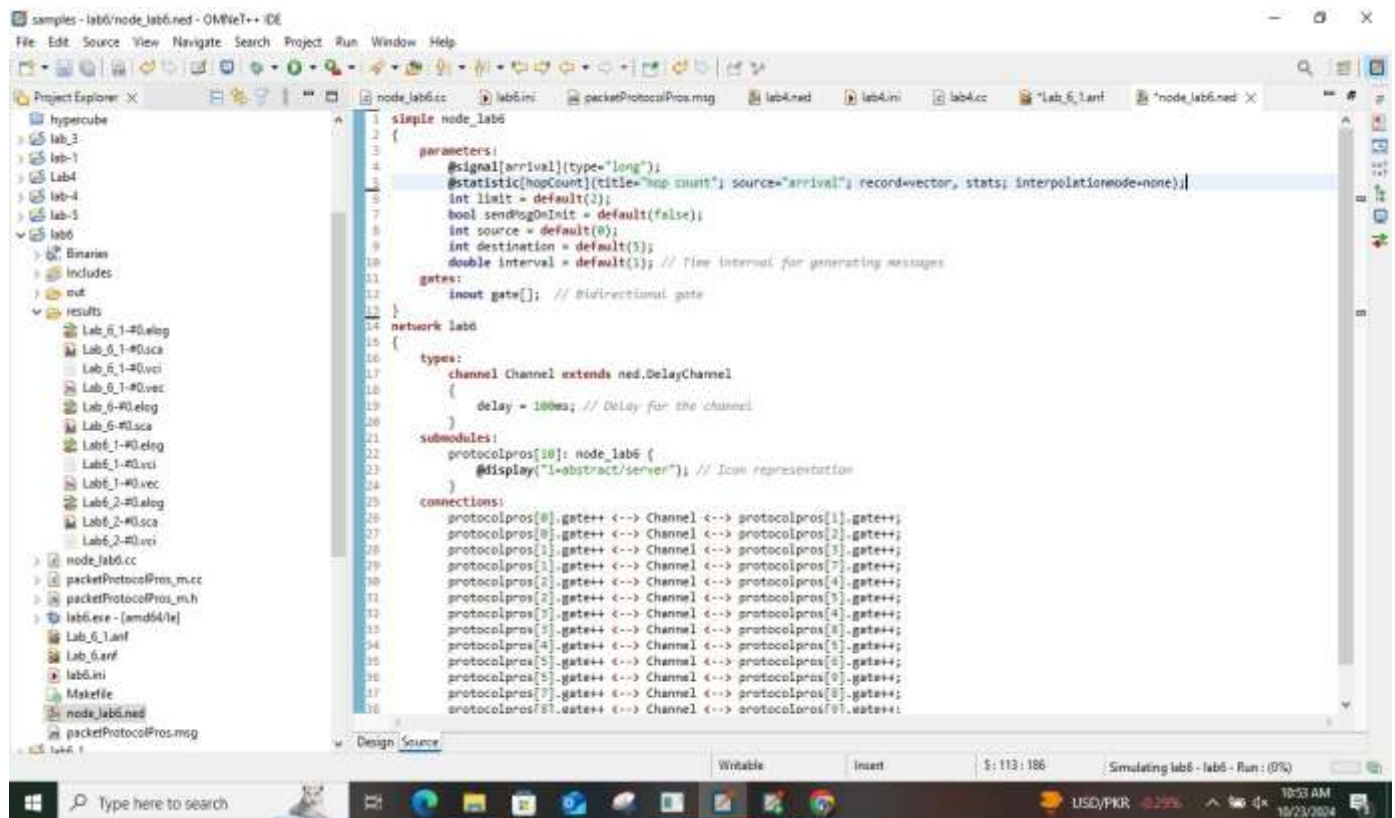
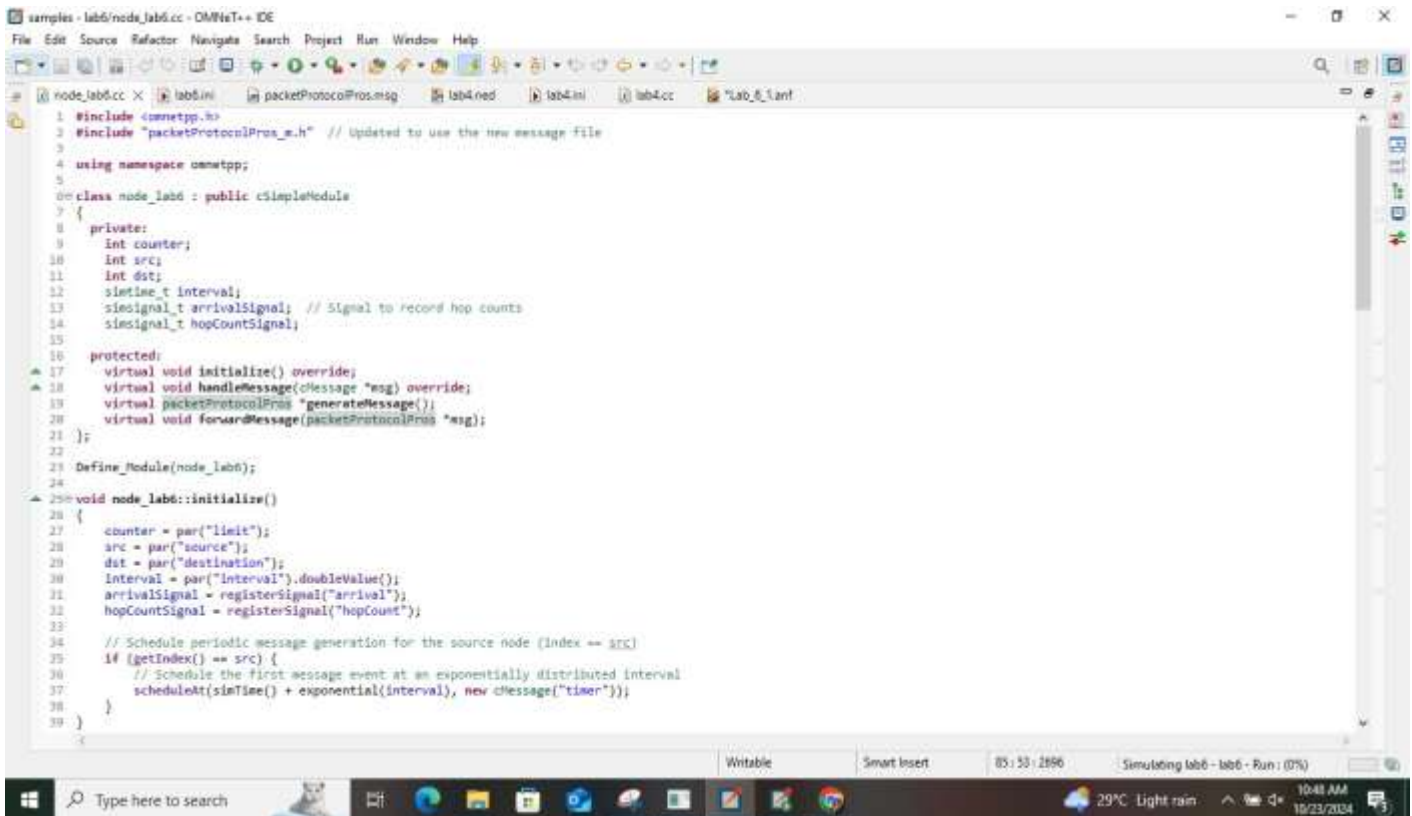


