Computer Networks Lab Report – Assignment 2

# TITLE

**Name –** Aditya Ganguly

**Roll –** 002010501019

**Class –** BCSE 3rd year

**Group –** **A1**

**Number – 2**

## Problem Statement – Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Sender, Receiver and Channel all are independent processes. There may be multiple Transmitter and Receiver processes, but only one Channel process. The channel process introduces random delay and/or bit error while transferring frames. Define your own frame format or you may use IEEE 802.3 Ethernet frame format.

Hints: Some points you may consider in your design.

***Following functions may be required in Sender.***

**Send:** This function, invoked every time slot at the sender, decides if the sender should (1) do nothing, (2) retransmit the previous data frame due to a timeout, or (3) send a new data frame. Also, you have to consider current network time measure in time slots.

**Recv\_Ack:** This function is invoked whenever an ACK packet is received. Need to consider network time when the ACK was received, ack\_num and timestamp are the sender's sequence number and timestamp that were echoed in the ACK. This function must call the timeout function.

**Timeout:** This function should be called by ACK method to compute the most recent data packet's round-trip time and then recompute the value of timeout.

***Following functions may be required in Receiver.***

**Recv:** This function at the receiver is invoked upon receiving a data frame from the sender.

**Send\_Ack:** This function is required to build the ACK and transmit.

***Sliding window:***

The sliding window protocols (Go-Back-N and Selective Repeat) extend the stop-and-wait protocol by allowing the sender to have multiple frames outstanding (i.e., unacknowledged) at any given time. The maximum number of unacknowledged frames at the sender cannot exceed its "window size". Upon receiving a frame, the receiver sends an ACK for the frame's sequence number. The receiver then buffers the received frames and delivers them in sequence number order to the application.

***Performance metrics:*** Receiver Throughput (packets per time slot), RTT, bandwidth-delay product, utilization percentage.

