Computer Networks Lab Report – Assignment 2

# TITLE

**Name –** Sourav Dutta

**Roll –** 001610501076

**Class –** BCSE 3rd year **Group –** A3 **Assignment 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.

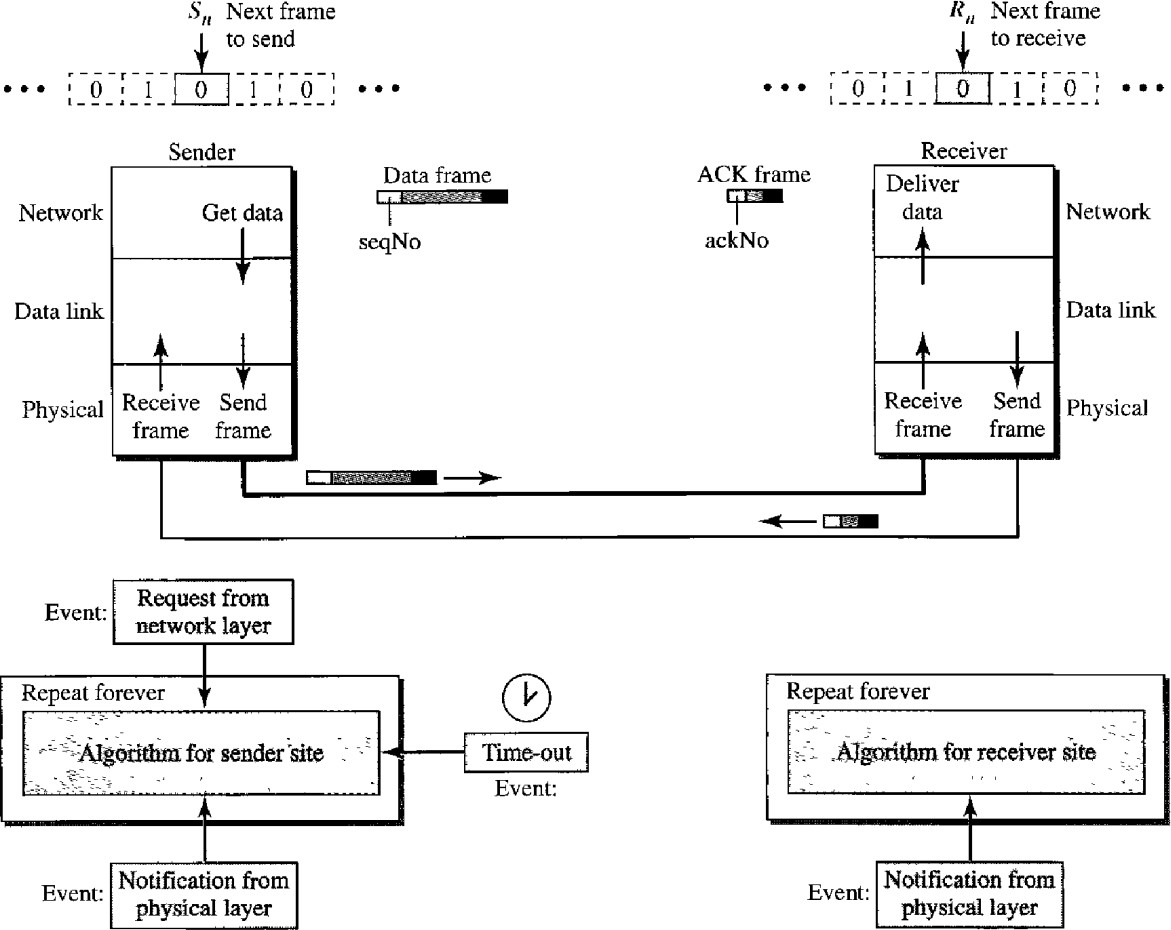
