Assignment 3
16-05-2018

Programmazione di Reti

Ing. Chiara Contoli
chiara.contoli@unibo.it

Corso di Laurea Triennale in
Ingegneria e Scienze Informatiche

Homework

Goal. The goal of this assignment is to leverage the network simulator modified during Assignment 2, in order to design and implement procedures that reproduce the **Selective-Repeat protocol**.

Programming environment

To design and implement the Selective Repeat protocol, you will leverage the network simulator modified during Assignment 2 (original network simulator is attached in Assignment 2 directory).

As described in the previous assignment, the network simulator has the primary goal of reproducing the behavior of an *unreliable network* layer. This means that the network might both *lose* and *corrupt* packets. The simulator is also able to emulate other characteristics in the form of routines. Therefore, the given environment contains two types of routines:

- those that you **do not have to modify**; those routines are devoted to emulate layer 3, the underlying layers and other features;
- those that you **have to design and implement**;

Your Assignment

Routines:

- A_output()
- A_input()
- A_timeinterrupt()
- A_init()
- B_input()
- B_init()

Those routines are detailed in the following.

Your assignment

Task1: In this task, you will **modify *netsimulator.c* code** in order to **implement** the **Selective Repeat protocol** seen during classes (for those who did not attend classes, see slides Chapter_3_V7.01.pdf). In particular, you will **implement** the protocol considering **only unidirectional transfer of data** (i.e., only from Sender A to Receiver B). However, keep in mind that B has to send packets as well, since the Receiver has to acknowledge receipt of data. Therefore, you are asked to write procedures both for Sending and Receiving side.

Routines you are asked to implement will be called by (and will call) procedures already implemented that emulate the network environment. All these routines (i.e., those already implemented and those to be implemented) are all declared in the file *netsimulator.c* .

Considerations

Source code contains several useful comments that describes the programming environments and that has to be read **carefully**. Few of these are reported here after.

A message is the unit of data passed between the upper layer and your protocol, and is declared as follow:

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

Upper layer is indicated with layer5 in the source code, and will deliver data in 20 bytes chunks to Sender A entity, while Receiver B entity will deliver these 20 bytes chunks of correctly received (and in order) data to the layer5 of receiving entity.

Considerations

The packet is instead considered the unit of data passed between your routines and the network layer, and is declared as follow:

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

Payload will be filled by your routines with data message received by layer5. Reliable delivery will be achieved by your protocol through the other packet parameters.

Considerations

Routines listed below are given and can be called by your routines:

- starttimer(calling_entity, increment)
- stoptimer(calling_entity)
- tolayer3(calling_entity, pkt): send packet into the network
- tolayer5(msg): called only by B to pass msg to the layer 5

calling_entity values:

- 0 for the Sending side
- 1 for Receiving side

increment: amount of time that has to elapse before timer interrupts. Since we consider only unidirectional transfer from A to be, only A should maintain the timer. **Note**: consider that it takes (on average) 5 time units for a packet to get to the other side of the network; therefore, chose such value of increment properly.

Your Assignment

Routines you are asked to implement:

- A_output(msg): called whenever Sender's upper layer (A) has a message to send. Your protocol is in charge of ensure that msg is delivered in-order and correctly at Receiver's upper layer (B)
- A_input(pkt): called whenever a pkt sent from the B side arrives at the A side (i.e., as a result of a *tolayer3()* call at B side)
- A_timeinterrupt(): when A's timer expires, a timer interrupt is generated through the call of this routine (it is used to control retransmission of packets)
- A_init(): used for (any) initialization purpose, it is called before any other A side routines
- B_input(pkt): called whenever a pkt sent from the A side arrives at the B side (i.e., as a result of a *tolayer3()* call at A side)
- B_init(): used for (any) initialization purpose, it is called before any other B side routines

Considerations

To compile and run the *netsimulator.c* source code, use following commands:

- gcc –g netsimulator.c –o netsimulator
- ./netsimulator

You will be asked to provide values about the simulated network environment:

- Number of message to simulate
- Loss probability
- Corruption probability
- Average time between messages from Sender's layer5
- Window size
- Retransmission timeout
- Tracing
- Random seed

Considerations

- **Number of message to simulate**: is the number of messages to be passed down from layer 5; this value should always be greater than 1, since the emulator will stop as soon as such number of messages have been passed down from layer 5 whether all of these messages have been correctly delivered or not. This means that you do not have to worry about messages still in your sender when the emulator terminates, thus resulting in undelivered and unACK'ed messages.
- **Loss probability**: is the packet loss probability. E.g.: a value of 0.1 indicates that, on average, in ten packets one is lost.
- **Corruption probability**: is the packet corruption probability. E.g., 0.2 indicates that, on average, one in five packets are corrupted. Note that corruption apply for those packets that **are not** lost. Also, your checksum should include data, sequence number and ACK fields, since payload, sequence, ACK or checksum can be corrupted.
- **Average time between messages from sender's layer5**: it can be any non zero positive value; the smaller the value, the faster messages will be arriving to your sender
- **Window size**: is the sending window limit
- **Retransmission timeout**: is the value of retransmission timeout
- **Tracing**: is the debugging level. A value of 2 (recommended) would be helpful to debug your code, since it will print useful information about what is going on in the emulation; a value greater than 2 would print any sort of messages, while 0 would turn debug off.
- **Random seed**: it is an initial seed for the generation of random numbers used for event scheduling (e.g., packet arrivals, delays, loss and corruption)