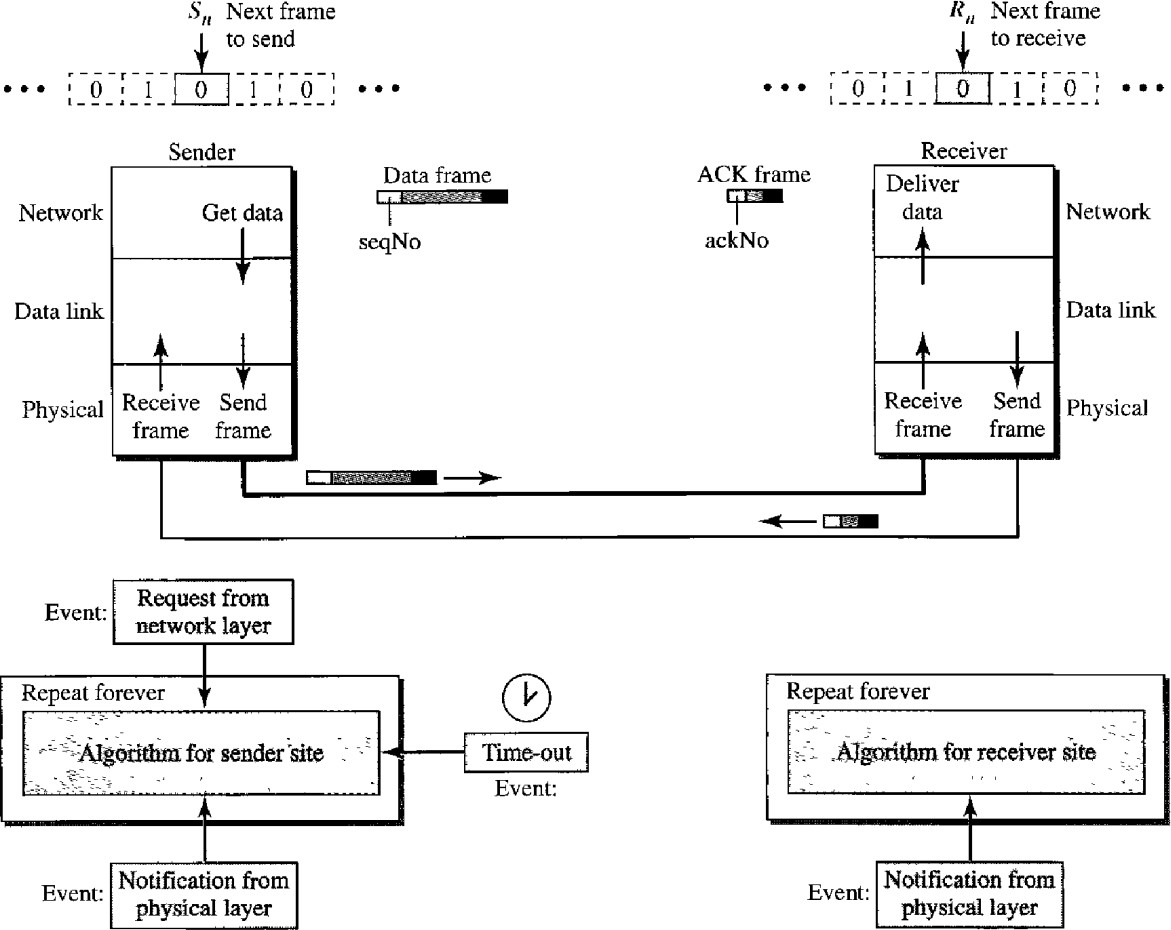
Submission date – 07/03/2019

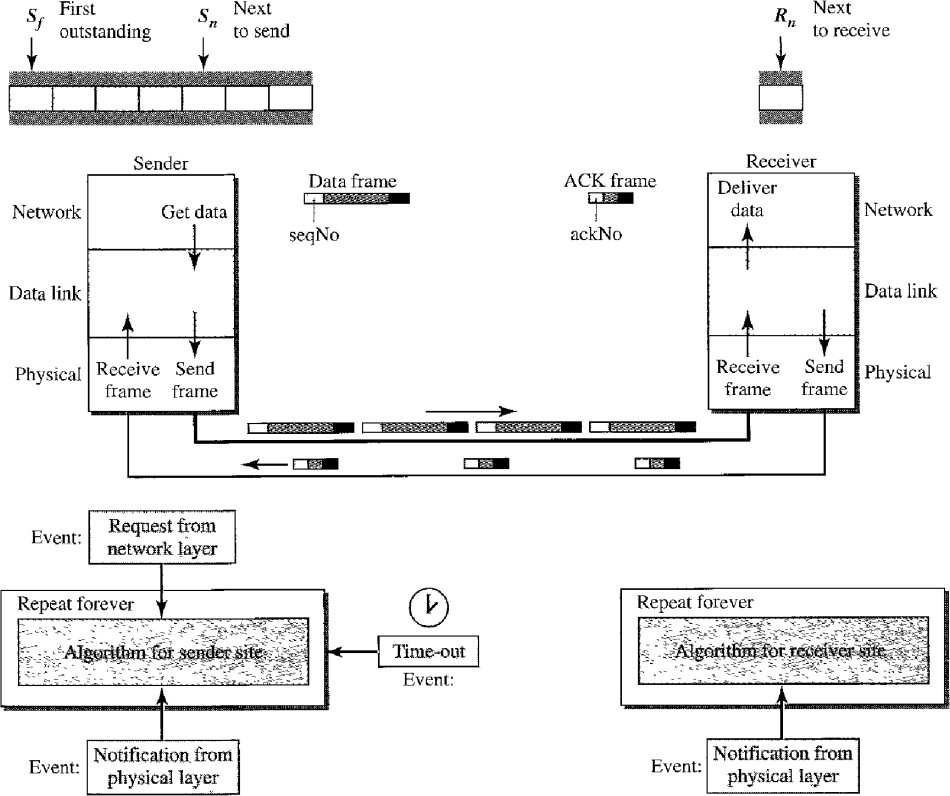
# DESIGN

We will see how the data link layer can combine framing, flow control and error control to achieve the delivery of data from one node to another. In our implementation, as the channel will be injecting errors, we are going to implement the three protocols for noisy channel.