Figure 1: Laptop view of NED file

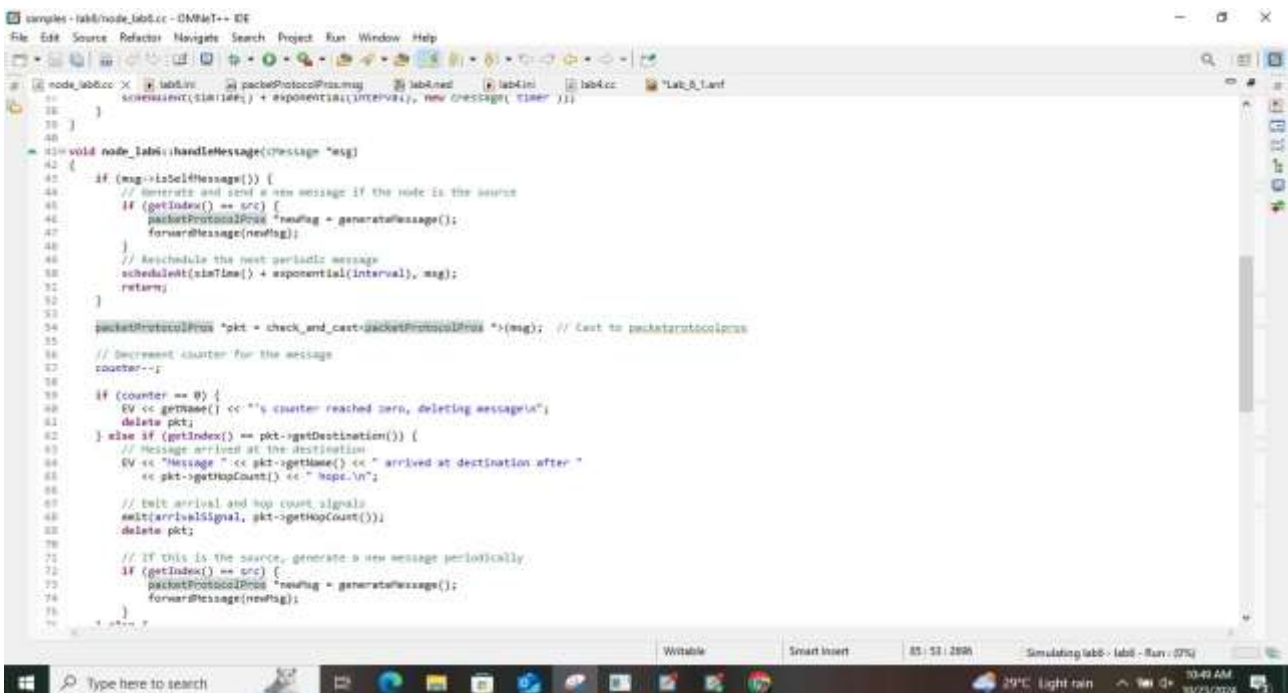
### 3. Building Module(C++ file):

