***University of Windsor***

***Electrical and Computer Engineering***

**Project Report**

**06-88-436 Computer Communications**

**Instructor: Huapeng Wu**

**Project 1: Implementation of Go-Back-N Protocol**

**Group Member: Daksh Patel           ID: 104 030 031**

**Group Member: Nyasha Kapfumvuti ID: 104 121 166**

**Date: Mar 30th, 2018**

# Introduction

The objective of this project is to implement a Go-Back-N (GBN) protocol using Python. Sample code was given. This code was used to understand the implementation. Further research was performed to add the specifics that this project deals with. Specifics in this project include an output which demonstrates at least 10 successful frame transmissions A to B with corruption rate and loss rate of 0.2. The code shows acks, sequence numbers, ack numbers. This report shall include flowchart of the implementation, description of the functions in the code files and will display the results of the client and server. This project helps us learn how to code in python, perform networking tests and provides a better understanding of Go-Back-N protocol.

# Procedure

1. Reading the textbook for the theory portion of this project. This gave insight on how to perform the task.
   1. Transport layer provides the logical communication between the application process on different hosts.
   2. In Python there is a socket library which was imported to send and receive packets from sockets. Ports and Transport type were specified.
2. Coding for it began with testing components separately and slowly combining them.

# Discussion/Analysis

The initial part of the code in the files socket, numpy, time, json, math and random are imported. Socket provides the communication capabilities, they are the endpoints of bidirectional communication channels where channel types include UDP and TCP. JSON provides support for lightweight data interchange format inspired by JavaScript object literal syntax. Math, time, numpy and random helps support the code in various ways to allow specific functions to perform properly.

There are several global variables. Key ones are windowSize which is 10, numofPackets which is 20, lossRate provides value to indicate loss which is value of 5 to provide the 0.2 loss and indicators to get the communication working. AF\_NET four-tuple (host, port, flowinfo, scopeid) is used and TCP is specified. SOCK\_STREAM is the type of communication between two endpoints. In the client and server code the server\_address is specified to connect to and bind to, respectively. Client and server addresses are specified as ‘localhost’ with port number of 10000. The server is listening once these global variables are setup.

The server always waits for communication and prints waiting until client is connected. Data arrives as a string and needs to be converted to an array. A random integer is chosen to simulate the packet corruption when the value is equivalent to 5. The acknowledgements are specified here. If packet is not corrupted, new package is not acknowledge put it into acknowledgment.

The client has more involved with. It prepares packets to be sent. There is a function to send the data. And a function to receive acknowledgements. A timer is used to simulate gbn timeout as a delay. The send data function starts off with an empty window and unAcked packets are added to the window to be resent. New packets are added if there is room in window. New packets are fitted into current window location. One packet is sent at a time. Sequence number will increase with acks.

[Explain CLIENT more detail and maybe Serve if I missed something]