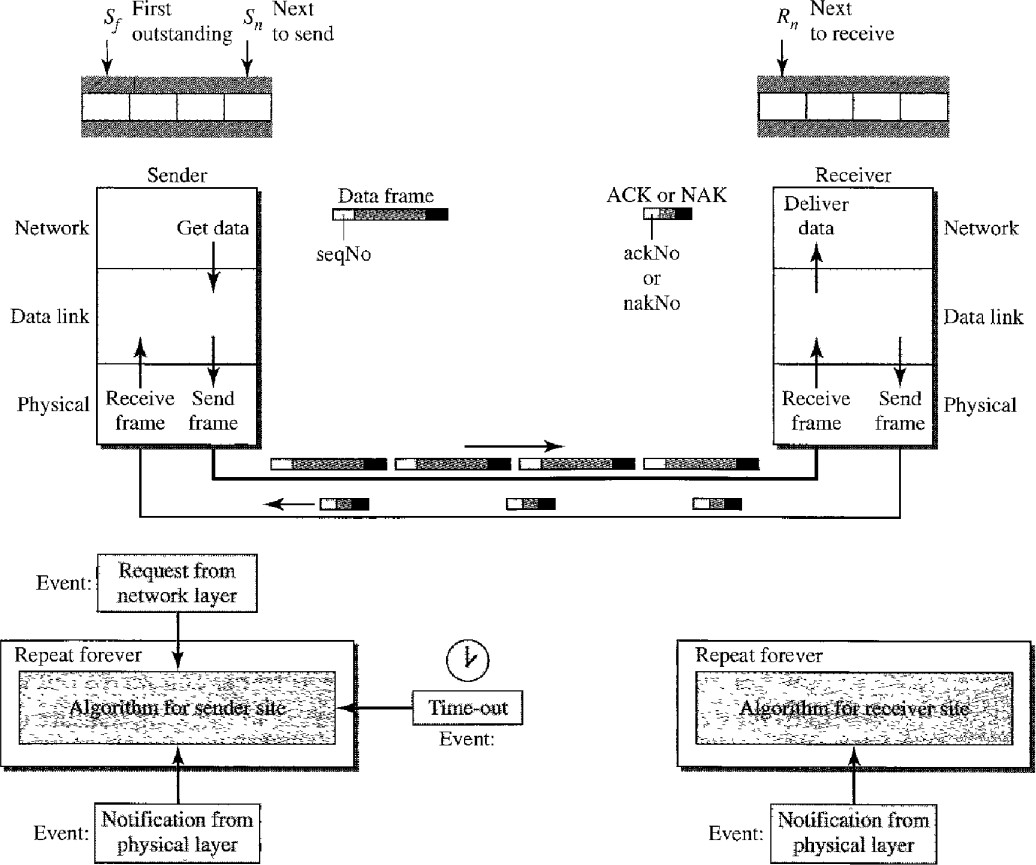
In this assignment, we shall discuss the following data link layer protocols in detail.

1. Stop and Wait protocol
2. Go Back N Sliding Window protocol
3. Selective Repeat Sliding Window protocol

DESIGN OF STOP AND WAIT PROTOCOL:

DESIGN OF GO BACK N SLIDING WINDOW PROTOCOL:

DESIGN OF SELECTIVE REPEAT SLIDING WINDOW PROTOCOL:



I have implemented the error detection module in three program files.

* **sender.py** (Sender program (we can create multiple sender processes))
* **channel.py** (Program for single channel process)
* **receiver.py** (Receiver program (we can create multiple receiver processes))

The individual files fulfils different assignment purposes, following which have been explained in details :

1. **sender.py** – The following are the tasks performed in this Sender program :
   1. We can create more than one sender processes, to send a message to channel.
   2. It first waits for the user to input a binary input string.
   3. The appropriate frame is created using the above input string.
   4. This frame is then sent to channel.
   5. It waits for a ACK/NAK to be received from channel, notifying the successful delivery of binary input message.
   6. If it does not receives a ACK/NAK for a time period of 2s, it resends again(according to the protocols defined).
2. **channel.py** – The following are the tasks performed in this Channel program :
   1. The channel process first takes number of senders and receivers as input.
   2. It initiates and connects all the sender and receiver processes.
   3. It receives the frame from any of the current senders.
   4. It then injects error randomly into the data frame.
   5. Then the frame is sent to one of the receiver process.
   6. The receiver then sends a ACK/NAK for the data received, to the channel.
   7. The channel then passes the ACK/NAK status to its corresponding sender.
3. **receiver.py** – The following are the tasks performed in this Receiver program :
   1. The receiver process first waits for a message to be received from channel.
   2. It then adds a random amount of time delay, before the message is sent back to its channel.
   3. It checks for any error in the data frame received, and sends a message ACK/NAK accordingly.
   4. The above message is then sent to channel.

