Lab 3: Implementing a Reliable Data Transfer Protocol

50.012 Networks

Hand-out: Oct 9
Hand-in: Oct 25 (Sunday) 23:59

1 Overview

In this lab, you will be writing code to implement the "Selective Repeat (SR)" protocol for achieving reliable data transfer. It will be a fun exercise to turn what you just learned from class into actual implementation!

To help you get started, you could refer to the sample code provided (adapted from `https://github.com/z9z/reliable_data_transfer`, author Susanna Edens, Tested on Python 3), where you could find the implementation of a simple stop-and-wait (RDT 3.0 in the textbook) protocol and the implementation of a Go-Back-N (GBN) protocol.

You could spend some time to understand how the two protocols are implemented in the provided code. Read the source code and think about whether the implementation follows exactly the Finite-State Machine (FSM) definitions for the corresponding protocols as in the textbook (and lecture slides).

You could test the two protocols (i.e., stop-and-wait and GBN) implemented (in the ss.py and gbn.py file respectively) and observe their behavior by running demo_sender.py and demo_receiver.py on the same host (from two different command line consoles) with the following commands:

```
python demo_receiver.py ss
python demo_sender.py ss
```

and

```
python demo_receiver.py gbn
python demo_sender.py gbn
```

respectively. You can also test them by running file_sender.py and file_receiver.py on the same host with the following commands:

```
python file_receiver.py ss [receivedFile name]
python file_sender.py ss [originalFile name]
```

and

```
python file_receiver.py gbn [receivedFile name]
```

```
python file_sender.py gbn [originalFile name]
```

respectively. Note that the receivedFile name should be different from the originalFile name. The latter should refer to some existing file.

Note that the provided code emulates an unreliable channel in its udt.py file. It also includes a config.py file, where you can vary the packet error probability and the message loss probability of the emulated unreliable channel. You can also vary the timeout parameter used by the reliable data transfer protocols in that file. Please test the different protocols (including the SR protocol you are going to implement) by varying these parameters. You are encouraged to compare the performance of SR, GBN, and the simple stop-and-wait protocols under different conditions.

2 Selective Repeat (SR) Protocol

As you have probably observed in your GBN protocol experiments, a single packet error can cause GBN to retransmit multiple packets, many of which are unnecessary if the receiver can buffer out-of-order packets. As its name suggested, a Selective Repeat (SR) protocol avoids unnecessary retransmissions by having the sender retransmit only those packets that it suspects were lost or received in error at the receiver. This individual retransmission is supported by letting the receiver individually acknowledge correctly received packets and letting the sender keep individual timer for each transmitted packet. The SR sender and receiver views of the sequence-number space is shown in the figure below.

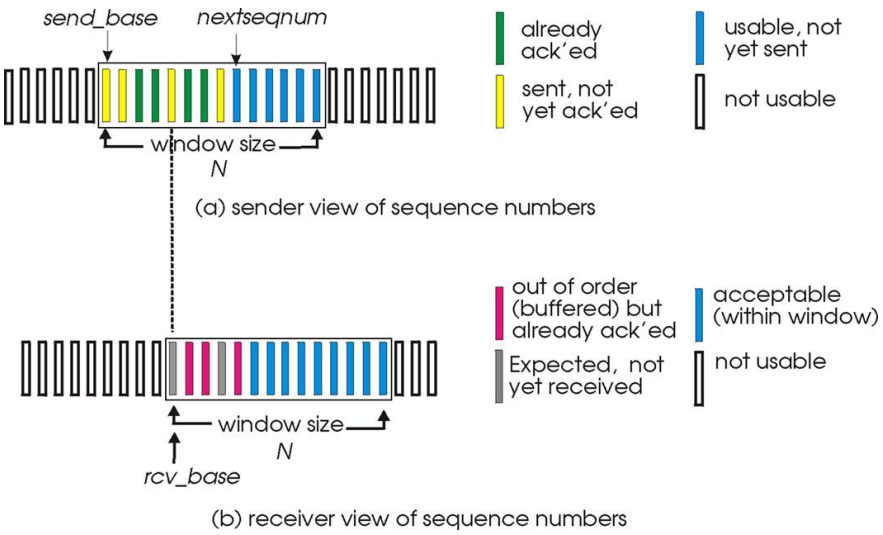

Figure 1: Selective repeat (SR): sender and receiver views of sequence-number space.

The main events and actions at the SR sender include:

- **Data received from above**: When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise, as in the reference GBN implementation, you can choose to return to the upper layer for later transmission.
- **Timeout**: Timers are used to deal with lost packets. However, different from the GBN implementation, each packet must now have its own logical timer, since only a single packet will

be transmitted on timeout. (*Hint: After you retransmit a packet, don't forget to start the timer again.*)

- **ACK received**: If an ACK is received, the SR sender marks the corresponding data packet as having been received. (*Hint: Remember to stop the corresponding timer.*) If the packet's sequence number is equal to send_base, the window base is moved forward to the unacknowledged packet with the smallest sequence number. The window move will allow new packets from upper layer to be transmitted too.

The main events and actions at the SR receiver include:

- **Packet with sequence number in [rcv_base, rcv_base+N-1] is correctly received**: In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (rcv_base), then this packet, and any previously buffered and consecutively numbered (beginning with rcv_base) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer.
- **Packet with sequence number in [rcv_base-N, rcv_base-1] is correctly received**: In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged. (*Question: why this is needed?*)

To simplify your implementation, use an integer (2 bytes as in the reference implementation) for the sequence number and assume the sequence-number space is big enough to send all your messages. (*Question: what is the largest file you could send in this case?*) With this assumption, you don't need to worry about the reuse of sequence number.

3 What to Hand in

You should create a new Python file called sr.py and put most of your code for the SR protocol there. You need to edit util.py file to add your SR protocol there. You can also change other files or introduce additional helper files. You should include sufficient comments and log printing in your code to help us (and yourself) understand the sender and receiver sides' actions triggered by different events.

You will submit a zip file that includes all the code files that are needed to run your code (not only the sr.py file). We will test your code by running the file_sender.py and file_receiver.py on the same host with the following two commands:

```
python file_receiver.py sr [receivedFile name]
python file_sender.py sr [originalFile name]
```

Before submitting your code, please check whether the receivedFile created by the file_receiver.py has the same content as the originalFile. (The commands should be able to transmit binary file too.)