# Data/Diagrams

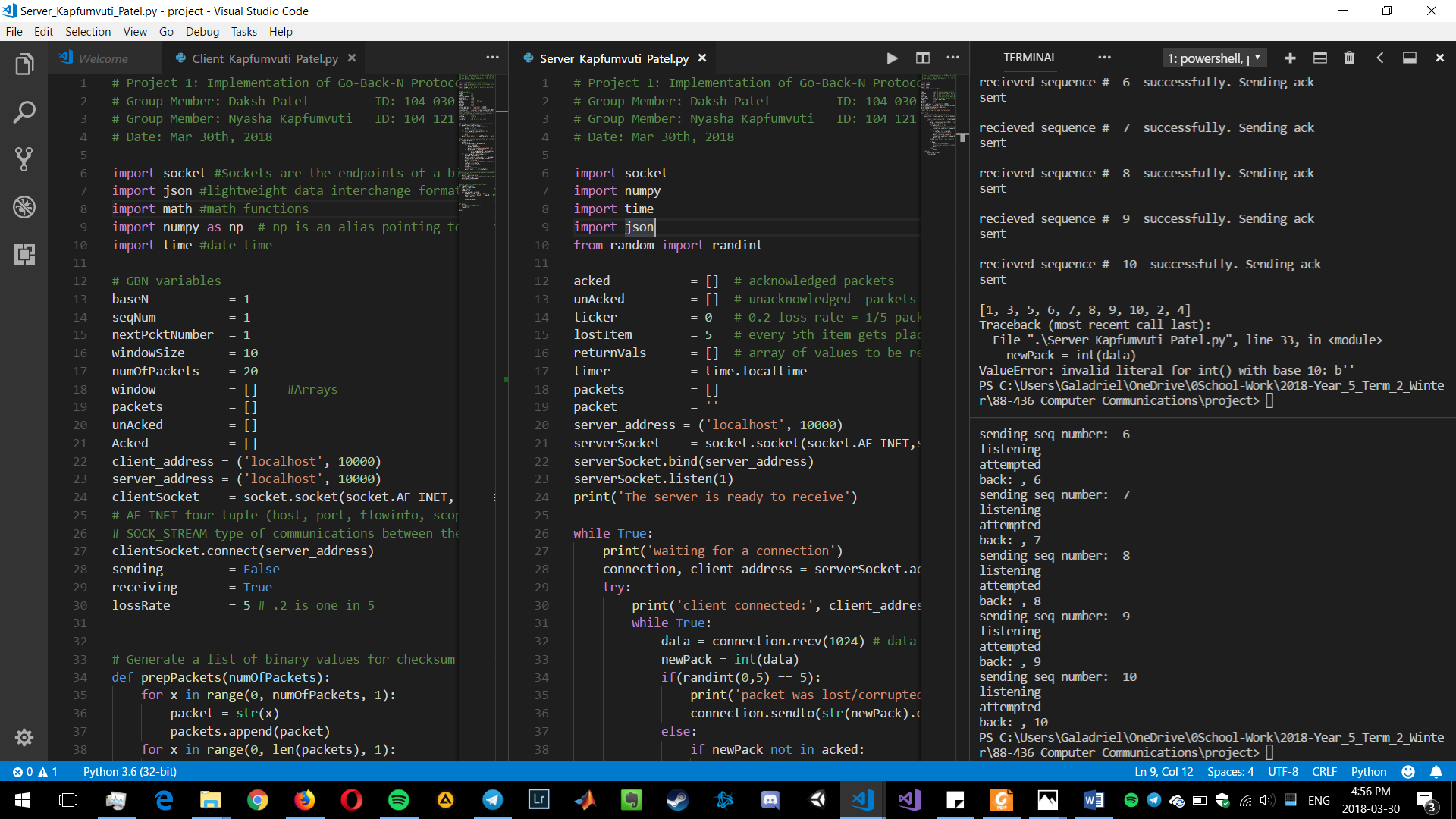


Figure 1: Code Alongside Output for Both Client and Server