# IMPLEMENTATION

## STOP AND WAIT PROTOCOL

**Code Snippet of sender.py:**

import socket

import sys

import time

from threading import Thread

from collections import deque

def CreateFrame(data):

ones = 0

for i in data:

if i == '1':

ones += 1

data += str(ones % 2)

return data

class Sender():

def \_\_init\_\_(self):

self.host = '127.0.0.1'

self.port = 8080

self.socket = socket.socket()

self.socket.connect((self.host,self.port))

self.socket.settimeout(2)

def send(self):

while True:

data = input("Enter the data: ")

if data=='exit':

break

data = CreateFrame(data)

timeout = True

while timeout:

prev = time.time()

timeout = False

self.socket.send(data.encode())

try:

ret = self.socket.recv(1024).decode()

except Exception:

pass

curr=time.time()

if curr-prev>2:

timeout=True

if timeout:

print("Timeout occured")

print("Sent the data")

self.socket.close()

return

if \_\_name\_\_=='\_\_main\_\_':

s = Sender()

s.send()

## Code Snippet of receiver.py:

import socket

import sys

import time

import random

from threading import Thread

def Wait():

x = random.randint(0, 5)

if x <= 1:

time.sleep(0.2)

return

def CheckError(data):

parity = 0

for i in data:

if i == '1':

parity += 1

return parity % 2

def Extract(data):

startind = -1

endind = -1

for i in range(len(data)):

if data[i]=='/':

endind=i

if startind==-1:

startind=i+1

return data[0:startind-1],data[startind:endind]

class Receiver():

def \_\_init\_\_(self):

self.host = '127.0.0.2'

self.port = 9090

self.socket = socket.socket()

self.socket.connect((self.host,self.port))

def receive(self):

while True:

data = self.socket.recv(1024).decode()

if data[0:len(data)-1]=='exit':

break

error=CheckError(data)

Wait()

if error==1:

print("Error occured")

time.sleep(2.2)

else:

print("Frame received")

ret="ACK"

self.socket.send(ret.encode())

self.socket.close()

return

if \_\_name\_\_=='\_\_main\_\_':

r = Receiver()

r.receive()

## Code Snippet of channel.py:

import socket

import time

import subprocess

import random

import os

from threading import Thread

def Noise(frame):

p = random.random()

if p < 0.5:

pos = random.randint(0, len(frame)-1)

frame = frame[:pos]+'1'+frame[pos+1:]

return frame

def Extract(data):

startind = -1

endind = -1

for i in range(len(data)):

if data[i] == '/':

endind = i

if startind == -1:

startind = i+1

return data[0:startind-1], data[startind:endind]

class Channel():

def \_\_init\_\_(self, sendno, recvno):

self.sendno = sendno

self.recvno = recvno

self.sendhost = '127.0.0.1'

self.sendport = 8080

self.sendconn = []

self.recvhost = '127.0.0.2'

self.recvport = 9090

self.recvconn = []

def InitializeSenders(self):

sock = socket.socket()

sock.bind((self.sendhost, self.sendport))

sock.listen(self.sendno)

for i in range(self.sendno):

conn = sock.accept()

self.sendconn.append(conn)

def InitializeReceivers(self):

sock = socket.socket()

sock.bind((self.recvhost, self.recvport))

sock.listen(self.recvno)

for i in range(self.recvno):

conn = sock.accept()

conn[0].settimeout(2)

self.recvconn.append(conn)

def CloseSenders(self):

for conn in self.sendconn:

conn[0].close()

def CloseReceivers(self):

for conn in self.recvconn:

conn[0].close()

def run(self):

while True:

conn = self.sendconn[0]

data=conn[0].recv(1024).decode()

if data[0:len(data)-1] == 'exit':

break

data=Noise(data)

rconn = random.choice(self.recvconn)

rconn[0].send(data.encode())

time.sleep(0.001)

try:

ret = rconn[0].recv(1024).decode()

conn[0].send(ret.encode())

except Exception:

pass

return

if \_\_name\_\_=='\_\_main\_\_':

c = Channel(1,1)

c.InitializeSenders()

c.InitializeReceivers()

c.run()

c.CloseSenders()

c.CloseReceivers()

