Lab 1: Implementation of Reliable Data Transfer Protocol - Selective Repeat
IERG 3310, Spring 2025, Due: 18/3/2025, Tue., 11:59pm

In this C programming assignment, you will complete the sending and receiving transport-level code for a simple reliable data transfer protocol Selective Repeat (SR). This assignment doesn't involve any socket programming. Instead, your program for both sender and receiver will be implemented in a protocol simulator provided in *sr.c*. Your program will simulate the behavior of SR as specified in the class notes (some differences will be noted in the following). A skeleton file *sr.c* is provided and **you are ONLY required to implement the blank functions listed in Section II of the file (except that you need to fill in your student ID in Section III).** The functions provided in Section IV "Student-callable ROUTINES" of the file can be called by your functions.

1. Program Input

The program takes three input parameters from *stdin* as follows:

- The number of messages to simulate: the number of messages from layer 5 that will be sent by SR.
- Time between messages from sender's layer5: the time duration between two messages from layer 5. This value is usually smaller than the RTT which is set to 15 simulation time units.
- Channel pattern string: this string specifies the characteristics of the underlying channel. Each character in the string specifies the action taken by the channel for a packet being transmitted. Character 'o' indicates the packet will be received correctly, '-' indicates the packet will be corrupted, 'x' indicates the packet will be lost. For example: a channel pattern string "xo-xo" indicates that 1st packet will be lost, 2nd received correctly, 3rd corrupted, 4th lost, and 5th received correctly. All the following packets will then repeat the same pattern, i.e., 6th lost, 7th received correctly and so on. Note that you can input different pattern string to test your program. Note that the sequence of the packets in channel pattern refers to both data packets from sender to receiver and the ACK packets from the receiver to sender. For example, the 1st packet is a data packet from sender to receiver, and 2nd packet may be ACK from receiver to sender (if the first data packet was not lost), 3rd packet may be a data packet again, and so on. The length of channel pattern string input by the user must be shorter than 40 characters.
- Sender's window size: the number of packets that the sender keeps in its sending window. The default value is 8.
- Each packet has its timer. To keep it simple, the timeout should always be set to be RTT in your program whenever the timer is started.

The accuracy of your program will be graded based on the results by different input parameters.

2. Program Output

The program outputs the important events during the execution of SR. These events include:

Sent a packet: the sender passes the packet to layer 3.

Received a packet: the receiver receives a packet with correct checksum.

Receive a corrupted packet: the receiver receives a packet with wrong checksum.

Timeout: the timer of packet goes off.

Buffer packet: when the sender receives a packet from layer 5 while its sending window is full, it buffers it in the buffer (defined by *struct msg buffer[MAXBUFSIZE];* in the code).

Summary information of overall packets from sender to receiver: after the simulator terminates (normal ends or earlier ends due to some reasons), it's expected to print some overall analysis for packets from sender to receiver, including **1) the total number of sent packets, and correctly received/resent/corrupted/lost; 2) the goodput of the SR.** The goodput is defined as the total number of bits correctly received per unit of time and has unit of bits per second (bps). In computer networks, goodput (a portmanteau of good and throughput) is the application-level throughput of a communication, i.e., the number of useful information bits delivered by the network to a certain destination per unit of time [1]. Suppose the size of each packet is 32KB (32 K bytes) and the unit of the simulation time is second. Please retain three decimal places for the results.

The output of each event also includes important information such as current system time (which can be obtained by calling *currenttime()*), packet sequence number, the base of the sending window and so on. For example, the following two lines are excerpted from the output of the program:

[18.0]□A:send□packet□[2]□base□[1]

……..

[40.0]□A: packets□[x]□time_□out, □resend packet

□ represents a space. Each line of output starts with current time and the node name (A is sender and B is receiver), then the event (send/receive etc.), and packet sequence number if a packet is involved, and the base of the sender's window if this event is from sender.

The following is the complete output of the program when the channel is error-free (the input from the user is underlined), window size is 3.

```
**********************************************************************************************************
    IERG3310 lab1: Reliable Data Transfer Protocol-SR, implemented by 1155xxxxxx (your SID)
**********************************************************************************************************
Enter the number of messages to simulate:
```

6

Enter time_ between messages from sender's layer5 [> 0.0]:

6

Enter channel pattern string

oooooooooooooo

Enter sender's window size

3

****

Packet Transmission Log

****

[6.0] A: send packet [0] base [0]

[12.0] A: send packet [1] base [0]

[12.5] B: packet [0] received, send ACK [0]

[18.0] A: send packet [2] base [0]

[18.5] B: packet [1] received, send ACK [1]

[19.0] A: received ACK [0]

[24.0] A: send packet [3] base [1]

[24.5] B: packet [2] received, send ACK [2]

[25.0] A: received ACK [1]

[30.0] A: send packet [4] base [2]

[30.5] B: packet [3] received, send ACK [3]

[31.0] A: received ACK [2]

[36.0] A: send packet [5] base [3]

[36.5] B: packet [4] received, send ACK [4]

[37.0] A: received ACK [3]

[42.5] B: packet [5] received, send ACK [5]

[43.0] A: received ACK [4]

[49.0] A: received ACK [5]

Simulator terminated.