Each node will forward the packet randomly to one of its neighbors, while increasing the hop count at each step.



```
1 #include <omnetpp.h>
2 #include "packetProtocolProc.h" // Updated to use the new message file
3
4 using namespace omnetpp;
5
6 class node_lab6 : public cSimpleModule
7 {
8 private:
9     int counter;
10    int src;
11    int dst;
12    simtime_t interval;
13    simsignal_t arrivalSignal; // Signal to record hop counts
14    simsignal_t hopCountSignal;
15
16 protected:
17    virtual void initialize() override;
18    virtual void handleMessage(cMessage *msg) override;
19    virtual packetProtocolProc *generateMessage();
20    virtual void forwardMessage(packetProtocolProc *msg);
21 };
22
23 Define_Module(node_lab6);
24
25 void node_lab6::initialize()
26 {
27     counter = par("limit");
28     src = par("source");
29     dst = par("destination");
30     interval = par("interval").doubleValue();
31     arrivalSignal = registerSignal("arrival");
32     hopCountSignal = registerSignal("hopCount");
33
34     // Schedule periodic message generation for the source node (Index == src)
35     if (getIndex() == src) {
36         // Schedule the first message event at an exponentially distributed interval
37         scheduleAt(simTime() + exponential(interval), new cMessage("timer"));
38     }
39 }
```

Figure 2.1: Laptop view of cc file



```
42 void node_lab6::handleMessage(cMessage *msg)
43 {
44     if (msg->isSelfMessage()) {
45         // Generate and send a new message if the node is the source
46         if (getIndex() == src) {
47             packetProtocolProc *newMsg = generateMessage();
48             forwardMessage(newMsg);
49         }
50         // Reschedule the next periodic message
51         scheduleAt(simTime() + exponential(interval), msg);
52         return;
53     }
54     packetProtocolProc *pkt = check_and_cast<packetProtocolProc*>(msg); // Cast to packetProtocolProc
55
56     // Decrement counter for the message
57     counter--;
58
59     if (counter == 0) {
60         EV << getName() << " counter reached zero, deleting message\n";
61         delete pkt;
62     } else if (getIndex() == pkt->getDestination()) {
63         // Message arrived at the destination
64         EV << "Message " << pkt->getName() << " arrived at destination after "
65             << pkt->getHopCount() << " hops.\n";
66
67         // Emit arrival and hop count signals
68         emit(arrivalSignal, pkt->getHopCount());
69         delete pkt;
70
71         // If this is the source, generate a new message periodically
72         if (getIndex() == src) {
73             packetProtocolProc *newMsg = generateMessage();
74             forwardMessage(newMsg);
75         }
76     }
77 }
```

