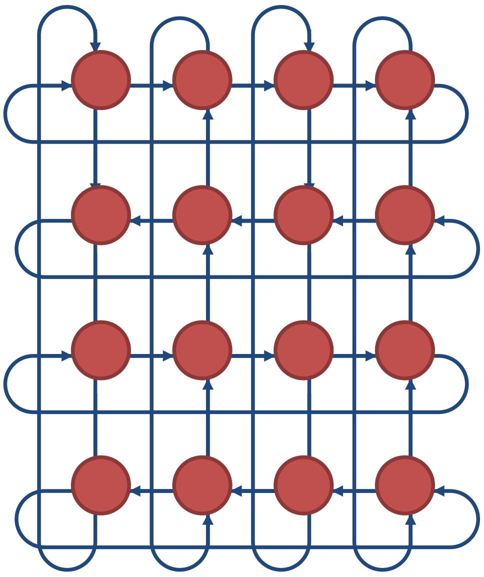
Network Simulation

Laboratory 1 - “The Manhattan Street Network”



In this lab, you will implement a topology known as MSN, Manhattan Street Network (see figure on the right). In this topology, nodes are laid out in a grid-like fashion and each node is connected to other nodes through 4 *unidirectional* links, two outgoing and two incoming. The direction of transmission of messages in each “row” of the grid is inverted with respect to adjacent rows. Likewise, the direction of transmission of messages in each “column” of the grid is inverted with respect to adjacent columns. Another way of looking at this topology is to view it as a mesh of horizontal and vertical *counter-rotating rings*.

A characteristic of the MSN is that the average distance between nodes (i.e., the average number of hops between any two nodes with a shortest-path routing) is of the order of the square root of the total number of nodes in the topology.

Modify the OMNET topology in the .ned file at Step 15 of the Tutorial so as to create a 25-node MSN topology (i.e., 5 rows x 5 columns). Use 10ms delay channels for each connection. Hint: use ‘for’ loops to set up the topology

LAB TASK 1.1: Hop count with random routing

Run the simulation performing random routing in Step 15 of the Tutorial, using the same C++ file, until any one node has generated 10000*G/10 messages (where G is your Team number), then let the simulation stop and collect the statistics.

Report in a table, for each node, the average number of hops needed to deliver a message sent from that node in the following three cases:

A. All links of the topology are operational

B. Numbering the nodes from 0 to 24 in horizontal order and left to right, you disable the links connecting node G to its predecessor and its successor in the same column (again, G is your team number)

C. You disable all the links forming the vertical ring in the column where node G is.

What differences are there between the three cases and how do you justify them?

LAB TASK 1.2: Message delivery delay

Define two types of delay channels: shortChannel and longChannel, with latency, respectively, of 10ms and *L* ms. Use the longChannel on every link incoming and outgoing from node G, while shortChannel is used everywhere else.

For each of the following values of *t={10, 30, 50, 70, 90},* run 10 simulations with different seeds, stopping each of them as in lab task 1.1, and report in a table, along with a 95% confidence interval, the *average delivery delay* of the messages received by node (G+2), as a function of the parameter *L.*

Consider only case A. where all links are operational.

LAB TASK 1.3: Row/column routing

Identifying every node in the MSN topology using a pairwise address (i,j), where *i* indicates the row and *j* indicates the column, implement a routing that no longer selects the output gate at random but, based on the message destination address implements the following steps:

1. the message is forwarded on the horizontal ring until it intercepts the vertical ring where the destination lays (unless the node is already on the destination vertical ring, in which case skip to step 2.)
2. the message is forwarded on the vertical ring until it is received by the destination

Run the simulations in the same settings of Lab Task 1.2 and report the table for the average delivery delay as you did in Task 1.2.

HINTS

- You can create a new project (preferable), or work in the existing 'tictoc' folder. In any case, keep in mind that OMNET will complain if you have classes and modules by the same name in the same workspace.
- To compute confidence intervals, you can use a spreadsheet
- To run a simulation from the terminal command line, change your directory to your project folder (e.g., `lab1`), then type: '`./lab1 -u Cmdenv omnetpp.ini`'.
- Refer to Section 10.4 "Parameter study" on the Omnet++ Manual to learn how to run several simulations with different seeds and different parameters using a single command line.

submit a written report with:

- The answer to Tasks 1.1, 1.2, 1.3
- The .ini, .ned and C++ files you wrote for each of the tasks. Be sure to exhaustively comment your code.

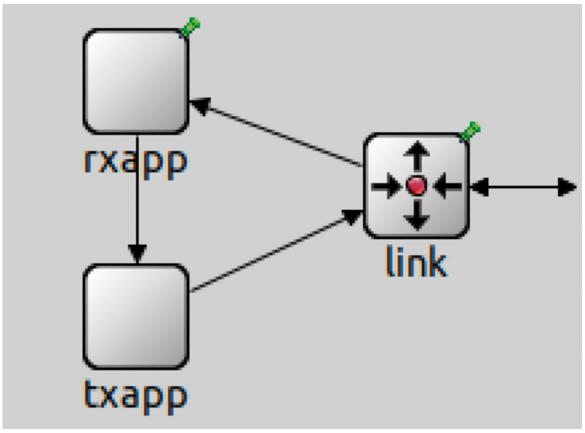
To submit your report, follow the instructions on the portal

Network Simulation

Laboratory 2 - “Adding layers”

In this lab, you will extend the simulation framework you worked on in Lab 1 and implement a two-layer architecture to distinguish between the task of routing messages from the task of generating (at the source) and receiving messages (at the destination). Each node will thus be changed from a single simple module into a compound module including three simple modules:

1. *linkLayer* : its task is simply to determine the next hop that the message has to take on its way to the destination, following whatever routing policy (random or rows/columns) you have implemented in Lab 1. It receives new messages from *txapp* and it delivers messages that have reached their destination to *rxapp.*
2. *rxapp* : it is handed by *linkLayer* the messages that have reached their final destinations. It destroys the messages and, after a time T_{ia} , it tells *txapp* to generate a new message
3. *txapp* : it generates a new message every time it is “woken up” by a message from rxapp.



LAB TASK 2.1: Hop count with random routing

Repeat the same topology and the same routing of Lab Task 1.3 and collect the same statistics, setting T_{ia}=0. You should obtain very similar results. This is a sanity check to make sure you have split the functionalities in a correct fashion.

LAB TASK 2.2: A stop-and-wait protocol

Set to 0.1 the probability of discarding a message at the *linkLayer* module. Implement a stop-and-wait protocol (using ACKs) to recover a lost message and retransmit it.

Computing the delivery delay as the time needed to receive a message *including* the time taken up by its retransmission(s), collected the same statistics as Lab Task 2.1 (still using T_{ia}=0 for now).

HINTS

- Use `getParentModule()` to access parameters and data of the compound module (such as its index number) from the c++ code of one of its simple modules.

submit a written report with:

- The answer to Tasks 2.1 and 2.2
- The .ini, .ned and C++ files you wrote for each of the tasks. Be sure to exhaustively comment your code.