Project Task2 Report

1. Install python inside the container

I chose python to write the ARQ protocal, so first I installed python3 in the container. Since I used alpine image, type the following commands to install python.

```
apk update
apk add python
```

2. Write ARQ protocal program

2.1 Set TCP connection between sender and receiver

Using the TCP socket library of python, the client side is

```
client_socket = socket.socket()
client_socket.connect((host, port))
```

And the server side code is

```
server_socket = socket.socket()
server_socket.bind((host, port))
server_socket.listen(2)
conn, address = server_socket.accept()
```

2.2 Transfering data framing

First split the whole text into 200 average packets as the information part of each frame. Before transforming, add the `sequence number` prior to the information part, and this is for the server to recognize the sequence number and send back the corresponding ACK.

```
frame = str(dataCnt) + ' ; ' + frame
client_socket.send(frame.encode())
```

2.3 Selective repeat ARQ design

First a selective repeat ARQ need a sending window and a receiving window, I designed the window size to be 4.
On the client side, use an array to store the sending window. Before sending a frame, check the length of sending window, cannot transmit if it is full.

```
if (dataCnt < 200) and (len(sendingWindow) < 4):
    # Event(transmit new data frame)
```

Add sequence number of a sent frame to the sending window. Remove the sequence number from the sending window if the corresponding ack is received, and the `win_strat` (first frame in the sending window) is sliding right 1.

```
if(received_ack == expected_ack):
   sendingWindow.pop()
   win_start += 1
```

When the sending window is full and the ack of oldest frame is not received till timeout, resend the oldest frame. In this situation, the sliding window is not changed.

```
if(Event(timeout)):
   Event(resend frame win_start)
```

On the server side, use an array to store the receiving window. Record the expected sequence number. If the received frame is the expected one, send back ack. Note that the acks are also droped at a possibility of 10%.

```
if(int(ack.strip()) == expected_frame):
   if(random.random() < 0.9):
       conn.send(str(ack).encode())
```

If it is not the expected one, but it is in the receiving window, store the frame in the buffer. If the received frame is the expected one, the receiving window slide right 1.

```
if(int(ack.strip()) in receivingWindow):
    receivingWindow.remove(int(ack.strip()))
    buffer.append(text)
```

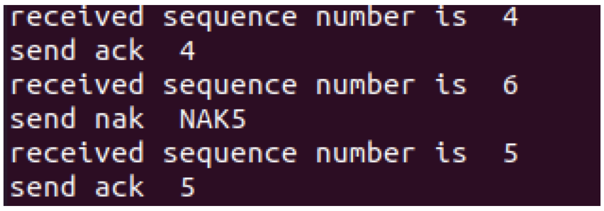
2.4 Intentially drop data frame at 10% possibility

Add a possibility of 90% before the transmission sentence at both client and server side

```
if(random.random() < 0.9):
    # Event(do transmission)
```

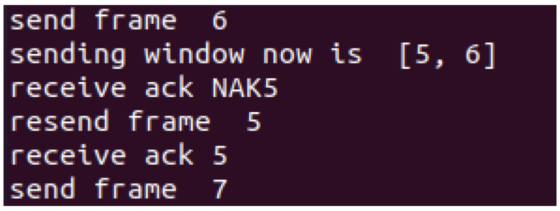
3. Results analysis

When the transmitted data is lost or corrupted, the receiver will send back a corresponding NAK.



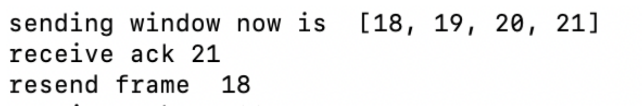
```
received sequence number is  4
send ack  4
received sequence number is  6
send nak  NAK5
received sequence number is  5
send ack  5
```

After the sender receive the nak, it will resend the data frame.



```
send frame  6
sending window now is  [5, 6]
receive ack NAK5
resend frame  5
receive ack 5
send frame  7
```

When the ACK is lost and do not reach sender side, the sliding window will slide right until reach the maximum size. At this time, the timer of the oldest frame in the sending window is timeout, so the sender will resend it.



```
sending window now is  [18, 19, 20, 21]
receive ack 21
resend frame  18
```