Figure 2.2: Laptop view of cc file

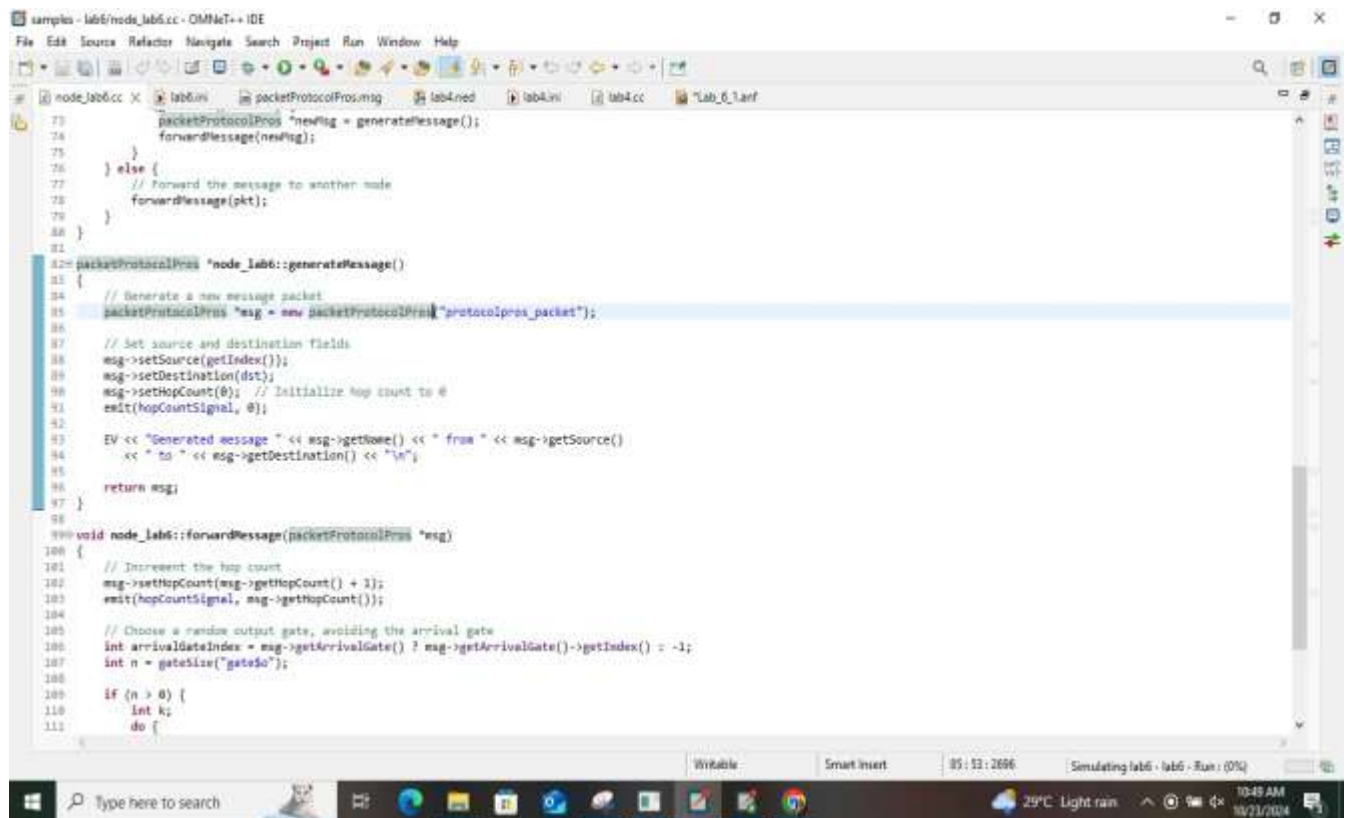


Figure 2.3: Laptop view of cc file

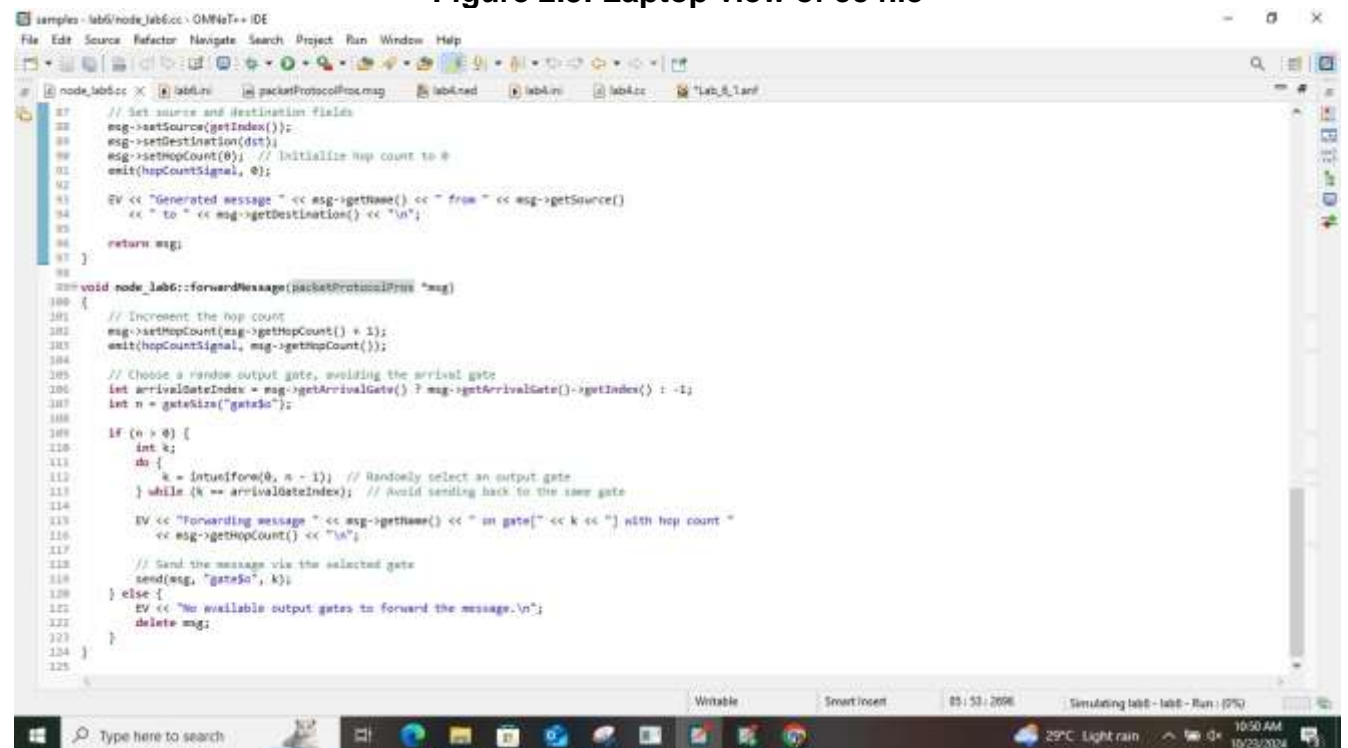
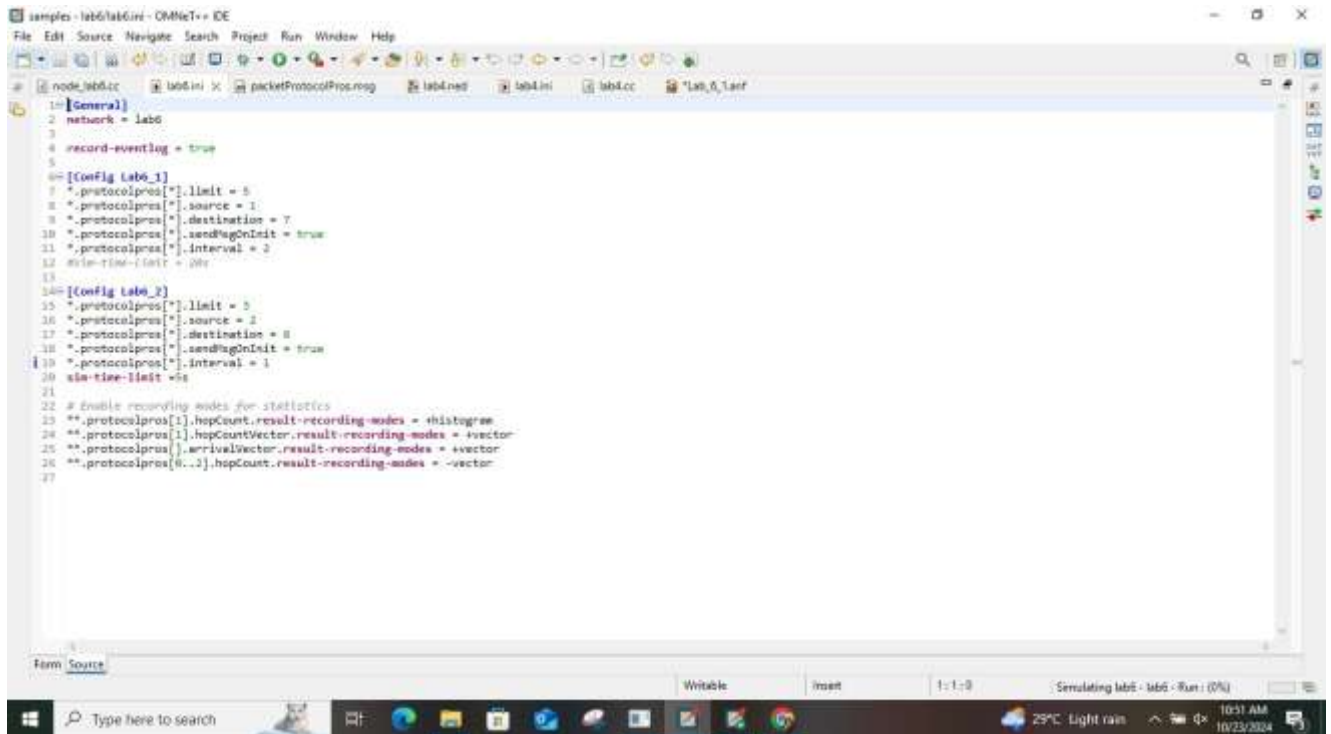