## GO BACK N SLIDING WINDOW PROTOCOL

**Code Snippet of sender.py:**

window\_size=3

count=0

def createFrame(data):

countOnes = 0

for ch in data:

if ch == '1':

countOnes += 1

data += str(countOnes % 2)

return data

def return\_seqno(data):

return int(data.split('/')[-1])

def return\_data(data):

return data.split('/')[0]

def take\_input():

global count

global total\_data

global window\_size

print("Enter q to stop")

data=input()

while data!="q":

frame=createFrame(data)

frame=frame+'/'+str(count)

count=(count+1)%window\_size

total\_data.append(frame)

data = input()

return

def keep\_sending(mySocket):

while True:

global sentframes

global total\_data

global window\_size

if len(sentframes)==0:

temp=copy.deepcopy(total\_data)

for i in range(min(window\_size,len(total\_data))):

f=temp[0]

seqno=return\_seqno(f)

print(f"Sent frame {seqno}")

sentframes.append(f)

temp.popleft()

time.sleep(0.2)

mySocket.send(f.encode())

total\_data=temp

def do\_receive(mySocket):

while True:

global returnlist

ret=mySocket.recv(1024).decode()

ret = ret.split(":")

ret=ret[0:-1]

returnlist.extend([l.split('/')[0] for l in ret])

def try\_resending(mySocket):

while True:

global returnlist

global sentframes

if len(returnlist)==len(sentframes):

temp=copy.deepcopy(returnlist)

timeout=False

for t in temp:

if t=="timeout":

timeout=True

break

else:

returnlist.popleft()

fr=sentframes[0]

print(f"Successfully Sent frame {return\_seqno(fr)}")

sentframes.popleft()

if timeout:

print("Resending required")

returnlist = deque()

for fr in sentframes:

mySocket.send(fr.encode())

time.sleep(2)

print(f"Resending frame {return\_seqno(fr)}")

time.sleep(0.2)

time.sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

host = '127.0.0.1'

port = 8080

mySocket = socket.socket()

mySocket.connect((host, port))

count = 0

sn = 0

t1=threading.Thread(target=do\_receive,args=[mySocket])

t2=threading.Thread(target=try\_resending,args=[mySocket])

t3=threading.Thread(target=keep\_sending,args=[mySocket])

take\_input()

t3.start()

t1.start()

t2.start()

t1.join()

t2.join()

t3.join()

## Code Snippet of receiver.py:

import socket

import sys

import time

import random

from threading import Thread

def Wait():

x = random.randint(0, 5)

if x <= 1:

time.sleep(0.2)

return

def CheckError(data):

parity = 0

for i in data:

if i == '1':

parity += 1

return parity % 2

print('initiating receiver ')

host = '127.0.0.2'

port = 9090

mySocket=socket.socket()

mySocket.connect((host,port))

rn=0

def receive(mySocket):

lis=[]

while True:

r = mySocket.recv(1024).decode()

data=r.split('/')[0]

seqno=int(r.split('/')[-1])

error=CheckError(data)

print(f"Received frame {seqno}")

if error==1:

# time.sleep(0.5)

ret="timeout"+'/'+str(rn)+":"

print(f"Error in frame {seqno}")

mySocket.send(ret.encode())

else:

#time.sleep(0.5)

ret="ack"+"/"+str(rn)+":"

mySocket.send(ret.encode())

if \_\_name\_\_=='\_\_main\_\_':

receive(mySocket)

