Homework 2

Execution

- Format:

```
make
./agent [IP] [port] [loss rate]
./receiver [IP] [port] [file]
./sender [IP] [port] [agent IP] [agent port] [receiver IP] [receiver port] [file]
```

- Example:

```
make
./agent 127.0.0.1 8081 0.01
./receiver 127.0.0.1 8082 result.pdf
./sender 127.0.0.1 8080 127.0.0.1 8081 127.0.0.1 8082 report.pdf
```

- Notice: To prevent the standard outputs of agent, receiver and sender from interleaving, you have to run the three programs in separate terminals or direct the standard outputs to separate files.

Program structure

- The format of TCP datagram: **Source address**: 16 bytes. **Destination address**: 16 bytes. **Sequence number**: 4 bytes. **Acknowledge number**: 4 bytes. **ACK bit**: 1 byte. **FIN bit**: 1 byte. **Data**: maximum of 1000 bytes.
- Flow charts are represented with step-by-step pseudocodes as shown in the next two pages.

Difficulties and solutions

- **Design of TCP datagram**. The TCP datagram needed for the assignment is not exactly the format on the textbook. Specifically, the source and destination addresses have to include IP addresses inside so that they could be passed to C functions, like `recvfrom` and `sendto`. In order to send *ACK*, *FIN* and *FINACK*, I adopted the same desgin as in the real-world TCP datagram where the combinations of **ACK bit** and **FIN bit** meet the need. Referring to the slides and the textbook helps put together the puzzle of TCP.
- **The logic of Go Back N and congestion control**. The sender and the receiver have to coordinate and implement the same logic to make congestion control works. This is essentially the most difficult part in the assignment. Given on the slides and the textbook are the finite state machine diagrams. But FSM diagrams are far from real implementation. Details have to be taken care of with great caution. For example, actions can overlap in different transitions, and a state would depend on multiple variables. All these factors make the effect of congestion control unpredictable. The only solution is trial and error.

Agent:

1. Initialize socket address and create UDP socket.
2. Receive packets from sender and receiver.
3. Check packet type:

- **If** packet type = *Data* (`!ACK && !FIN`):
    1. Determine to drop/keep a packet:
    - **If** to drop the packet: **Return to** Step 2.
    - **Else if** to keep the packet: Foward packets to destination.
- **Else if** packet type = *FINACK* (`ACK && FIN`): Foward packets to destination.
- **Else if** packet type = *ACK*: Foward packets to destination.
- **Else if** packet type = *FIN*: Foward packets to destination.

4. **Return to** Step 2.

Receiver:

1. Initialize socket address and create UDP socket.
2. **Set** receive base := 0.
3. Receive packets from sender.
4. **If** packet type = *FIN*: **Go to** Step 8.
5. **If** sequence number = receive base:
    1. Check if the buffer is full or not:
    - **If** buffer is full: Flush the buffer.
    - **Otherwise**:
        1. Copy packet data to the buffer.
        2. **Set** receive base := receive base + 1.
6. Send *ACK* #(receive base - 1) to sender.
7. **Return to** Step 3.
8. Flush anything left in the buffer.
9. Send *FINACK* to sender.

Sender:

1. Initialize socket address and create UDP socket.
2. Set UCP time out interval.
3. Initialize:

- **Set** send base := 0
- **Set** threshold := 16
- **Set** window size := 1
- **Set** timeout := **false**

4. **Set** send limit := send base + window size.
5. Check timeout:

- **If** timeout = **true**: Send all packets from last-sent to send limit.
- **Otherwise**: Resend all packets from send base to send limit.

6. Wait for packets from receiver until timeout:

- **If** timeout:
    1. **Set** threshold := max(window size/2, 1)
    2. **Set** window size := 1
    3. **Return to** Step 4.
- **Otherwise**:
    1. **If** acknowledge number = max sequence number: **Go to** Step 8.
    2. **If** acknowledge number = send base:
        1. Check if window size < threshold:
        - **If** window size < threshold: Set window size := window size × 2.
        - **Otherwise**: Set window size := window size + 1.
        2. **Set** send base := send base + 1.

7. **Return to** Step 4.
8. Send *FIN* to receiver.
9. Receive *FINACK* from receiver.