Figure 2.4: Laptop view of cc file

#### 4. Initializing simulation(ini file):



The screenshot shows the OMNeT++ IDE interface with the 'lab6.ini' file open. The file contains configuration for a network simulation, including general settings, protocol parameters for two nodes (Lab6\_1 and Lab6\_2), and recording modes for statistics. The code is as follows:

```
1=[General]
2network = lab6
3
4record-eventlog = true
5
6[[Config Lab6_1]]
7*.protocolpro[*].limit = 5
8*.protocolpro[*].source = 1
9*.protocolpro[*].destination = 7
10*.protocolpro[*].sendMsgOnInit = true
11*.protocolpro[*].interval = 2
12*.time-limit = 20s
13
14[[Config Lab6_2]]
15*.protocolpro[*].limit = 5
16*.protocolpro[*].source = 2
17*.protocolpro[*].destination = 8
18*.protocolpro[*].sendMsgOnInit = true
19*.protocolpro[*].interval = 1
20*.time-limit = 1s
21
22# Enable recording modes for statistics
23**protocolpro[1].hopCount.result-recording-modes = +histogram
24**protocolpro[1].hopCountVector.result-recording-modes = +vector
25**protocolpro[*].arrivalVector.result-recording-modes = +vector
26**protocolpro[0..1].hopCount.result-recording-modes = -vector
27
```

The IDE window title is 'samples - lab6/lab6ini - OMNeT++ IDE'. The bottom status bar indicates 'Simulating lab5 - lab6 - Run : (0%)'.

