Homework2

Due date: 2021/11/18 03:00

Homework Policy:(Read before you start to work)

1. 作業請勿抄襲，如果被發現，作業以零分計算
2. 如果作業上遇到困難可以討論，但是手寫和程式碼的部分必須是你自己完成，並且請在作業的pdf檔註明討論的同學姓名及學號
3. 程式作業請於期限內至ceiba作業區上傳，格式爲zip檔，解壓縮後應恰爲一個以學號爲名的資料夾，資料夾內有report.pdf檔及以學號命名之.c檔案，如下所示

Figure 1: folder tree

4. 逾期繳交一天內，分數$\times\frac{2}{3}$，超過一天未滿兩天，分數$\times\frac{1}{3}$，超過兩天則不予計分， 請務必盡早開始，並努力完成。
5. 如有任何問題歡迎來信，並請在郵件的標題註明課程。
   範例:[2021ICN]作業二問題
   信箱:莊鎧爾 R09942172@ntu.edu.tw

Programming Problems:(Programming Assignment in Chapter 3)
Due: 2021.11.18 - 03:00 am
[**Overview**]

In this homework, you will write the transport-level code for the reliable data transfer protocol: Go-Back-N. Your code will be executed in a simulated hardware/software environment. However, the programming interface provided to your routines is very close to what is done in an actual UNIX environment. Stopping/starting of timers is also simulated, and timer interrupts will cause your timer handling routine to be activated.
The codes you will write are for the sending entity (A) and the receiving entity (B). There's only one direction that 'real' data be sent(From A to B). Of course, B will have to send packets to A to acknowledge receipt of data. Your codes will be called by (and will call) procedures already written that emulate a network environment.(See figure 1.)

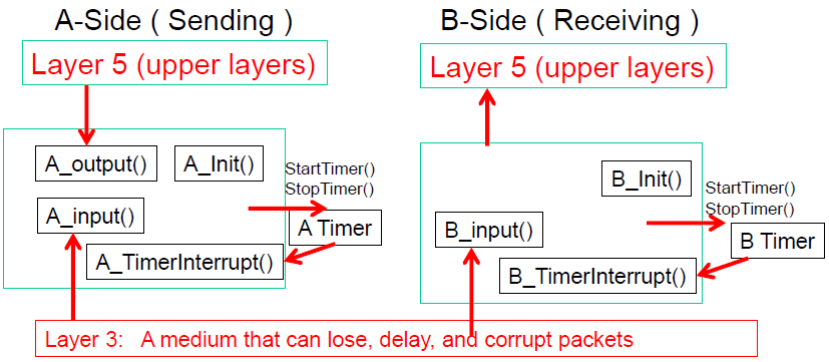

Figure 2: Overall structure

1. [**GBN Programming**] – 60%(8% for each function + 12% report)
   a.(Programming 48%) The following figures 3,4 describe the state of sender and receiver. You have to implement some functions in sender(A) and receiver(B). The sliding window size should be set to 8. The timeout value should be set by a EWMA estimated timer.
   First we define the data structure used in this assignment.

   - The unit of data passed between the upper layer (layer5) and your protocol is a message, which is declared as:

     ```
     struct msg {
         char data[20]
     };
     ```

   Your sending entity(A) will receive data in 20-byte chunks from layer5; your receiving entity(B) should deliver 20-byte chunks of correctly received data to layer5 at the receiving side.

   - The unit of data passed between your routines and the network layer is the packet, which is declared as:

```
struct pkt {
    int seqnum;
    int acknum;
    int checksum;
    char payload[20];
};
```

Your routines will fill in the payload field from the message data passed down from layer5. The other packet fields will be used by GBN protocols to ensure reliable delivery.