Link

https://jbox.sjtu.edu.cn/l/X1qVK2

Appendix

client.py

```python
import socket
import os
import time
import random
import string
import math


def client_program():
    host = socket.gethostname() # for test, should change to server ip
    port = 1234
    client_socket = socket.socket()
    client_socket.connect((host, port)) # connect to server

    print ("Client-Server Connect")
    windowSize = 4
    win_start = 1 # = ack_expected
    win_last = 4 # last frame in window
    ack_expected = 1 # store received ack
    sendingWindow = []
    filePath = "./shakespeare.txt"
    repeatFlag = 0 # flag=1 means resend win_start frame
    with open(filePath) as f:
        data = f.read()
    frameSize = math.floor(len(data)/200)# 1000 bytes per frame
    dataCnt = 0 # count the start byte index of data

    while(1):
        # check if there is sending request
        if (dataCnt < 200) and (len(sendingWindow) <= 4):
            # check if resend
            if (repeatFlag == 1): # resend win_start frame
                # check EOF
                if(len(data) - frameSize*(win_start-1) < frameSize):
                    frame = data[(win_start-1)*frameSize : len(data)]
                else:
                    frame = data[(win_start-1)*frameSize :
win_start*frameSize]
                frame = str(win_start) + ' ; ' + frame  # sequence number
= dataCnt+1
                client_socket.send(frame.encode())
                print ("resend frame ", win_start)

            else: # send new frame
                # check EOF
                if(len(sendingWindow) < 4):
                    if(len(data) - frameSize*dataCnt < frameSize):
                        frame = data[dataCnt*frameSize : len(data)]
                    else:
                        frame = data[dataCnt*frameSize :
(dataCnt+1)*frameSize]
                    dataCnt+=1
                    frame = str(dataCnt) + ' ; ' + frame  # sequence
number = dataCnt
                    if(random.random() < 0.9): # drop frame at 10%
possibility
                        client_socket.send(frame.encode())
                    if(dataCnt > 4): win_last+=1
                    sendingWindow.append(dataCnt)
                    print ("send frame ", dataCnt)
                    print ("sending window now is ", sendingWindow)
                else:
                    repeatFlag = 1
                    win_start = sendingWindow[0]
        elif (dataCnt >= 200): # all data transmitted
            break


        time.sleep(0.5)
        # receive respond
        client_socket.settimeout(1)
        try:
            ack_received = client_socket.recv(1024).decode()
        except socket.timeout:
            continue
        print ("receive ack", ack_received)
        ack_expected = sendingWindow[0]
        # check if nak
        ack_received = str(ack_received).strip()
        if(ack_received[0:3] == 'NAK'):
            repeatFlag = 1
            win_start = int(ack_received[3: len(ack_received)])
            sendingWindow = [win_start]
        else:
            if(int(ack_received.strip()) != ack_expected) and (win_last-
win_start == 4): # win_start frame is lost
                repeatFlag = 1
                sendingWindow = [win_last+1]
            elif (int(ack_received.strip()) != ack_expected) and
(win_last-win_start < 4):
                continue
            else: # correct ack is received
                if(repeatFlag == 1):
                    win_start = win_last+1
                    sendingWindow = []
                else:
                    del(sendingWindow[0])
                    if(len(sendingWindow) > 0):
                        win_start = sendingWindow[0]
                    else:
                        win_start = dataCnt+1
                repeatFlag = 0



    client_socket.close()
    print ("Socket closed")

if __name__ == '__main__':
    client_program()
```

server.py

```
import socket
import os
import time
import random
import string

def server_program():
    #host = socket.gethostname() # for test, should change to server ip
    host = "127.0.0.1"
    port = 1234
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(2)
    conn, address = server_socket.accept()  # accept new connection
    print ("Connection from: ", str(address))

    receivingWindow = [1, 2, 3, 4]
    win_last = 4
    buffer = []

    while(True):
        receivedMsg = conn.recv(1024).decode()
        if not receivedMsg:
            break
        ack, text = receivedMsg.split(' ; ', 2)
        print ("received sequence number is ", ack)
        expected_frame = receivingWindow[0]
        if(int(ack.strip()) == expected_frame):
            if(random.random() < 0.9):
                conn.send(str(ack).encode())
            print ("send ack ", ack)
            receivingWindow.remove(expected_frame)
            win_last += 1
            receivingWindow.append(win_last)
            expected_frame = receivingWindow[0]
        else:
            nak = "NAK" + str(expected_frame)
            conn.send(nak.encode())
            print("send nak ", nak)
            if(int(ack.strip()) in receivingWindow):
                receivingWindow.remove(int(ack.strip()))
                buffer.append(text)
            win_last += 1
            receivingWindow.append(win_last)


    server_socket.close()
    print ("close server")

if __name__ == '__main__':
    server_program()
```