Figure 3: Laptop view of ini file

## 5. Msg file:

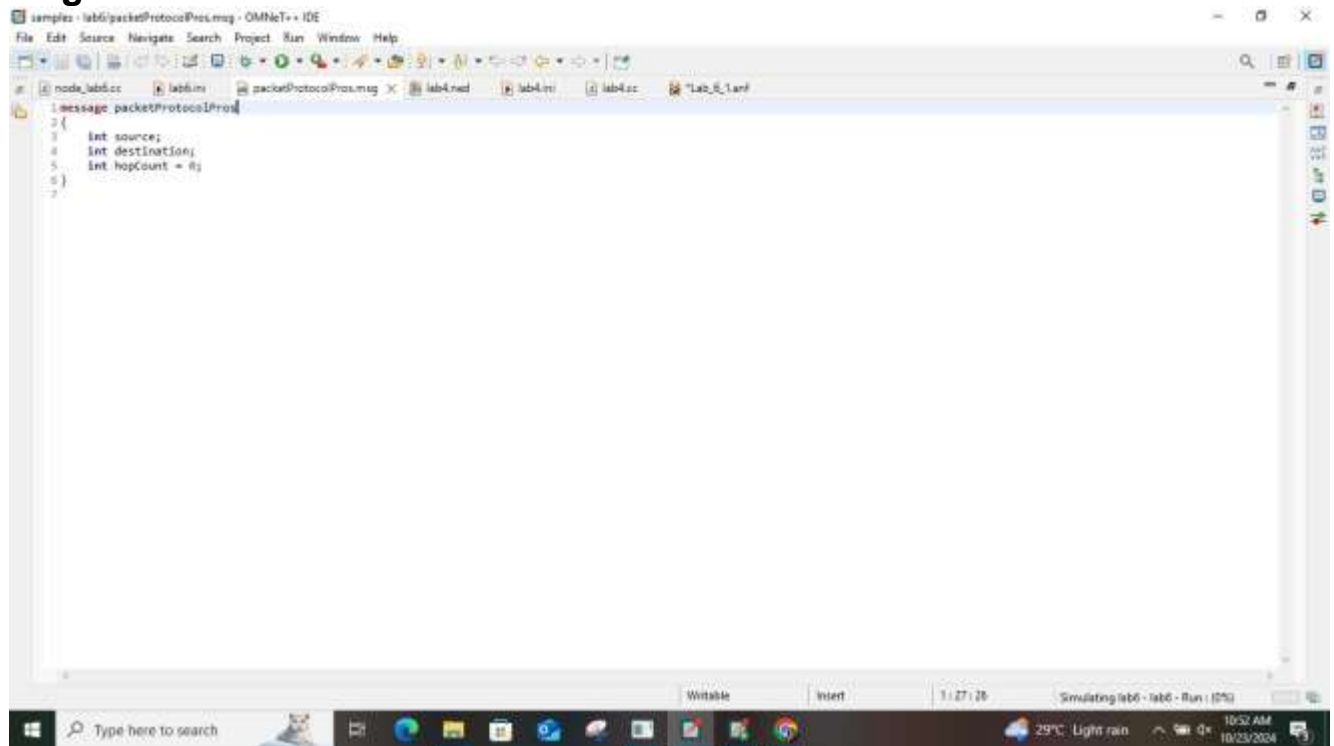


Figure 4: Laptop view of msg file

## 6. Experiment:

To examine the random routing efficiency in a 10-node network, the experiment involves the following steps:

1. **Setup Configuration:** The simulation is initialized with router\_PP[0] as the source node and router\_PP[9] as the destination node. The sendMsgOnInit parameter is enabled for the source node to start generating packets immediately, which then travel through the network via random forwarding.
2. **Data Collection:** Each packet tracks the number of hops it takes to reach the destination. These hop counts are recorded for statistical analysis, providing a data set to measure network efficiency.
3. **Run Multiple Configurations:** The simulation is repeated with varying values for parameters like delay, network size, and message frequency, to observe any effects on hop count distribution.

