# CMPUT 313 - Lab #2 (6%)

Reliable Data Transfer Protocols

(first draft)

**Due: Friday, November 6, 2015, 09:00 PM**
**(electronic submission)**

## Objectives

The programming assignment is intended to give you hands-on experience with the event-driven programming style of *cnet* and its use in implementing and analyzing simple reliable data transfer protocols.

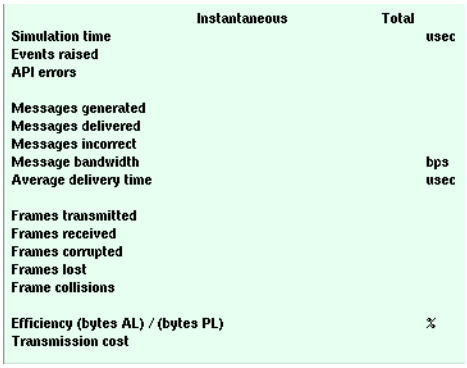
## Part 1: Performance of a STOP-AND-WAIT Protocol

In this part you will investigate the average throughput of the stop-and-wait protocol implemented in *cnet* . The four network topologies defined below are used for this purpose. For each network, the investigation will be done using both mathematical modelling, and *cnet* simulation.

To start, edit file `STOPANDWAIT` to set the link bandwidth to 56 Kbps, the one way propagation delay between the two hosts to 2.5 sec, and the `messagerate` to 1 sec. Construct the topology files of the four networks by making four copies (named `NETa`, `NETb`, `NETc`, and `NETd`) of the `STOPANDWAIT` file. Modify the copies to satisfy the following requirements.

- In `NETa`, the length of each application layer message is 2000 bytes, and the physical layer does not corrupt or lose frames.
- In `NETb`, the length of an application layer message is a uniform random variable with value between 2000 bytes and the default maximum length, and the physical layer does not corrupt or lose frames.
- In `NETc`, the length of each application layer message is 2000 bytes, and the physical layer does not lose frames, but corrupts only data frames (that is, ACK frames are delivered reliably) with probability 0.5 each.
- In `NETd`, the length of each application layer message is 2000 bytes, and the physical layer does not lose frames, but corrupts both data and ACK frames with probability 0.5 each.

For each network, run the protocol for about $T_{sim} = 300$ sec. of simulation time, and record all *per-node* statistics and *global* statistics produced by *cnet* (illustrated below).


In your report:

1. Present the obtained per-node statistics for each host, and the global statistics. You may include snapshots of the simulator outputs. (Note: the global statistics can also be obtained by using ``cnet -W -q -T -e 300sec -s *topology_file*").

2. **Mathematical Modelling:** Using a mathematical approach similar to what we discussed in the classroom, develop a suitable mathematical model to estimate the protocol's average throughput for the given network. Express your answer in units of **messages/sec.** (i.e., the average number of successfully transmitted application messages per second), and **KBytes/sec.** (i.e., the average number of successfully transmitted KBytes of application layer traffic per second). Explain.

3. **Analysis of *cnet* Simulation Results:** Using the obtained simulation results, obtain a numerical estimate of the protocol's average throughput both in units of **messages/sec.** and **KBytes/sec.** (as explained above). Write the expressions used in your answer and identify *cnet* quantities used in each expression.

4. Discuss the obtained results. In particular, comment on the agreement (or lack thereof) between the analytical results and the simulation results obtained in parts 2 and 3 above.

## Part 2: STOP-AND-WAIT Protocol on Path Networks

In this part you are asked to extend the protocol in `stopandwait.c` to provide reliable data transmission between the two end hosts of a *path* network where all internal hosts are routers. Store your protocol in a new file called `saw.c`. For simplicity, assume that the network has only one traffic flow corresponding to the application layer messages generated by the leftmost host and destined to the rightmost host of the path. At any instant, each router runs a stop-and-wait protocol for managing the incoming traffic from its left neighbour, and another stop-and-wait protocol for managing the outgoing traffic to its right neighbour. That is, the stop-and-wait protocol runs on every link, and **not directly between the two end hosts**.

Each router has a buffer for storing frames containing application messages that pass through the router. Again, for simplicity, assume that such buffer has one slot that can store one frame containing an application message. If a frame containing an application message arrives to a router and finds the slot occupied, the arrived frame is discarded (and should be retransmitted).

Test your protocol using a 5-node topology file, called `SAW`, that can lose and corrupt frames on each link.

## Deliverables

1. All submitted programs should compile and run on the lab machines.

2. Typeset a project report (two to five pages either in text, HTML, or PDF) with the following (minimal set of) sections:

   - **Objectives:** describe the project objectives and value from your point of view
   - **Part 1:** for each of the specified four network topologies, present the specified statistics, the required analysis, and comments on the obtained results, as explained in Part 1.
   - **Part 2 (Design Overview):** highlight in point-form the important ideas and features of your protocol. Write a suitable high level description of your protocol.
   - **Project Status:** describe the status of your project; mention difficulties encountered in the implementation
   - **Acknowledgments:** acknowledge sources of assistance

3. Combine the project report with files `NETa` through `NETd` (for Part 1), `saw.c`, and SAW (for Part 2) into a single tar archive `'submit.tar'`.

4. Upload your tar archive using the **Lab #2 submission/feedback** link on the course's web page. Late submission (through the above link) is available for 24 hours for a penalty of 10%.

5. It is strongly suggested that you **submit early and submit often**. Only your **last successful submission** will be used for grading.

## Marking

Roughly speaking, the breakdown of marks is as follows:

**40%** : Part 1: correctness of the used topology files, mathematical analysis, analysis of the obtained simulation results, and the written discussions

**45%** : quality of the designed protocol: error-free compilation and execution, correctness, simplicity and elegance of the design, and clarity of the presentation

**05%** : ease of reading and understanding the program

**10%** : quality of the project report (results, justifications, discussions, etc.)

---