CS352 Project 3: Automatic Repeat Request

Deadline: 13th August (No late submissions will be accepted this time)

1. Objective

Automatic repeat request (ARQ), also known as automatic repeat query, is an error-control method for data transmission that uses acknowledgements and timeouts to achieve reliable data transmission over an unreliable service. When the receiver detects an error in a packet, it automatically requests the transmitter to resend the packet. In this project, you are required to implement three mechanisms of automatic repeat request (ARQ): Stop-and-wait, Go-back-N and Selective Repeat using Python socket programming.

ARQ is used in both link layer and transport layer hence it is a very generic mechanism. In this project, we are using Python socket programming, so we simplify the packet format that only have payload and sequence number without IP address and port number.

2. Stop-and-wait

Stop-and-wait ARQ is the simplest ARQ method that both sending window size and receiving window size is 1; that is to say the sender will not send the next packet until receiving acknowledgement from receiver of previous packet. Here are the things you need to implement:

1. The sender needs to send 10 packets to receiver in total.
2. The sender needs to send 1 packet with payload and sequence number to receiver at a time, and print the acknowledge packet received from receiver.
3. The receiver needs to print the packet it received to the console and send acknowledgement packets with sequence number to sender.

Here is the format of a packet:

```
Hello:<sequence number>
```

Here is the format of an acknowledge packet:

```
Ack:<sequence number>
```

Sample expected output (Sequence number 0-4): (You need to open two separate terminal windows, run sender in one window and later run receiver in the other window)

```
sender:         receiver:

Ack:0           Hello:0
Ack:1           Hello:1
Ack:2           Hello:2
Ack:3           Hello:3
Ack:4           Hello:4
```

3. Go-back-N

Stop-and-wait ARQ is not efficient and suffers from poor network channel utilization since it sends only 1 packet at a time. To improve this, we have Go-back-N ARQ. Go-back-N ARQ set the sending window size to be N and receiving window size is 1; that is to say the sender will send N packets at a time, and the receiver will receive packets in the sequence number order. When the packets are delivered to receiver but not supposed to receive (e.g. receiver is supposed to receive packet of sequence number 3, but receive packet of sequence number 4), the receiver will discard those packets.

1. The sender needs to send 10 packets to receiver in total. The sending window size is 5.
2. The sender needs to send 5 packets with payload and sequence number to receiver at a time, and print the acknowledge packet received from receiver.
3. When the receiver receives a packet that is supposed to receive, print the packet to the console and send an acknowledgement packet with the sequence number.
4. When the receiver receives a packet that is not supposed to receive (out-of-order packet), discard it and send an acknowledgement packet with the last successfully received sequence number.

Here is the format of consecutive packets: (5 packets are sent at a time, these packets are split by "|")

```
Hello:<seq num> | Hello:<seq num> | Hello:<seq num> | Hello:<seq num> | Hello:<seq num> |
```

Here is the format of an acknowledge packet:

```
Ack:<seq num>
```

Sample expected output of in order sequence (Sequence number 0-4): (You need to open two separate terminal windows, run sender in one window and later run receiver in the other window)

```
sender:         receiver:

Ack:0           Hello:0
Ack:1           Hello:1
Ack:2           Hello:2
Ack:3           Hello:3
Ack:4           Hello:4
```

Sample expected output of lost packet sequence (Sequence number 0,1,2,4,5): (You need to open two separate terminal windows, run sender in one window and later run receiver in the other window)

```
sender:         receiver:

Ack:0           Hello:0
Ack:1           Hello:1
Ack:2           Hello:2
Ack:2
Ack:2

Ack:3           Hello:3
Ack:4           Hello:4
```

Note: in the example, we only considering 5 packets, but GBN should send N packets when it re-transmit, that is by shifting the window.

4. Selective Repeat

Go-back-N is better than stop-and-wait, but it could not automatically buffer the packets that received out-of-order. To solve this issue, Selective Repeat will buffer the packets within receiving window that received out of order. Selective Repeat sets the sending window size to be N and receiving window size to be N as well. At the receiver's end, if a packet within the receiving window is received, it will send an acknowledgement (Ack) packet to the sender. After the timeout period, if the sender does not get the Ack of a packet within its sending window, it just repeat sending this packet to receiver.

1. The sender needs to send 10 packets to receiver in total. The sending window size is 5.
2. The sender needs to send 5 packets with payload and sequence number to receiver at a time, and print the acknowledge packet received from receiver.
3. When the receiver receives a packet that is supposed to receive, print the packet to the console and send an acknowledgement packet with the sequence number.
4. When the receiver receives a packet that is not supposed to receive (duplicated packet or lost packet), discard it.

Here is the format of consecutive packets: (5 packets are sent at a time, these packets are split by "|")

```
Hello:<seq num> | Hello:<seq num> | Hello:<seq num> | Hello:<seq num> | Hello:<seq num> |
```

Here is the format of an acknowledge packet: (There could be less than 5 acknowledgements, these packets are split by "|")

```
Ack:<seq num> | Ack:<seq num> | Ack:<seq num> | Ack:<seq num> | Ack:<seq num> |
```

Sample expected output of duplicated sequence (Sequence number 0,1,2,2,4): (You need to open two separate terminal windows, run sender in one window and later run receiver in the other window)

```
sender:         receiver:

Ack:0           Hello:0
Ack:1           Hello:1
Ack:2           Hello:2
Ack:4           Hello:4

Ack:3           Hello:3
```

5. Resources

```
Python Socket Programming:
    https://www.geeksforgeeks.org/socket-programming-python/
```

6. Submission

You need to submit your work on Sakai before the deadline. Your submission MUST include all your source code and a write-up of your implementation and screenshot of your program.

Create a **.zip** file structured as follows:

```
- <your_netid>.pdf
- Stop-and-wait (folder)
    - sender.py
    - receiver.py
- Go-back-N (folder)
    - sender.py
    - receiver.py
- Selective-repeat (folder)
    - sender.py
    - receiver.py
```

Do not share and copy code from each other or do not copy and paste from the Internet, we will find out!