## 7. Result and Analysis:



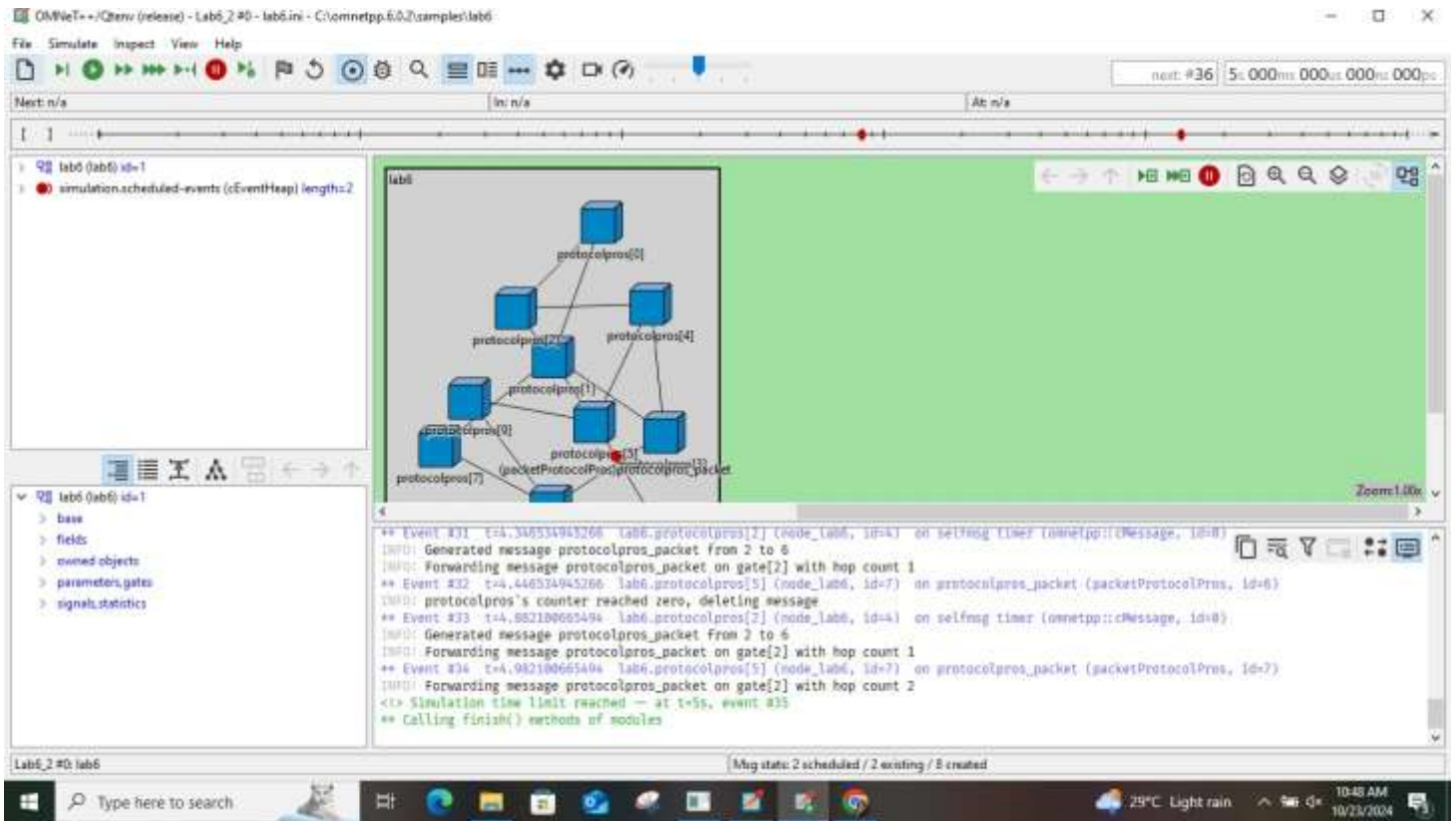


Figure 5.1: Laptop view of simulation and result

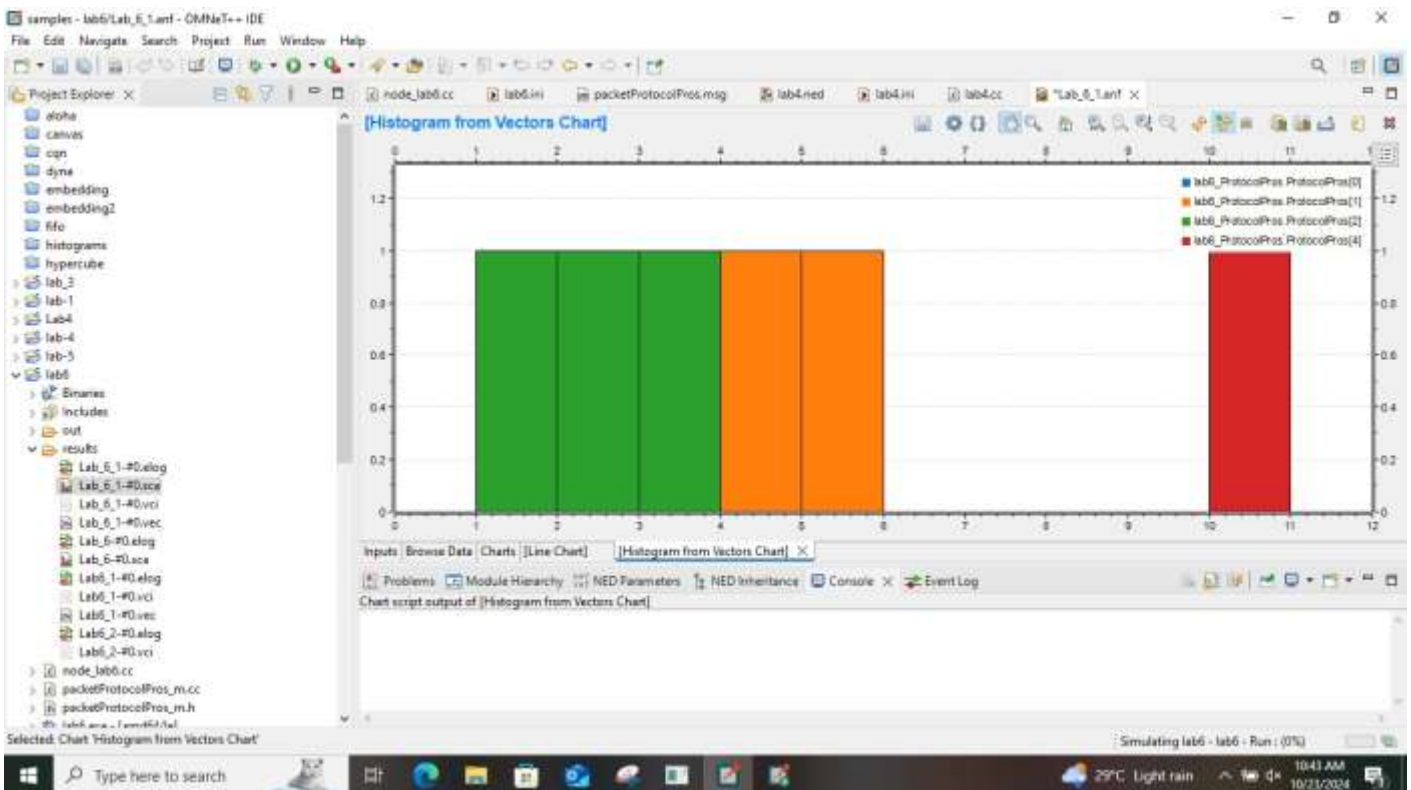


Figure 5.2: Laptop view of Bar Chart

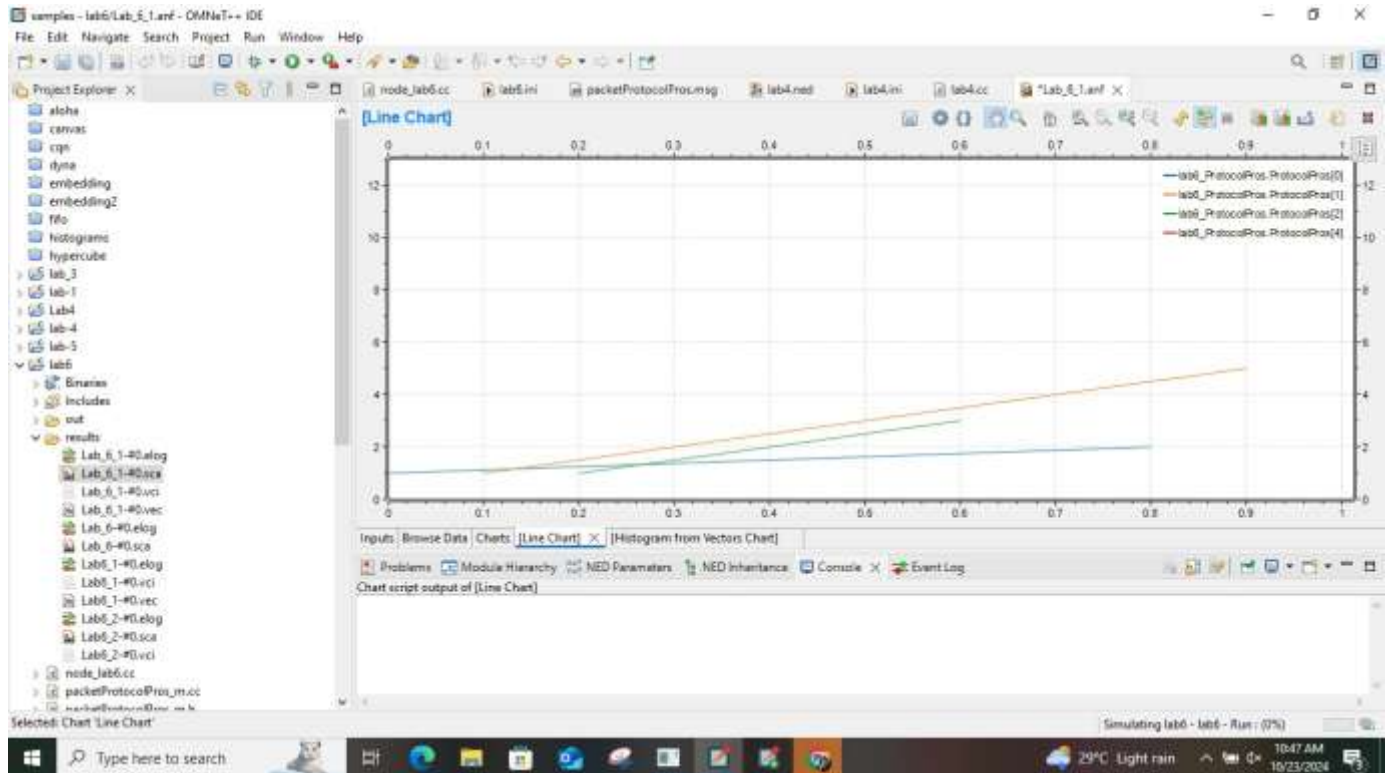


Figure 5.3: Laptop view of Line chart (Mean Hop Count)

## Result and Analysis:

The hop count data is analyzed to understand how efficiently packets are routed through the network. Using Python, we calculate the mean hop count to represent the average routing efficiency and the standard deviation to show the variability.

### Observations:

- **Mean Hop Count:** Provides the average hops taken, indicating general routing efficiency.
- **Standard Deviation:** Reflects the variability in hop counts, showing the network's randomness.
- **Histogram:** Visualizes the frequency distribution of hop counts.

A lower mean hop count indicates more efficient routing, while a high standard deviation suggests more variability in routing paths, consistent with random routing.

## 8. Conclusion:

The random routing network demonstrated a range of hop counts, reflecting the inherent randomness in node selection for message forwarding. This behavior results in a distribution of hop counts, where packets may take different paths and thus varying hops to reach the destination. This experiment illustrates the variability of random routing, with implications for real-world applications that require dynamic routing methods for packet delivery across a network.