## Code Snippet of channel.py:

**Same as Stop and Wait**

## SELECTIVE REPEAT SLIDING WINDOW PROTOCOL

**Code Snippet of sender.py:**

## import socket

## import sys

## import time

## import random

## from threading import Thread

## def Wait():

## x = random.randint(0, 5)

## if x <= 1:

## time.sleep(0.2)

## return

## import socket

## import sys

## import time

## import copy

## from collections import deque

## import threading

## sentframes = dict()

## returnlist = dict()

## total\_data = deque()

## window\_size = 4

## count = 0

## nak\_count=1000

## sending=False

## def createFrame(data):

## countOnes = 0

## for ch in data:

## if ch == '1':

## countOnes += 1

## data += str(countOnes % 2)

## return data

## def return\_seqno(data):

## return int(data.split('/')[-1])

## def return\_data(data):

## return data.split('/')[0]

## def take\_input():

## global count

## global total\_data

## global window\_size

## print("Enter q to stop")

## data = input()

## while data != "q":

## frame = createFrame(data)

## frame = frame+'/'+str(count)

## count = (count+1) % window\_size

## total\_data.append(frame)

## data = input()

## return

## def keep\_sending(mySocket):

## while True:

## global sentframes

## global total\_data

## global window\_size

## global nak\_count

## global sending

## if len(sentframes.keys()) == 0:

## temp = copy.deepcopy(total\_data)

## nak\_count = min(window\_size, len(total\_data))

## sending=True

## for i in range(min(window\_size, len(total\_data))):

## f = temp[0]

## seqno = return\_seqno(f)

## print(f"Sent frame {seqno}")

## sentframes[seqno]=[f,"not\_received"]

## temp.popleft()

## time.sleep(0.2)

## mySocket.send(f.encode())

## sending=False

## total\_data = temp

## def do\_receive(mySocket):

## while True:

## if not sending:

## global returnlist

## ret = mySocket.recv(1024).decode()

## ret = ret.split(":")

## ret = ret[0:-1]

## # print(ret)

## for r in ret:

## returnlist[return\_seqno(r)]=r

## def try\_resending(mySocket):

## while True:

## global returnlist

## global sentframes

## global nak\_count

## global sending

## if len(returnlist.keys()) == nak\_count:

## nak\_count=0

## temp = copy.deepcopy(returnlist)

## nak\_received = False

## for seqno,frame in temp.items():

## if return\_data(frame)=='nak':

## nak\_received=True

## else:

## try:

## sentframes[seqno][1]='received'

## except Exception:

## pass

## if nak\_received:

## print("Resending required")

## returnlist = dict()

## nak\_received = False

## sending=True

## senttemp = copy.deepcopy(sentframes)

## for seqno,lis in senttemp.items():

## if lis[1]=='not\_received':

## nak\_count+=1

## nak\_received=True

## print(f"Resent frame {seqno}")

## time.sleep(2)

## mySocket.send(sentframes[seqno][0].encode())

## elif nak\_received==False:

## del sentframes[seqno]

## print(f"Successfully sent frame {seqno}")

## sending=False

## time.sleep(1)

## if \_\_name\_\_ == '\_\_main\_\_':

## host = '127.0.0.1'

## port = 8080

## mySocket = socket.socket()

## mySocket.connect((host, port))

## count = 0

## sn = 0

## t1 = threading.Thread(target=do\_receive, args=[mySocket])

## t2 = threading.Thread(target=try\_resending, args=[mySocket])

## t3 = threading.Thread(target=keep\_sending, args=[mySocket])

## take\_input()

## t3.start()

## t1.start()

## t2.start()

## t1.join()

## t2.join()

## t3.join()

## Code Snippet of receiver.py:

## import socket

## import sys

## import time

## import random

## from threading import Thread

## rn=0

## def Wait():

## x = random.randint(0, 5)

## if x <= 1:

## time.sleep(0.2)

## return

## def CheckError(data):

## parity = 0

## for i in data:

## if i == '1':

## parity += 1

## return parity % 2

## print('initiating receiver ')

## host = '127.0.0.2'

## port = 9090

## mySocket = socket.socket()

## mySocket.connect((host, port))

## window\_size=4

## rn = 0

## def receive(mySocket):

## lis = []

## global rn

## while True:

## r = mySocket.recv(1024).decode()

## data = r.split('/')[0]

## seqno = int(r.split('/')[-1])

## error = CheckError(data)

## print(f"Received frame {seqno}")

## if error == 1:

## ret = "nak"+'/'+str(rn)+":"

## print(f"Error in frame {seqno}")

## rn=(rn+1)%window\_size

## mySocket.send(ret.encode())

## else:

## ret = "ack"+"/"+str(rn)+":"

## rn = (rn+1) % window\_size

## mySocket.send(ret.encode())

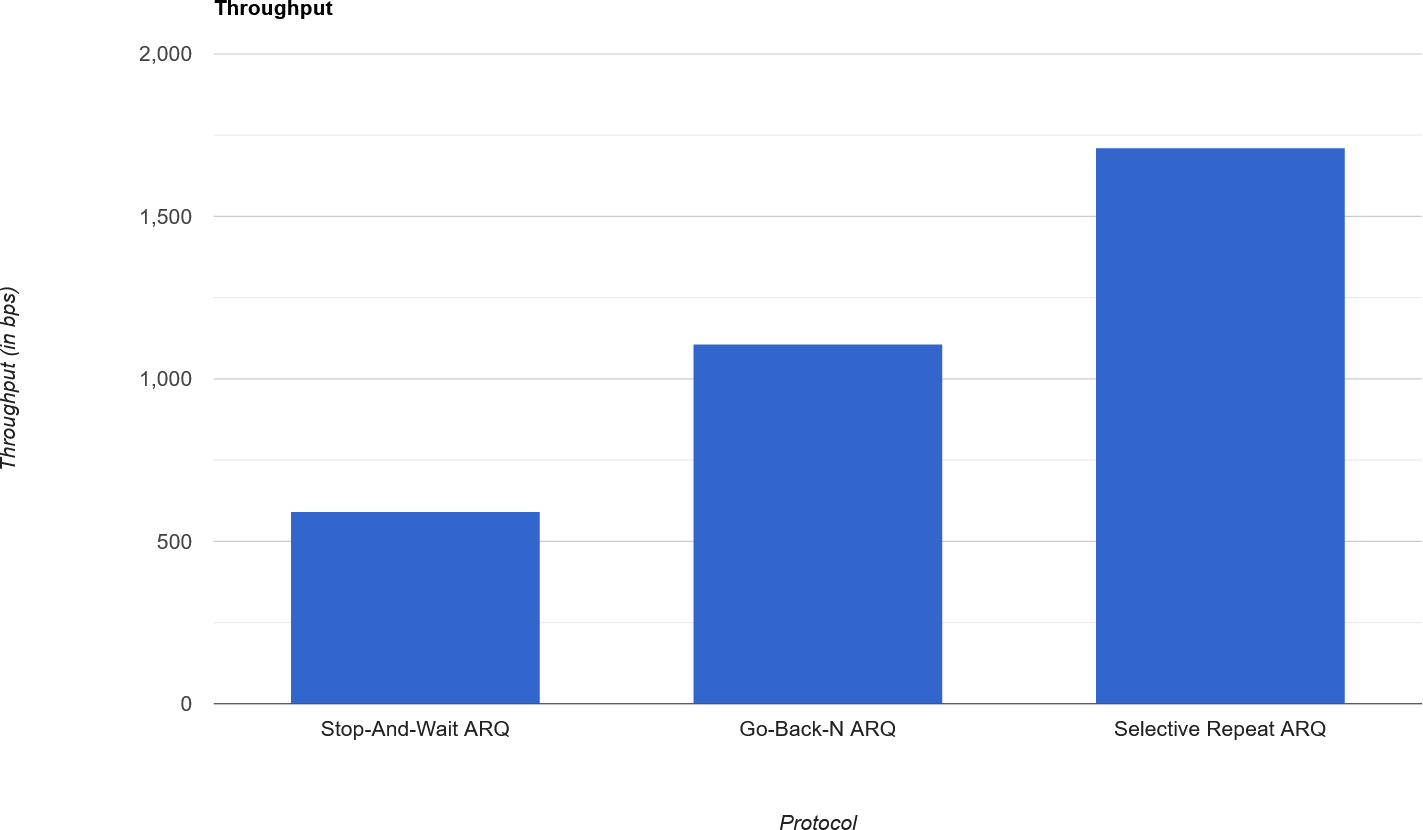
## if \_\_name\_\_ == '\_\_main\_\_':