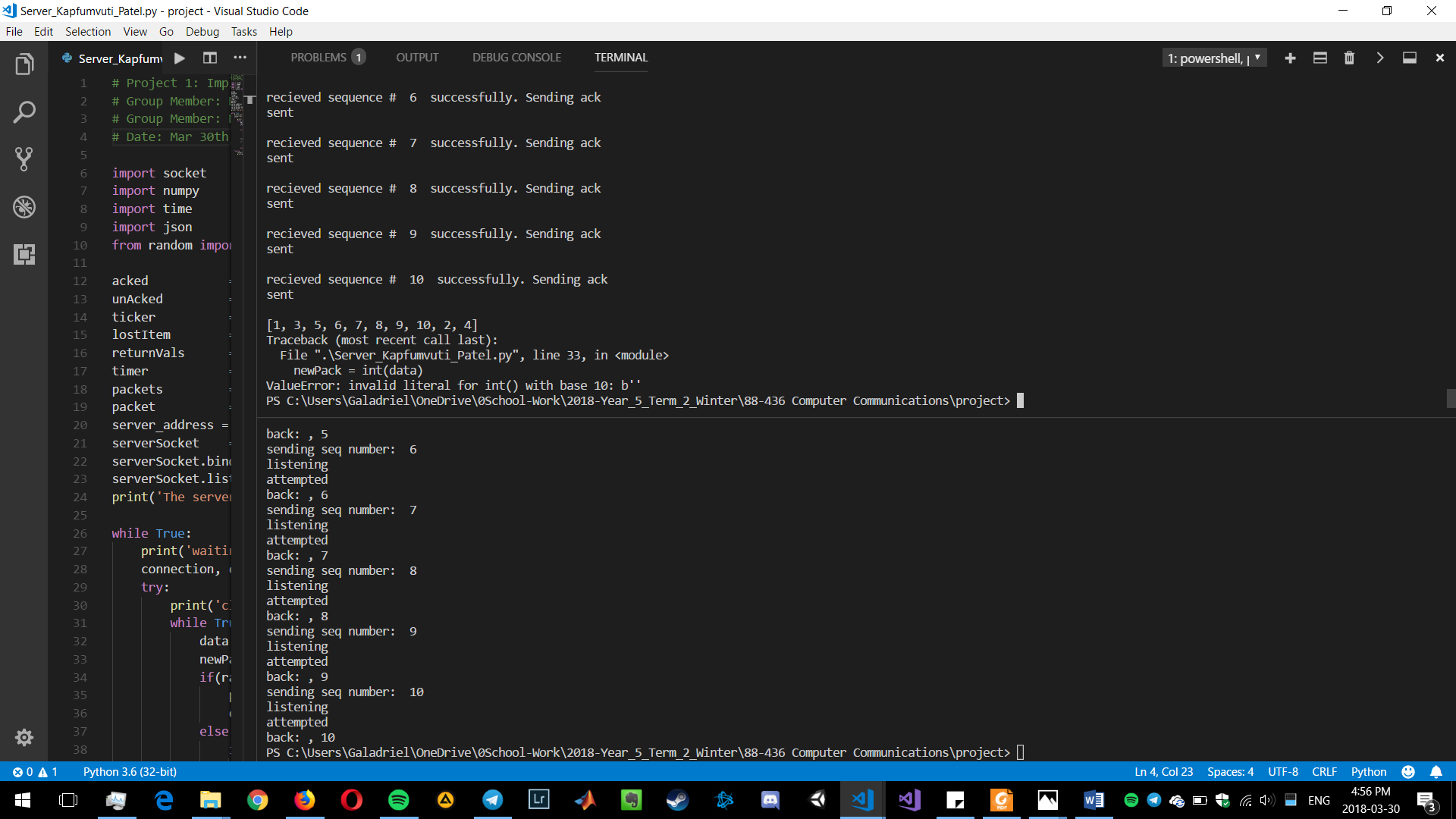


Figure 2: Ending of Output for Both Client and Server (Top is Server, Bottom is Client)