Your Assignment

Note that Assignment 2 is preparatory to this Assignment, but in this Assignment you have to take into account few things that you did not have to consider in the previous one. This time, you should consider also the following behavior:

- the Receiver has to be able to buffer out of order packets and send cumulative ACKs
- the Sender has to retransmit only the next unACK'ed (i.e., missing) packet either on timeout or duplicate ACK

Also, this time:

- A_output(msg): will be called sometimes when there are unacknowledged messages in the medium; therefore, because of the nature of Selective Repeat protocol, you will have to buffer multiple messages in your Sender side. This means that you have to handle the case in which Sender is not allowed to send the new message because it falls outside of the window. For your assignment, even if it is not an elegant solution compared to what happens in the real-word case scenario, you can consider to adopt the solution of finite buffers of maximum 50 messages at the Sender side. This means that, if at certain point in time such space is fully in use, you Sender side will abort by exiting.
- A_timerinterrupt(): recall that this procedure is called when Sender's timer expires (generating a timer interrupt). Note that you have only one timer, and may have many outstanding, unACK'ed packets in the medium; therefore, you will have to think about how to use this single timer.

To recap

For Task1. Implement the Selective Repeat protocol following instructions provided by these slides and according to what you have seen during classes, considering that:

- You have to implement **only** the unidirectional case.
- You have to use **only ACK** (i.e., **do not consider NACK**).
- ACK might be corrupted as well.
- This time you also have to consider the implementation of the behavior described at slide 12

Your protocol must work at least in the following 3 cases, described in the next slide.

Cases

Generate a run long enough so that at least 20 messages are successfully transferred from Sender to Receiver (i.e., sender receives 20 ACKs), considering a window size of 8, set a retransmission timeout of 20, a random seed of 2233, a trace level of 2, and a meantime of 10 between arrivals. Note that, for the assignment, the adoption of a static retransmission timeout is acceptable, even if in real-world implementation this value changes dynamically.

Verify that your application works in the following cases:

Case 1: set loss and corruption probability to zero (i.e., test the case in which there is no loss and corruption at all)

Case 2: set loss probability and a corruption probability of 0.2

Case 3: set loss probability and a corruption probability of 0.5

What to submit (and how)

Whoever fails in following these simple rules will incur a penalty in the grading process.

Submit by your institutional e-mail a zip file of a directory containing the **whole** assignment. Mail sent from others e-mail will not be considered. The *object* of the e-mail *must* indicate *which* assignment you are submitting and *all* group members full name. Also, whoever of the group member submit the assignment *must* put in CC *all* the other member of the group.

The directory **must** contain:

1. The modified *netsimulator.c* source code;
2. The generate “OutputFile” (one for each of the tested case)
3. A report of the **whole** assignment in *.pdf* format(; report has to be written in *neat* and *clear* Italian.

What to write in the Report

Length is not always a synonymous of quality: keep your report relevant to the topic you are required to write about, and try to be clear and effective in your explanation.

For **Task1**, report **must** contain:

- A description of the design strategy, i.e., how you implemented each of the the routine listed at slide 4.
- A Finite State Machine that describes the behavior of the Sender and the Receiver side implemented in your solution.
- Tests to show that the protocol works in the three cases previously described; to this end, include in the document some sample outputs and all the parameters provided as input at the network simulator startup for each tested scenario.
- Discuss some consideration about the usage of a static retransmission timeout.

Hints and additional considerations

Sample outputs can be obtained by making your procedures log (i.e., print out) a message whenever an event occurs at Sender or Receiver side (e.g., message or packet arrival, a timer interrupt and action taken in response to those events). In particular, sample output should contain information about:

- Who is sending/receiving packet
- If the packet is data or ACK
- Which event is occurring
- How the event is handled
- Other useful information about the dialog that prove the correctness of your protocol

When printing your sample output in the report, you should highlight how your protocol correctly recovered from data and ACK loss and corruption; also, make sure to make distinctions between the case packets are delivered and the case in which are not. Identify at least one place when full window is retransmitted after a timeout, at least one place when an ACK is lost and a later cumulative ACK in the same direction move the Sender's window by more than 1.

Hints and additional considerations

Your program should also gather, via proper counters, statistics about:

- number of original data packets transmitted
- number of retransmissions
- number of ACK packets
- number of corrupted packets received (consider both data and ACK)

Those statistics should be printed at the end of the simulation.

Hints and additional considerations

Use global variables for shared state among your routines, if needed.

For checksumming, a TCP-like approach is suggested, i.e., sum of the integer sequence and ack field values, added to a character-by-character sum of the payload field of the packet (i.e., treat each character as if it were an 8-bit integer and just add them together).