Submission date – 07/03/2019

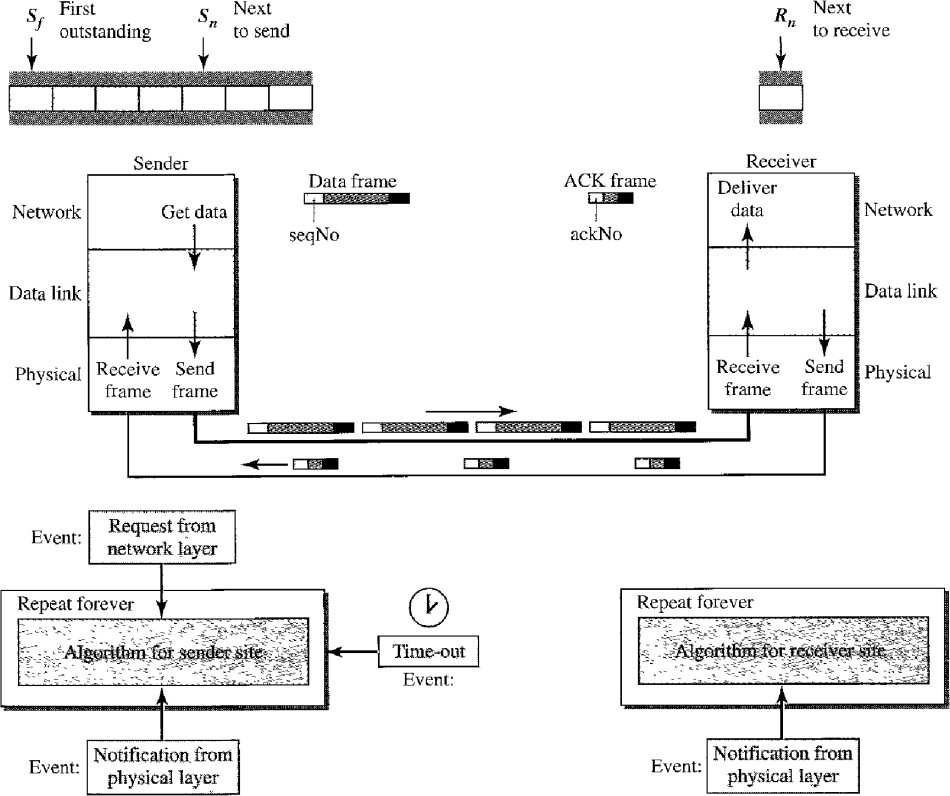
# DESIGN

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.

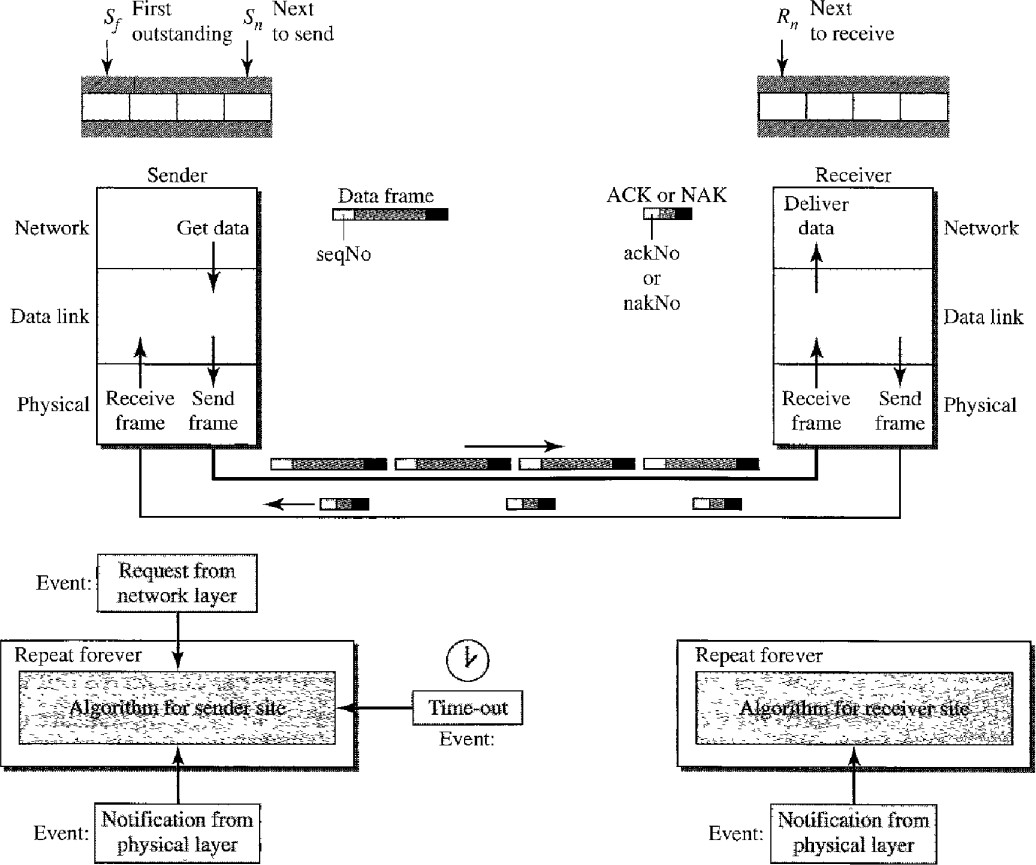
In this assignment, we shall discuss the following data link layer protocols in detail.