****

Packet Transmission Summary

****

From Sender to Receiver:

total sent pkts: 6

total correct pkts: 6

total resent pkts: 0

total lost pkts: 0

total corrupted pkts: 0

the overall throughput is: 31.347 Kb/s

The following is the complete output of the program when the first packet is lost, fourth and fifth corrupted by the channel (as specified by the channel pattern string), window size is 4.

```
**********************************************************************************************************
        IERG3310 lab1: Reliable Data Transfer Protocol-SR, implemented by 1155xxxxxx (your SID)
**********************************************************************************************************
Enter the number of messages to simulate:
6
Enter time_ between messages from sender's layer5 [ > 0.0]:
2
Enter channel pattern string
xoo--oooooooooooooooooooooooo
Enter sender's window size
4
**********************************************************************************************************
                                Packet Transmission Log
**********************************************************************************************************
[2.0] A: send packet [0] base [0]
[4.0] A: send packet [1] base [0]
[6.0] A: send packet [2] base [0]
[8.0] A: send packet [3] base [0]
[10.0] A: buffer packet [4] base [0]
[10.5] B: packet [1] received, send ACK [1]
[12.0] A: buffer packet [5] base [0]
[12.5] B: packet [2] received, send ACK [2]
[14.5] B: packet [999999] corrupted
[17.0] A: packet [0] time_ out, resend packet
[17.0] A: ACK corrupted
[19.0] A: packet [1] time_ out, resend packet
[19.0] A: received ACK [2]
[23.0] A: packet [3] time_ out, resend packet
[23.5] B: packet [0] received, send ACK [0]
[25.5] B: packet [1] received, send ACK [1]
[29.5] B: packet [3] received, send ACK [3]
[30.0] A: received ACK [0]
[30.0] A: send packet [4] base [1]
[32.0] A: received ACK [1]
[32.0] A: send packet [5] base [2]
[36.0] A: received ACK [3]
[36.5] B: packet [4] received, send ACK [4]
```

[38.5] B: packet [5] received, send ACK [5]

[43.0] A: received ACK [4]

[45.0] A: received ACK [5]

Simulator terminated.

****

Packet Transmission Summary

****

From Sender to Receiver:

total sent pkts: 9

total correct pkts: 6

total resent pkts: 3

total lost pkts: 1

total corrupted pkts: 1

the overall throughput is: 34.133 Kb/s

3. Walk Through

To complete this lab, **you are ONLY required to implement the blank functions listed in Section II of the skeleton file** *sr.c* **(except that you need to fill in your student ID in Section III).** This file uses the following notation. Node A is the sender and B is the receiver. "Layer 5" refers to the application layer that calls the functions of SR protocol to send data. "Layer 4" refers to the transport layer in which you implement the SR protocol. "Layer 3" refers to the network layer that calls the functions of SR protocol when a new packet arrives. Each section of file *sr.c* is explained in the following.

- SECTION I: this section defines a few variables that can be used by the routines to be implemented. You may define new global variables if necessary. However, you should reduce the number of new global variables to the minimum. Excessive new global variables will result in point deduction (see 'Grading' for details).
- SECTION II: this section contains ALL the functions that you need to complete in this lab.
    - *ComputeChecksum( )*: compute the checksum of the packet to be sent from the sender. Note that you need to include payload, sequence number, and ACK number in the computation. The payload always has a fixed length of 20 (as defined in the structure '*struct msg*'). What's more, possible carry-over in the adding operations needs to be considered as stated in the lecture slides.
    - *CheckCorrupted( )*: check the checksum of the packet received, return TRUE if packet is corrupted, FALSE otherwise.
    - *A_output( )*: called by layer5 (application) to send data to the other side. Follow the operation of the SR sender discussed in class. The basic logic flow of this function

is described in the following. Note that only important steps are discussed here and it's your responsibility to ensure that your code implements the correct logic of SR and produces the correct results described in Program Output. First, check if '*nextseqnum*' falls within the sender window. If yes, create a packet by filling in the payload (which is contained in the application message) and a proper sequence number. As this is a data packet, set ACK number to 'NOTUSED'. Compute checksum and send out the packet by calling function *tolayer3*. Copy the packet to the buffer defined by *winbuf*. Then create a timer by filling a proper sequence number and start the timer. The timeout of the timer should be set to *RTT* (which is defined in the file). If '*nextseqnum*' does not fall within the sender window, buffer the message if the sender message buffer is not full, exit otherwise (which typically will not occur).

