LABORATORY MANUAL

CSC302 : Net-Centric Computing

Assignment 1

SESSION 2010/2011
SEMESTER 1
COMPUTER SCIENCE COURSE

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Implementation of a Sliding Window Protocol

1. OBJECTIVE

This assignment aims to enhance students' understanding of the network protocol hierarchy and flow control and error control techniques by implementing a sliding window protocol in a simulated communication network system. The programming language will be *Java*.

2. LABORATORY

Computer Lab 1 (CL1).

3 EQUIPMENT

PC with Windows and Java.

4. THE SIMULATED COMMUNICATION SYSTEM (provided)

The simulated communication system consists of the following two major components:

1. **Network Simulator**
   This component simulates the physical transmission media which connects two communicating virtual machines. The component may be set to operate in one of the four different quality levels of service:

   - Level 0: an error-free transmission media.
   - Level 1: a transmission media which may lose frames.
   - Level 2: a transmission media which may damage frames (i.e., generating checksum-errors).
   - Level 3: a transmission media which may lose and damage frames.

2. **Virtual Machine**
   This component simulates a communicating virtual machine. Internally, it is divided into two sub-components:

   (1) Sliding Window Protocol
   This component implements the sliding window protocol (i.e., the data link layer). In this simulated system, this component cannot work alone, and must interact with the Sliding Window Environment component (to be described next) in order to fetch/deliver a sequence of packets from/to the upper network layer, and to fetch/deliver a sequence of frames from/to the lower physical layer.

   (2) Sliding Window Environment
   This component provides the environment in which the sliding window protocol component is working. Basically, this component implements the following major interfaces:

   - The interface between the data link layer and the network layer. This interface consists of three procedures:
     - `to_network_layer()`, to deliver a packet to the network layer.

- `from_network_layer()`, to fetch a packet from the network layer.
- `enable_network_layer(int creditnr)`, to grant the number of `creditnr` credits to the network layer so that the network layer can generate the number of `creditnr` new packets.

In addition, this component implements the capability of automatically sending/receiving a sequence of packets to/from the data link layer.

- The interface between the data link layer and the physical layer. This interface consists of two procedures:
  - `to_physical_layer()`, to deliver a frame to the physical layer.
  - `from_physical_layer()`, to fetch a frame from the physical layer.

In addition, this component is responsible for interacting with the Network Simulator to transmit/receive frames to/from the underlying transmission media.

- The interface between the data link layer and the underlying event queue. This interface consists of the following procedures:
  - `wait_for_event()`: to wait for the arrival of an event.
  - `generate_acktimeout_event()`: to generate an acknowledgement timeout event.
  - `generate_timeout_event(int seqnr)`: to generate a timeout event for an outstanding frame with the sequence number `seqnr`. Note: When a timeout event is being handled by the sliding window protocol, the protocol variable `oldest_frame` will be automatically set to the sequence number corresponding to the current timeout event.

It should be pointed out that the Network Simulator component is running in one process, and the Virtual Machine component (including both the Sliding Window Protocol and the Sliding Window Environment) is running in another process. To simulate the communication between two virtual machines, two Virtual Machine processes must be executed (see Section 6 of the textbook for how to test and run the program).

5. YOUR TASK

Both the Network Simulator and Sliding Window Environment components have been implemented and supplied. Your task is to implement the Sliding Window Protocol component (i.e., the data link-layer) of the simulated communication system. This component must implement all the features in the sliding window protocol specified in Fig.3-19 of the text book, including:

1. Full-duplex data communication.
2. In-order delivery of packets to the network-layer.
3. **Selective repeat** retransmission strategy.
4. Synchronization with the network-layer by granting credits.
5. Negative acknowledgement.
6. Separate acknowledgment when the reverse traffic is light or none.

Your implementation must be able to withstand quality level 3 of the Network Simulator component.

6. TESTING AND RUNNING THE SYSTEM

To run your code in the simulated communicated system, you should first compile your java code by typing:

```
javac SWP.java
```

Then the following steps should be followed to run the system:

1. First, start the Network Simulator (NetSim) component in one window by typing:

   java NetSim n, where n is the quality level and may take a value 0, 1, 2, or 3.

2. Then, start the first Virtural Machine (VMach) component in another separate window by typing:

   java VMach 1, where 1 is the identifier of this VMach.

3. Finally, start the second Virtural Machine (VMach) component in the third separate window by typing:

   java VMach 2, where 2 is the identifier of this VMach.

