CMPUT 313 - Lab #1 (6%)

The CNET Simulation Environment
(first draft)

Due: Friday, October 2, 2015, 09:00 PM
(electronic submission)

Objectives

The programming assignment is intended to give more experience with the *cnet* network simulator API and its use in developing a simple protocol.

Part 1: The CNET API

This part requires that you carefully examine the C code in the following files: the *cnet* standard header file `cnet.h`, the *cnet* standard support header file `cnetsupport.h` (make local copies of the two files from directory `$CNETPATH`; you may need to refer to them frequently in the future), the `ticktock` project files, the `stopandwait` project files, and *cnet*'s FAQs. Answer the following questions.

1. File `cnet.h` defines a number of basic data types that are neither C structures nor enumerated data types. List all such types. Which types are not equivalent to the `int` data type on the lab machines?

2. How can one obtain a node's current simulation time in *cnet* (the time advanced by the occurrence of events)? How to print such value using a *printf* statement? (Hint: check the FAQs as well as `cnet.h`.)

   Remarks: The simulation time is typically different from the wall-clock time. For example, if the *run-speed* parameter in the GUI is set to a value other than *normal*, the simulation time may run either faster or slower than the wall-clock time. Also, one should not assume that the simulation time is synchronized between all nodes by default. Some nodes may have pending undelivered events, and thus their simulation time may be lagging others. The design of the simulator, however, ensures that each node encounters a valid sequence of timed events.

3. Does `cnet.h` specify the content and format of the messages generated by the application layer? If yes, describe the contents and layout of a typical message.

   Remark: If the answer is no then one may deduce that the contents of the messages generated by the default application layer are not intended to be accessed by the developed protocols.

4. What does the `CHECK` macro do when the enclosed function call returns an error value (typically -1)? Explain how the information mentioned under *API FOR ERROR HANDLING* in `cnet.h` can be used to report an API error to the user (using some easy to understand text message) while allowing the simulator to continue execution.

Remark: You may test the effect of the macro by checking, for example, its behaviour when `CNET_write_physical` is called with a non-existing link.

5. In the `stopandwait.c` protocol, can `FRAME_SIZE(f)` be different from `sizeof(f)`? Explain.

6. Does the `stopandwait.c` protocol detect corrupted frames? If yes, explain how this feature is programmed (when appropriate, give C code fragments to illustrate the steps involved).

7. How does the `stopandwait.c` protocol stop a timer when an ACK is received?

Part 2: A Simple Data Transfer Protocol

In this part, you will use topology files `WAN-1`, `LAN-1`, and `WLAN-1` available for download from the course's web site. Each file defines a network composed of a server node (assigned address 200), and 5 clients $c1$ through $c5$ (assigned addresses 101 to 105). Each file executes a protocol in file `lab1.c` that you are asked to develop and implement.

The required protocol behaves as follows. When executed by a client node, the protocol repeatedly reads an application layer message that is destined to the server node, and sends the message to the server. When executed by the server node, the protocol reads any incoming message and delivers it to the application layer. Thus, the protocol implements many-to-one data transfers from the client nodes to the server. Here are more remarks on the required program.

– For convenience of developing the program, start with `protocol.c` (in the CNET's examples directory), and introduce modifications as required.

– Client nodes should not generate application messages to each other.

– Your program should allow physical layer errors (controlled by the `probframecorrupt` parameter) to affect the transmitted frames. Note that if `probframecorrupt` is set to a positive integer in a topology file then some application layer messages will be corrupted. The corrupted messages will be detected and dropped by function `CNET_write_application`. Moreover, once a message with a particular sequence number is dropped, the application layer at the destination node will refuse to accept subsequent messages (unless CNET is run with the `-Q` option).

– Your protocol is **not** required to recover from physical layer errors if such errors occur. So, retransmissions because of errors are **not** required.

– Topology files `LAN-1` and `WLAN-1` utilize local area networks. In such networks, we typically assign a 6-byte link layer address to each network interface card (NIC); such address is called `nicaddr` in CNET. If used properly, `nicaddr` allows an interface card to filter unwanted traffic. The use of NIC addresses, however, is not required by CNET. As you may note, for simplicity, the above files omit them.

– A call to `CNET_write_application` may fail for various reasons that are likely to arise when developing your program. For example, the message in the call may be corrupt (because of the physical layer), out of sequence (if a previous message has been dropped), or carries a destination address different from that of the current node. To avoid unwanted program termination, do **not** call the function from within the `CHECK (...)` macro.

Part 2 Questions

After implementing your `lab1.c` program, investigate the following aspects and include your findings in a program report.

1. For the network in `WAN-1`: how many application layer messages are successfully received by the server in a 100 sec. run (e.g., using `CNET -W -T -e 100s -s WAN-1`)? Does this number change significantly if `wan-bandwidth` is doubled? Explain.

2. For the network in `LAN-1`: does tracing your program (e.g., using the `-t` option) reveal errors? If yes, what client or server errors are being reported?

3. Repeat the above question for the network in `WLAN-1`.

Deliverables

1. Typeset a solution file `'answers'` (text, HTML, or PDF) containing your answers to the questions in Part 1, and a program report (one or two pages) for the program in Part 2. The program report should contain the following (minimal set of) sections:

   – **Design Overview:** highlight in point-form the important features of your design
   – **Program Status:** describe the status of your program; mention difficulties encountered in the implementation
   – **Findings:** report your findings and answers to Part 2 Questions
   – **Acknowledgments:** acknowledge sources of assistance

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

3. Combine files `answers`, and `lab1.c` into a single tar archive `'submit.tar'`.

4. Upload your tar archive using the **Lab #1 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 primary submission** will be used for grading.

Marking

Roughly speaking, the breakdown of marks is as follows:

35% : Part 1 questions

50% : correctness, testing, and results of Part 2 program

15% : quality of the program report (results, justifications, discussions, etc.)