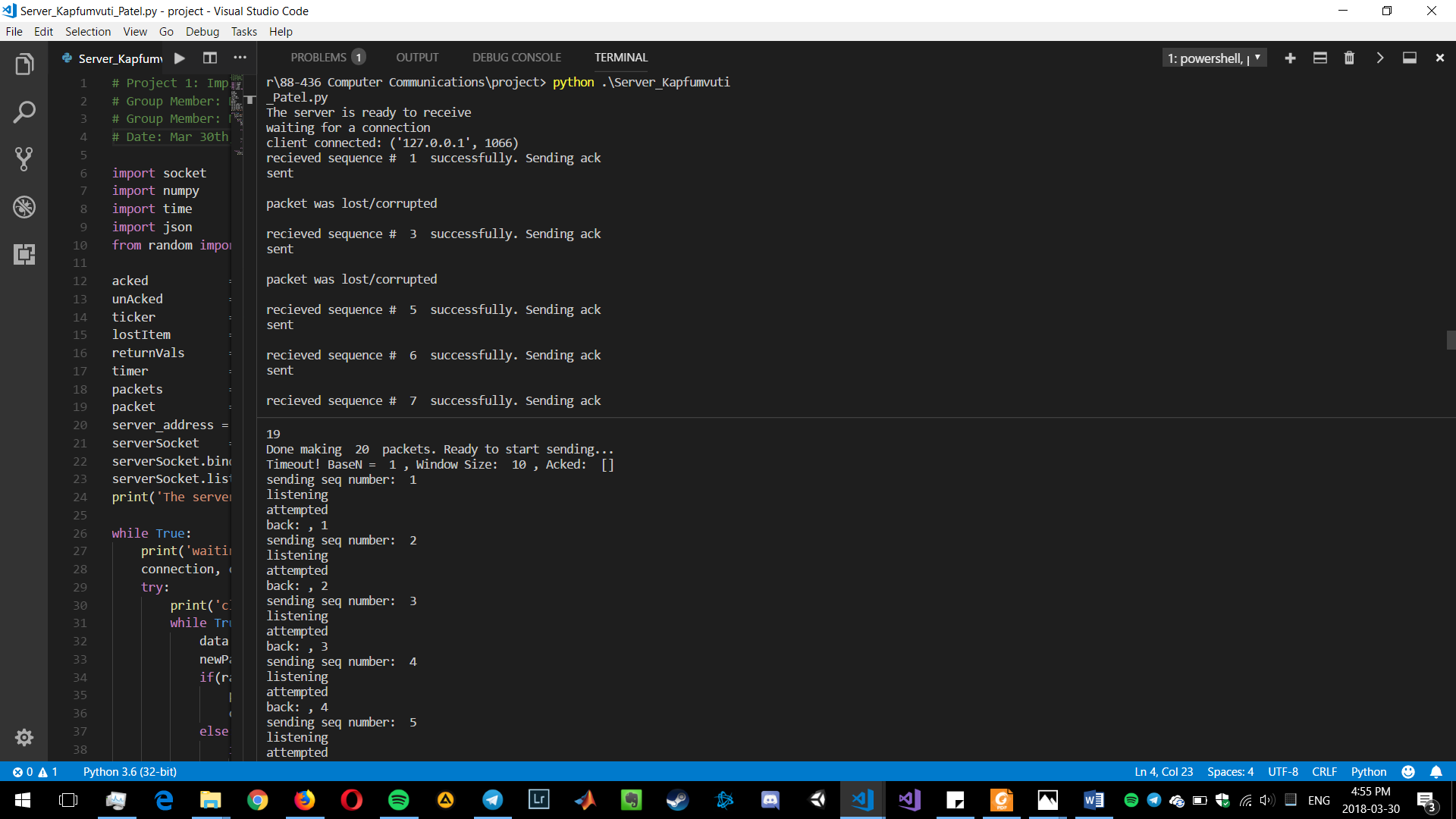


Figure 3: Beginning of Output for Both Client and Server (Top is Server, Bottom is Client)

[INSERT FLOWCHART]

# Conclusion

This project went successfully. We were able to learn more in-depth about Go-Back-N and Python. The client and server code communicate with each other and performs GBN properly. The

[DISCUSS FLOWCHART]