To terminate the whole system at the end or middle of the execution, you may simply type control-c in anyone of the three windows.

After each run of the system, two output text files - `receive_file_1.txt` and `receive_file_2.txt` - will be automatically generated, with `receive_file_1.txt` being generated by VMach 1 and containing the texts received from VMach 2, and `receive_file_2.txt` being generated by VMach 2 and containing the texts received from VMach 1. If the sliding window protocol is correctly implemented, the contents in `receive_file_1.txt` should be the same as `send_file_2.txt`, and the contents in `receive_file_2.txt` should be the same as `send_file_1.txt`.

7. GENERAL SUGGESTIONS

1. Read this document and the supplied source files carefully.

2. Read and fully understand the sliding window protocol in Fig.3-18. and relevant sections (particularly Sections 3.3 and 3.4.) of the textbook.

3. While testing your sliding window protocol implementation, first test it while NetSim is running at quality level 0, then at quality levels 1, 2, and 3, one by one.

8. MODE OF WORKING

You may perform this assignment in pairs, selected by yourselves, or individually.

9. DUE DATE AND TIME

Written reports for this assignment are **Due at 5 pm Friday of Week 6.**

.

10. SUBMISSION PROCEDURE

One group needs only one submission. You should submit your work in time. There are penalties for late submissions.

Your submission for assessment should comprise the following:

1. A cover page, clearly showing the code and name of the course (CSC302 Net-Centric Computing), your name and student number, and including a summary of how much of the assignment you have completed and what parts of the assignment were untouched/unfinished. For the completed parts, a summary of your approach should

be included. If you choose to do this assignment in pair, each member of the group must write an independent summary, which should describe the strategies used for implementing the whole system, and clearly explain which part is by you, your partner, or jointly.

The written report must also contain a listing of java source files (with sensible comments), including all Java source files used for completing the implementation of the sliding window protocol.

2. In addition to the written report, please place all java source and class files in designated directories on lab PCs by the due date. Please refer to the instructions in the Computing Lab 1 for details.

11. ASSESSMENT

The assessment of this assignment will be based on your written report and testing of your program. Your assignment will be assessed according to the following criteria:

- correctness,
- completeness,
- documentation - this includes internal and external documentation,
- general quality, and
- testing of the program.

APPENDICES

The following files have been provided:

1. **Source files**:
   Two java source files are provided:

   - `SWP.java`: The skeleton of the Sliding Window Protocol component. Note: this is the only provided java source file that you can change. You may add new java classes in order to fully implement the sliding window protocol.
   - `PFrame.java`: The frame class source file. Note: this java source file is provided for your reference in implementing sliding window protocol, but you should not change anything in this file.

2. **Class files**
   A number of java class files are provided, which implement the Network Simulator component and the Sliding Window Environment component:

   - `NetSim.class`: the main class of the Network Simulator component.
   - `Forwarder.class`: an auxiliary thread class of the NetSim component.
   - `VMach.class`: the main class of the Virtual Machine component.
   - `SWE.class`: the major class of the Sliding Window Environment component.
   - `FrameHanlder.class`: an auxiliary thread class of the SWE component.
   - `NetworkSender.class`: an auxiliary thread class of the SWE component.
   - `NetworkReceiver.class`: an auxiliary thread class of the SWE component.
   - `EventQueue.class`: an auxiliary class of the SWE component.
   - `Packet.class`: an auxiliary class of the SWE component.

- `PacketQueue.class`: an auxiliary class of the SWE component.
- `PEvent.class`: an auxiliary class of the SWE component.
- `PFrame.class`: an auxiliary class of the SWE component.
- `PFrameMsg.class`: an auxiliary class of the SWE component.

3. **Testing text files**
   Two text files are provided for testing purpose. They are used by the Sliding Window Environment component to generate a sequence of packets to be sent to the other communicating machines:

   - `send_file_1.txt`: the file used by VMach 1 to generate a sequence of (text) packets to VMach 2.
   - `send_file_2.txt`: the file used by VMach 2 to generate a sequence of (text) packets to VMach 1.

   You do not need to change these two input files. But if you wish, you are free to modify the contents of these two files to suit your special needs of testing.

You should download all the above files from the edventure course site into your local PC in any directory of your choice. Then, you should update the ``classpath" environment variable of your system (Windows or Unix/Linux) to include the path to the directory containing the downloaded files.