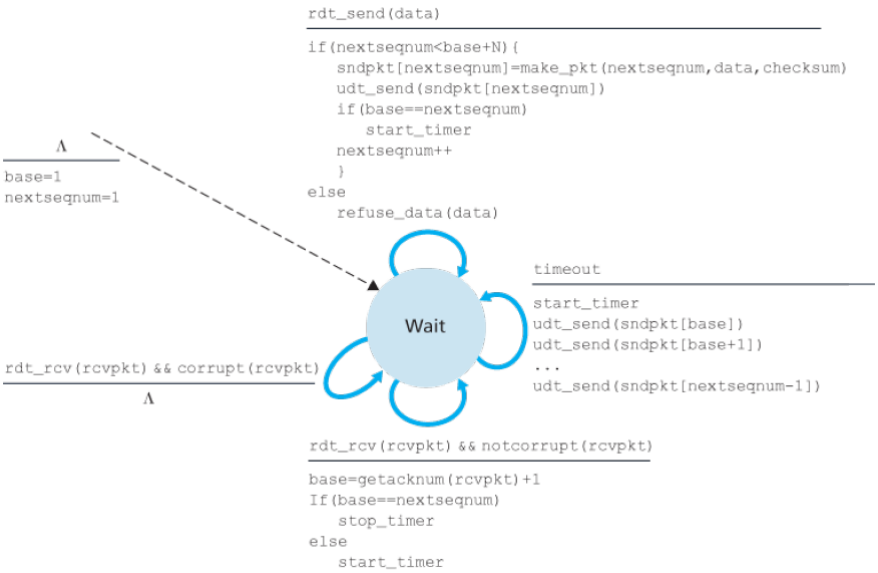

Figure 3: FSM of GBN sender

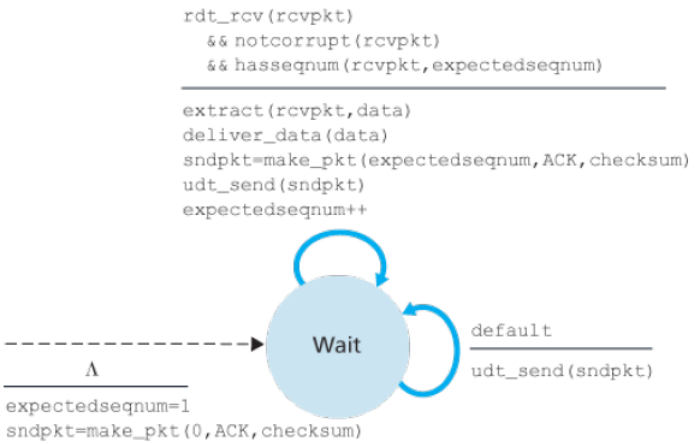

Figure 4: FSM of GBN receiver

In the code, you have to implement A-side as GBN sender and B-side as GBN receiver. There are some functions you have to implement.

- `struct Sender, struct Receiver`
  The data structure of sender and receiver should be constructed first. You should design the state variables which can make layer 4 sender and receiver functioning properly. Be sure to set the sender structure with sliding window size 8.
- `void A_init(), void B_init();`
  This routine will be called once, before any of your other A-side or B-side routines are called. It can be used to do any required initialization.
- `void A_output(struct msg message);`
  where message is a structure of type msg, containing data to be sent to the B-side. This routine will be called whenever the upper layer at the sending side (A) has a message to send. It is the job of your protocol to ensure that the data in such a message is delivered in-order, and correctly, to the receiving side upper layer.
- `void A_input(struct pkt packet);`
  where packet is a structure of type pkt. This routine will be called whenever a packet sent from the B-side (i.e., as a result of a tolayer3() being done by a B-side procedure) arrives at the A-side. packet is the (possibly corrupted) packet sent from the B-side.
- `void A_timerinterrupt(int index);`
  This routine will be called when A's timer expires (thus generating a timer interrupt). You'll use this routine to control the retransmission of packets. See starttimer(int index) and stoptimer(int index) below for how the timer is started and stopped.
- `void B_input(struct pkt packet);`
  where packet is a structure of type pkt. This routine will be called whenever a packet sent from the A-side (i.e., as a result of a tolayer3() being done by a A-side procedure) arrives at the B-side. packet is the (possibly corrupted) packet sent from the A-side.

(i) [**Software interfaces**]
The following routines are provided to you and can be called by your routines:

- `void startTimer(int AorB, float increment, int index);`
  where calling entity is either 0 (for starting the A-side timer) or 1 (for starting the B side timer), and increment is a float value indicating the amount of time that will pass before the timer interrupts. A's timer should only be started (or stopped) by A-side routines, and similarly for the B-side timer. Please use a EWMA estimated timer to set appropriate increment value.
- `void stopTimer(int AorB, int index);`
  where calling entity is either 0 (for stopping the A-side timer) or 1 (for stopping the B side timer).
- `void toLayer3(int AorB, struct pkt packet, int index);`

where calling entity is either 0 (for the A-side send) or 1 (for the B side send), and packet is a structure of type pkt, index would be index of packet in sliding window. Calling this routine will cause the packet to be sent into the network, destined for the other entity.

- `void toLayer5(int AorB, struct msg datasent);`
  where calling entity is either 0 (for the A-side delivery to layer 5) or 1 (for B-side delivery to layer 5), and message is a structure of type msg. With unidirectional data transfer(only from A to B), you would only be calling this with calling entity equal to 1 (delivery to the B-side). Calling this routine will cause data to be passed up to layer 5.