# Appendix

## Server Code

# Project 1: Implementation of Go-Back-N Protocol

# Group Member: Daksh Patel ID: 104 030 031

# Group Member: Nyasha Kapfumvuti   ID: 104 121 166

# Date: Mar 30th, 2018

import socket

import numpy

import time

import json

from random import randint

acked = [] # acknowledged packets

unAcked = [] # unacknowledged packets

ticker = 0 # 0.2 loss rate = 1/5 packets get "lost" => placed in unAcked

lostItem = 5 # every 5th item gets placed in unacked

returnVals = [] # array of values to be returned as acks/unacks

timer = time.localtime

packets = []

packet = ''

server\_address = ('localhost', 10000)

serverSocket = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

serverSocket.bind(server\_address)

serverSocket.listen(1)

print('The server is ready to receive')

while True:

print('waiting for a connection')

connection, client\_address = serverSocket.accept()

try:

print('client connected:', client\_address)

while True:

data = connection.recv(1024) # data arrives as a string. Need to convert this back to an array

newPack = int(data)

if(randint(0,5) == 5):

print('packet was lost/corrupted')

connection.sendto(str(newPack).encode(), server\_address)

else:

if newPack not in acked:

acked.append(newPack)

print('recieved sequence # ', str(newPack), ' successfully. Sending ack')

connection.sendto(str(newPack).encode(), server\_address)

print('sent')

ticker += 1 # loss rate leads to every nth item getting lost

if data:

# send acknowledgement

# connection.sendto(str(newPack).encode(), server\_address)

print('')

else:

break

finally:

connection.close()

print(acked)

## Client Code

# Project 1: Implementation of Go-Back-N Protocol

# Group Member: Daksh Patel ID: 104 030 031

# Group Member: Nyasha Kapfumvuti   ID: 104 121 166

# Date: Mar 30th, 2018

import socket #Sockets are the endpoints of a bidirectional communications channel. Channel types include TCP and UDP

import json #lightweight data interchange format inspired by JavaScript object literal syntax

import math #math functions

import numpy as np # np is an alias pointing to numpy. importing as np helps keep away any conflict due to namespaces

import time #date time

# GBN variables

baseN = 1

seqNum = 1

nextPcktNumber = 1

windowSize = 10

numOfPackets = 20

window = [] #Arrays

packets = []

unAcked = []

Acked = []

client\_address = ('localhost', 10000)

server\_address = ('localhost', 10000)

clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# AF\_INET four-tuple (host, port, flowinfo, scopeid) is used

# SOCK\_STREAM type of communications between the two endpoints

clientSocket.connect(server\_address)

sending = False

receiving = True

lossRate = 5 # .2 is one in 5

# Generate a list of binary values for checksum demonstration : # seq nummber, acked, data

def prepPackets(numOfPackets):

for x in range(0, numOfPackets, 1):

packet = str(x)

packets.append(packet)

for x in range(0, len(packets), 1):

print(packets[x])

pass

print('Done making ', numOfPackets, ' packets. Ready to start sending...')

# Send new and unAcked packets that are in the window

def sendData(baseN):

window = [] # empty window

while len(window) < windowSize:

for x in unAcked: # Add any unAcked packets to window for re-sending

window.append(x)

print('unacked package added', x)

if len(window) < windowSize: # Add new packets if there is room left in the window

for x in range(baseN, (windowSize+baseN), 1): # New packets fit current window location

window.append(packets[x])

for x in range(0, len(window),1): # send packet one at a time

message = window[x]

print('sending seq number: ', message) #sequence num will increase with acks

clientSocket.send(message.encode())

print('listening')

data, server = clientSocket.recvfrom(1024)

newPack = int(data)

Acked.append(newPack)

baseN += 1

print('attempted')

print('back: ,', int(newPack))

# Continually receive acknowledgements and update containers

def receiveData():

print('receiving data')

modifiedMessage = clientSocket.recvfrom(1024)

print(modifiedMessage)

clientSocket.close()

# Delay Timer to simulate GBN timeout

def timeOut(seconds):

start = time.time()

time.clock()

elapsed = 0

while elapsed < seconds:

elapsed = time.time() - start

print('Timeout! BaseN = ', baseN, ', Window Size: ', windowSize, ', Acked: ', Acked, )

time.sleep(1)

sendData(baseN)

def main():

prepPackets(numOfPackets)

timeOut(5)

main()