- *A_input( )*: called from layer 3, when a packet arrives for layer 4. Follow the operation of the SR sender discussed in class. The basic logic flow of this function is described in the following. Note that only important steps are discussed here and it's your responsibility to ensure that your code implements the correct logic of SR and produces the correct results described in Program Output. First, check if the ACK packet is corrupted. If the ACK packet is corrupted, print "[currentime_()] A: ACK corrupted". If not, then stop the timer of the packet and ack the packet in window buffer. If the acked packet is located on base, then advance the window base pointer according to acked packets in the window buffer. If the message buffer is not empty (i.e., there are application messages waiting to be sent), create a new packet and send it.
- *A_timerinterrupt( )*: called when A's any time_r of packet goes off. Restart the timer and resend the packet.
- *B_input( )*: called from layer 3, when a packet arrives for layer 4 at receiver B. Follow the operation of the SR receiver discussed in class. The basic logic flow of this function is described in the following. Note that only important steps are discussed here and it's your responsibility to ensure that your code implements the correct logic of SR and produces the correct results described in Program Output. If the packet from A is not corrupted, create an ACK packet (setting seqnum to NOTUSED), send it by calling *tolayer3*, and deliver received packet to layer 5 by calling *tolayer5*. Otherwise, drop the corrupted packet.
- *print_info( ):* called after the simulator terminates to print the overall analysis to packets from sender to receiver. You can use variables such as *packet_sent, packet_correct, packet_resent, packet_lost, packet_corrupt* to calculate them.
- *A_init( )* and *B_init( )*: called before any other A's or B's functions to do initialization. Be sure to initialize the variables used by your functions such as *base, nextseqnum,*

buffront bufrear, msgnum, winfront, winrear, pktnum etc.

- SECTION III: network emulation code. THERE IS NOT REASON THAT ANY STUDENT SHOULD HAVE TO READ OR UNDERSTAND THE CODE BELOW. YOU SHOLD NOT TOUCH, OR REFERENCE (in your code) ANY OF THE DATA STRUCTURES.
- SECTION IV: The functions provided in this section can be called by your functions. Reading the code of these functions may help you understand how the network emulator works. However, this is absolutely not necessary in order to complete the lab. AND YOU SHOULD NOT MODIFY ANY OF THE CODE IN THIS SECTION.

4. Important Notes

- It is important for your program to follow **exactly** the output format shown in the above two examples.
- The sequence number from the sender starts from 0.
- When the sender receives a packet from layer 5 while its sending window is full, it buffers it in the buffer (defined by *struct msg buffer[MAXBUFSIZE];* in the code), and sends it when the window has a slot.
- You are **NOT** allowed to change any code in Sections 0, III, and IV of *sr.c* **(except that you need to fill in your student ID in Section III)**. Violations will result in point reduction (see Grading Policy).
- Suggested IDE (for conveniently compiling and debugging): Codeblocks, Visual Studio, CLion, online C compiler, etc.

5. Grading

1. Late penalty: 20% per day.
2. Plagiarism: Those students involved will get 0% for their assignment.
   1) You are welcome to discuss with other students. Just don't copy from one another.
   2) Copying lab solutions from public sources or other students is considered plagiarism.
3. Change of code in Sections 0, III, and IV of *sr.c*: -5 points per change **(except that you need to fill in your student ID in Section III).**
4. You may define new global variables in Section II of *sr.c*. However, you should keep the number of new global variables to be fewer than three. 2 points per variable will be deducted when the number of new variables is more than three.
5. The total points of this lab assignment are 100 points. Your submission will be graded according to the following criteria:
   1) **Compilation:** Your code should be compliable on the environment you stated in your report. **50 points will be deducted if your code fails to compile.**

2) **Protocol logic and programming style**: Correct logic of **SR** protocol behavior; Proper and meaningful comments for variables and statements.
3) **Accuracy:** Your program should generate the correct results AND follow the exact format of output as specified in the examples.
4) **Analysis Report:** You are supposed to submit a report **(No more than two A4 pages)** to clearly state your analysis on the lab, which will account for 10 points in lab1. The report should include:
    - The environment for compiling your codes,
    - Flow chart of your function 'A_output', 'A_input', 'B_input' and 'A_packet_time_rinterrupt',
    - Problems you met in the implementation and your corresponding solutions.

6. Bonus Points

The part comparing Go-Back-N (GBN) and SR will earn extra 20 points. In this experiment, you must implement the GBN protocol based on the existing simulator used for SR. You need to compare the performance of them with different conditions to demonstrate the cases where SR outperforms GBN and vice versa if possible. To earn the extra marks, you need to provide:

- A separate source file with name *gbn.c* in addition to *sr.c*.
- A document (in txt/word/pdf format) that describes your analysis and code implementation in details.
- Numerical results and output of the program to verify your conclusion.

7. Submission

1. For the basic implementation of SR protocols: You are supposed to submit the **modified *sr.c*** and the **corresponding report** specified in the 'Grading' section.
2. For the 'Bonus Points' section: You are supposed to submit another source code ***gbn.c*** and the **document** specified in the 'Bonus Points' section.
3. All of your submissions should be included **in one folder** named with related to your student ID and uploaded through the *'Blackboard'* system.

[1] Definition of Goodput. https://en.wikipedia.org/wiki/Goodput.