CS 462 – Sliding Window Protocols

Introduction

You are to implement by simulation the Selective Repeat (SR) protocol. The project must be implemented with the C++/C language. You may use the ACE or Boost framework or any available networking APIs.

Remember that we are doing this in software and that you should introduce situational errors to test the correctness of your implementation. This means that you must have the ability to force packets to be out of sequence, lost, damaged, and/or simulate lost acknowledgement packets. Situational errors can be randomly generated or user-specified (described later).

As you progress, you will notice that other minor details not mentioned in this document may be required. This is to simulate a real-world environment where a customer initially provides as much information as possible for you to get started. As you start your design, you have questions. You contact the customer to get more information. It is therefore expected that you visit my office for questions.

Design Issues

There are two types of Internet sockets, stream sockets and datagram sockets. Most of you have used the stream sockets in your Operating Systems class. Stream sockets are reliable sockets, i.e., frames received are always in the order they are sent. For this project, use TCP (not UDP) sockets.

In designing your program, you should discuss the pros and cons of the following design issues:

1. The size of your packet.
2. Frame format, i.e., type of header information you want to keep.
3. Sequence numbers (do not use infinite sequence numbers)
4. Error detection (use the CRC algorithm). Is this a reasonable algorithm to use?
5. The size of your sliding window.
6. The time-out interval (static or dynamically determined).
7. How to implement timeouts.

You will be asked to demo your project. Hence, it is important to capture as much information as possible of your run.

Use the md5 hash utility to check the integrity of the files you have transferred over, i.e., the file at the source must have the same md5 hash value as the one at the destination. To use the md5 hash on our linux machines.

```
$ md5sum random.txt
6b3d28a099f3bd67b57410759ffb439f   random.xt
```

GUI/Input

Design a simple (text-based or graphical) UI for your program. The user should be prompted for either default values (determined by the designer) or user-defined values to be entered. The following, at a minimum, must be prompted for:

(a) Size of packet
(b) Timeout interval (user-specified or ping-calculated)
(c) Size of sliding window
(d) Range of sequence numbers
(d) Situational Errors (none, randomly generated, or user-specified, i.e., drop packets 2, 4, 5, lose acks 11, etc.)

Your output should provide enough information for me when you demo your project. There should be a separate area where all the necessary information is logged and printed on the screen.

The user should be able to specify a server host for the file transfers. At a minimum, you should be able to transfer one file.

File Transfer Phase:

You will be required to open two windows, one for the client and one for the server. Information should be available from both windows that show the protocol in action. The information provided in the client (sender) window should include, but not limited to:

- Packet sequence number of packet sent.
- Packet (sequence number) that timed out
- Packet that was re-transmitted.
- All packets in the current window.
- Acks received.
- Which packets are damaged, .i.e., deliberately trigger a checksum error on the receiver side.

Information to be provided by the receiver:

- Packets (sequence number) received.
- Damaged packet(s) (checksum error should match corresponding damaged packet from sender window).
- Packets in the current receiver window. o Packets (duplicated) that are discarded.
- Ack packets that are sent.
- Frames arriving out of order should be re-sequenced before assembly.

Example of output on the sender side:

You MUST strictly follow the format specifications for both sender and receiver shown below.

Note: Remember relationship between sequence numbers (SN) and window size (WS).
Example below has SN = 32 and WS = 8.

```
Packet 0 sent
Packet 1 sent
:
Ack 0 received
Current window = [1, 2, 3, 4, 5, 6, 7, 8]
Ack 1 received
Current window = [2, 3, 4, 5, 6, 7, 8, 9]
Packet 3 *****Timed Out *****
Packet 3 Re-transmitted.
:
:
Session successfully terminated

Number of original packets sent:  xxxx
Number of retransmitted packets: xxxx
Total elapsed time: xxxx
Total throughput (Mbps): xxxx
Effective throughput: xxxx
```

Example of output on the receiver side:

```
Packet 0 received
Checksum OK
Ack 0 sent
Current window = [1]   ← and WS = 1 for GBN
Packet 1 received
Checksum failed
Current window = [1]
:
Last packet seq# received:xxxx
Number of original packets received:  xxxx
Number of retransmitted packets received: xxxx
```

Stress Test Files

You can create a test file of 2G using the dd command:

```
dd if=/dev/urandom of=/tmp/1Gtestfile iflag=fullblock bs=500M count=4
```

You should first check `/tmp` to make sure that a 1G file exists. If someone else had created it, use that one instead of creating another huge file.

Submission

1. Create and zip a folder with both your email names, e.g., stark_lannister_proj.
2. Put all .cc files (no executables) and a README under the folder.
3. The naming of all .cc files must adhere to the naming convention used in the naming of your folder, e.g.,
   (a) stark_lannister _main.cc
   (b) stark_lannister _SR.cc
   (c) stark_lannister _packet.h, etc.

4. Submit the zip folder to Canvas
5. Schedule a demo.

CS 462 – Computer Networks Project Cover Page

Name(s) _____

1. Include a copy of your source code, a design document (if required), and a sample run
2. Fill in the table of contents including the names of routines and the corresponding pages.
3. Indicate the status of your program by checking one of the boxes.
4. Submit the assignment at the beginning of the class on the due date.

Program Status: (check one box)

□ Program runs with user-defined test cases. □ Program runs with some errors.

□ Program compiles and runs with no output. □ Program does not compile.

Workload:

<table>
<tr><td></td><td colspan="4">Written By</td></tr>
<tr><td>Design document</td><td colspan="4"></td></tr>
<tr><td>Test cases</td><td colspan="4"></td></tr>
<tr><td rowspan="7">Modules (classes)
List only major modules that are typically more than 50 lines of code</td><td>Module name</td><td>Written by</td><td>Module name</td><td>Written by</td></tr>
<tr><td>S & W</td><td></td><td>Testing</td><td></td></tr>
<tr><td>GBN</td><td></td><td></td><td></td></tr>
<tr><td>SR</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

Grading:

Metric	Score Points
Design (Data Struc/Alg) (Fast/Slow/Molasses) Stress test (memory leaks, seg faults, etc.) Correctness	/ 30 / 30 / 30
Lab Total Specs Compliant	/ 90 / 10
Project Total	/ 100