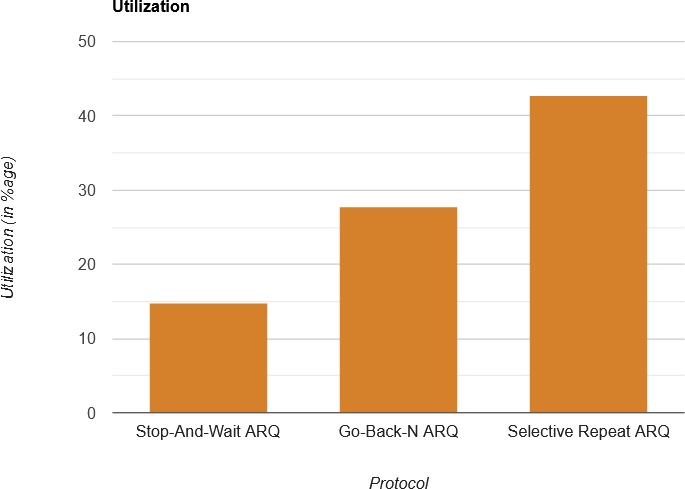
## receive(mySocket)

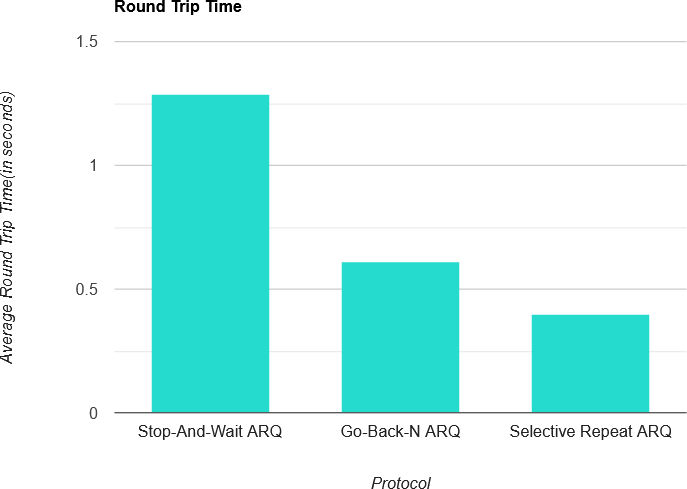
## Code Snippet of channel.py:

**Same as Stop and Wait**

**RESULTS**







Stop-And-Wait is memory efficient as the sequence numbers are only 0 and 1 and thus, keeps a copy of just 1 sent frame. If the channel is thick and long, the potential of the channel is wasted because we are just waiting for an ACK from the receiver, whereas we could have

sent a few packets lined up next too at the same time. This would boost the throughput to a great extent.

The need to utilize more of the channel brings us to Go-Back-N ARQ, where we send many frames before waiting for ACK. This ensures that many frames are in transit at the same time, which is desired when the bandwidth-delay product is high. But here the receiver needs to accept the frames in order. So a timer is maintained on the sender side to resend the frames, in case the frame or ACK was lost during transit and thus the frame was either not acknowledged or the sender didn’t receive the ACK. Whenever such happens, all the frames from the last acknowledged frames are resent by the sender.

In Selective Repeat ARQ, multiple frames are in transit and the channel is also utilized well. The improvement here is that the receiver can accept the frames in any order. It just needs to make sure that the data is delivered to the file accurately. As a result, the frames within a window can be acknowledged in any order. 1 NAK can inform regarding the last missing packet and 1 ACK can serve as ACK for the previously received ACKs as well because an ACK is transferred only when the frames are converted in order to message and delivered to the file. This releases contention on the channel. But the out-of-order hack necessitates individual timers, so more memory overhead is present on the sender side and special care must be given to synchronization issues.

**COMMENTS**

For this assignment, I understood the working of various data link layer flow control protocols. I also understood the advantages and disadvantages of each of them.