(ii) [**The Simulated Network Environment**]
The medium is capable of corrupting, losing, and reordering packets.. When you compile your procedures and simulation procedures together, and run the resulting program, you will be asked to specify values regarding the simulated network environment. Here's what you will be asked:

- **Number of messages to simulate** :
  The emulator (and your routines) will stop as soon as this number of messages have been passed down from layer 5, regardless of whether or not all of the messages have been correctly delivered. Thus, you need not worry about undelivered or unACK'ed messages still in your sender when the emulator stops. Note that if you set this value to 1, your program will terminate immediately, before the message is delivered to the other side. Thus, this value should always be greater than 1.
- **Loss** :
  You are asked to specify a packet loss probability. A value of 0.1 would mean that one in ten packets (on average) are lost and not delivered to the destination.
- **Corruption** :
  You are asked to specify a packet loss probability. A value of 0.2 would mean that two in ten packets (on average) are corrupted. Note that the contents of payload, sequence, ack, or checksum fields can be corrupted. Your checksum should thus include the data, sequence, and ack fields.
- **Average time between messages from sender's layer5** :
  You can set this value to any non-zero, positive value. Note that the smaller the value you choose, the faster packets will be be arriving to your sender.
- **Tracing** :
  Setting a tracing value of 1, or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for emulator-debugging purposes. A tracing value of 2 may be helpful to you in debugging your code.

(iii) [**Program Testing**]

- **Submission** :
  You should only write your code in hw2.c, and submit as [student_id].c .
  Make sure you can compile your code with the testing command.
- **Compile** :

```
$ gcc [student_id].c -o [student_id] -std=gnu11
```

b.(Report 12%) Please screenshot the ouputlog of the program. There are four commands you should screenshot and discuss how GBN works. You can mark some important logs that you want to discuss with. We will run your code to verify your outputlog. The points you get depends on whether you can have a comprehensive discussion on the functions of GBN.

(a) ./[student_ID] 10 0 0 10 2

(b) ./[student_ID] 20 0.1 0 10 2

(c) ./[student_ID] 20 0 0.1 10 2

(d) ./[student_ID] 20 0.1 0.1 10 2

We highly recommend you using

```
./[student\_ID] 10 0 0 10 2 > [outputfile]
```

to store the outputlog into a output file.

2. [**Exponential weighted moving average**] – 40 %
   a. (Programming 30 %). The SampleRTT will fluctuate from segment to segment due to congestion in routers and to the varying load on the end systems. To fix this problem, TCP maintains an average, called EstimatedRTT and updates it as follows.

$$\texttt{EstimatedRTT} = (1 - \alpha) \cdot \texttt{EstimatedRTT} + \alpha \cdot \texttt{SampleRTT}$$

Figure 5: EstimatedRTT formula

   In problem 1, you use TCP exponential weighted moving average (EWMA) to set appropriate timeout value for GBN transmission. In this problem, please discuss the impacts on EstimatedRTT with different alpha. You have to put the outputlog in your report as a proof for your discussion.

   b. (Plot 10 %). Please plot the result of EstimatedRTT with different alpha as Fig. 6.

3. [**Bonus**] – 10 %
   You can change the timeout value to a constant, and discuss the difference between constant timeout and EWMA estimated timeout. You have to put the outputlog in your report as a proof for your discussion.

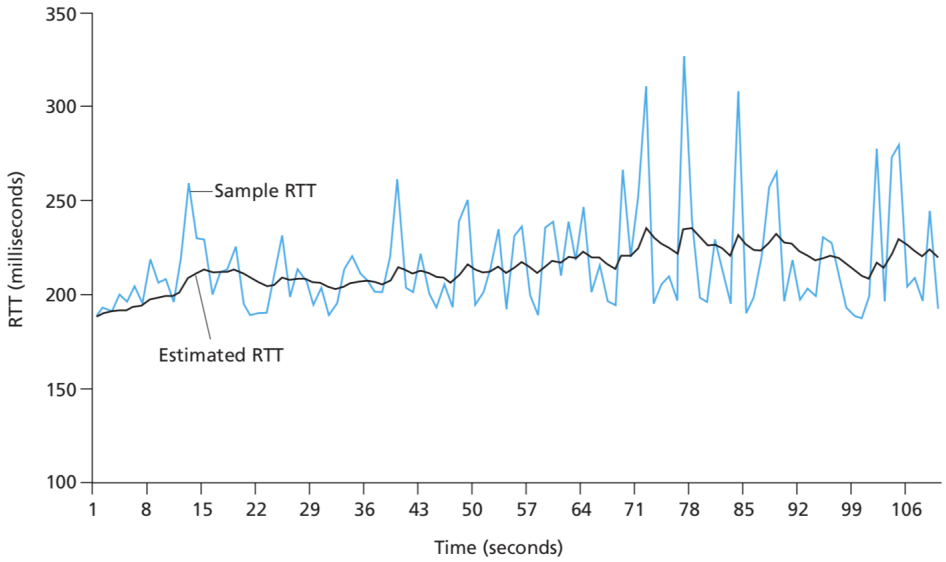

Figure 6: ERTT plot