Exercise 1

Open OMNeT++, create a new project using the "Tictoc example" as a basis. Explore the resulting project structure.

1. How is the "*Tic talks first*" behavior managed?
2. Run a simulation. *[Tip 1]*
3. Create a configuration where "*Toc talks first*" and run it. *[tip 2]*
4. Create a configuration where "*Both Tic and Toc talk first*" and run it. Who is actually talking first? Why?
5. Create a variable *counter* within each node and print to log its value upon each message arrival. *[tip 3]*
6. Model the "*processing time*" of a node with a timer. The value of the timer for **tic** has to be randomly generated from C++, while for **toc** it has to be randomly generated from NED. Print with EV the value used for the timer in both cases. Do you see any difference?
7. Selectively enable/disable the "*processing time*" feature via NED. Configure it through the .ini file.

Exercise 2

Create, in the src folder, a new simple module **Hub** with the following characteristics

- It has two *gate vectors*, one for input and one for output. *[Tip 4]*
- Whenever it receives a packet from in[i], forwards it through out[i+1]. (manage index wrap around)

Create, in the simulation folder, a network **TicHubToc**, containing a tic, a toc and a hub node. Connect tic and toc to hub. Connect them so as to have `in[0]`/`out[0]` for tic and `in[1]`/`out[1]` for toc. Do not specify any connection parameter. *[Tip 5]*

1. Create a new configuration using the new network and set the simulation duration to 5 seconds. Does it stop? *[Tip 6]*
2. Apply a delay of 100 ms to each connection. Does the simulation stop now? Why?
3. Create a @statistic *countMsg* with type "`sum`" for each module. Use *countMsg* to count the number of messages seen by each node. Print a chart for *countMsg*, can you guess if it is correct? Is it uniformly distributed between tic and toc? Try increasing the duration of the simulation. *[Tip 7]*
4. Measure (with a @statistic) the time between two consecutive message transmissions for tic and toc. Call it *responseTime*. *[Tip 8]*
5. Define a `bool` parameter *randHub* for the hub. If it is true, let the hub choose the output interface randomly **every time** it receives a packet. Set the simulation duration to 50 seconds. Are the values of *responseTime* similar? Increase the number of repetitions. Any change? *[Tip 9]*

Exercise 3: Producer-consumer

Consider the system depicted in Figure 1. The two nodes A and B are consumers which periodically, with period t_A and t_B respectively, request services to a central node C. The node C forwards their requests to the first available producer within a pool. The pool of producers is composed of two nodes, namely 1 and 0, which receive the service requests from node C and produce a response with a reaction time r_0 and r_1 respectively. The response will be send to a dispatcher node D, that will forward it back to the proper consumer node.

The central node S serves the requests from nodes A and B according to their order of arrival. The producers are selected in increasing order of ID (node 1 has ID=1, node 2 has ID=1)

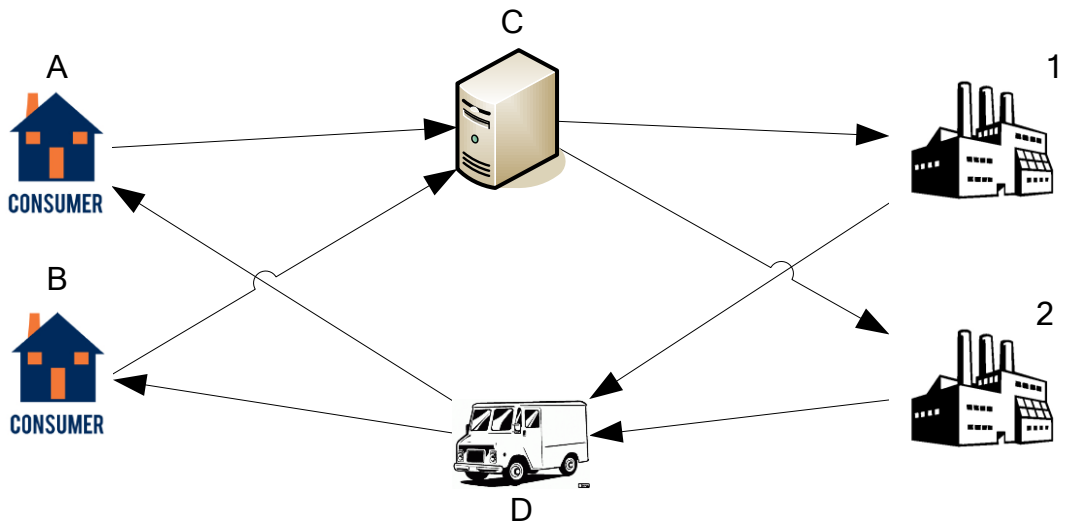

Figure 1 - Producer-consumer scenario

Tips [spoiler alert]

1. Open the omnetpp.ini file then hit the “white and green play button”
2. Hit the small arrow on the right of the “white and green play button”, then select a different *config name*
3. You can print messages into the log window using `EV` in the same you would use `cout`.
4. You can create the vector of gates using the following syntax: `input in[]; output out[];`
5. You can connect array of gates with the following syntax: `tic.out --> hub.in++;`
6. Add the line `sim-time-limit = 5s` to the .ini file
7. Open the simulation/results/ folder then double click a .sca file
8. You can obtain the current time with the function `simTime()`.
9. Set the number of repetition with `repeat = 5` and set the rng seed accordingly with `seed-set=${repetition}`