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

def createFrame(data): countOnes = 0 for ch in data:

if ch == '1':

countOnes += 1 data += str(countOnes%2) return data

def Main(senderno):

print('Initiating Sender #',senderno) host = '127.0.0.1'

port = 8080

mySocket = socket.socket()

mySocket.connect((host, port))

while True:

print()

data = input("Enter $ ") prevtime = time.time() data = createFrame(data)

print('Sending to channel :',str(data)) mySocket.send(data.encode())

if not data:

break

if data == 'q0': break

rdata = mySocket.recv(1024).decode() print('Received from channel :',str(rdata)) curtime = time.time()

print('Round trip time: ',str(curtime-prevtime)) if curtime-prevtime > 2:

timeout = 1

else:

timeout = 0

fileout = open('checktime.txt', "w") fileout.write(str(timeout)) fileout.close()

while timeout==1:

print()

prevtime = time.time() if timeout == 1:

print('TIMEOUT of 2s EXPIRED !!')

else:

print('THE FRAME GOT CORRUPTED !!!')

print('Again Sending to channel :',str(data)) mySocket.send(data.encode())

rdata = mySocket.recv(1024).decode() print('Again Received from channel :',str(rdata)) curtime = time.time()

print('Round trip time:',str(curtime-prevtime),'seconds') #print('Bandwidth-delay product:',str((curtime-

prevtime)\*8),'bits/seconds')

if curtime-prevtime > 2: timeout = 1

else:

timeout = 0

fileout = open('checktime.txt', "w") fileout.write(str(timeout)) fileout.close()

mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of receiver.py:

import socket import sys import time import random

def waitRandomTime():

x = random.randint(0,5) if x <= 1:

time.sleep(2)

def checkError(frame): countOnes = 0 for ch in frame:

if ch == '1':

countOnes += 1 return countOnes%2

def Main(senderno):

print('Initiating Receiver #',senderno) host = '127.0.0.2'

port = 9090

mySocket = socket.socket() mySocket.connect((host, port))

while True:

print()

data = mySocket.recv(1024).decode() if not data:

break

if data == 'q0': break

print('Received from channel :', str(data)) waitRandomTime()

if checkError(data) == 0: rdata = 'ACK'

else:

time.sleep(2) rdata = 'TIMEOUT'

print('Sending to channel :',str(rdata)) mySocket.send(rdata.encode())

mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of channel.py:

import socket import time import subprocess import random import os

def injectRandomError(frame):

pos = random.randint(0, len(frame)-1) frame = frame[:pos]+'1'+frame[pos+1:] return frame

class Channel():

def init (self, totalsender, totalreceiver):

self.totalsender = totalsender self.senderhost = '127.0.0.1'

self.senderport = 8080 self.senderconn = []

self.totalreceiver = totalreceiver self.receiverhost = '127.0.0.2'

self.receiverport = 9090 self.receiverconn = []

def initSenders(self):

senderSocket = socket.socket() senderSocket.bind((self.senderhost, self.senderport)) senderSocket.listen(self.totalsender)

for i in range(1, self.totalsender+1): conn = senderSocket.accept() self.senderconn.append(conn)

print('Initiated all sender connections')

def closeSenders(self):

for conn in self.senderconn: conn[0].close()

print('Closed all sender connections')

def initReceivers(self): receiverSocket = socket.socket()

receiverSocket.bind((self.receiverhost, self.receiverport)) receiverSocket.listen(self.totalreceiver)

for i in range(1, self.totalreceiver+1): conn = receiverSocket.accept() self.receiverconn.append(conn)

print('Initiated all receiver connections')

def closeReceivers(self):

for conn in self.receiverconn: conn[0].close()

print('Closed all receiver connections')

def processData(self): while True:

for i in range(len(self.senderconn)): print()

conn = self.senderconn[i]

data = conn[0].recv(1024).decode() if not data:

break

if data == 'q0': break

print('Received from Sender',i+1,':',str(data))

recvno = random.randint(0,len(self.receiverconn)-1) print('Sending to Receiver',recvno+1)

rconn = self.receiverconn[recvno] data = injectRandomError(data)

rconn[0].sendto(data.encode(), rconn[1])

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

print('Received from Receiver',recvno+1,':', str(rdata))

print('Sending to Sender',i+1) conn[0].send(rdata.encode())

time.sleep(0.002)

filein = open('checktime.txt',"r") timeout = int(filein.read()) filein.close() os.remove('checktime.txt') print(timeout)

while timeout==1:

print()

data = conn[0].recv(1024).decode() print('Again Received from

Sender',i+1,':',str(data))

str(rdata))

data = injectRandomError(data) print('Again Sending to Receiver',recvno+1) rconn[0].sendto(data.encode(), rconn[1]) rdata = rconn[0].recv(1024).decode()

print('Again Received from Receiver',recvno+1,':',

print('Again Sending to Sender',i+1) conn[0].send(rdata.encode())

time.sleep(0.002)

filein = open('checktime.txt',"r") timeout = int(filein.read()) filein.close() os.remove('checktime.txt') print(timeout)

if data == 'q0': break

return

if name == ' main ':

totalsen = int(input('Enter number of senders: ')) totalrecv = int(input('Enter number of receivers: '))

ch = Channel(totalsen, totalrecv) ch.initSenders() ch.initReceivers() ch.processData() ch.closeSenders() ch.closeReceivers()

## GO BACK N SLIDING WINDOW PROTOCOL

**Code Snippet of sender.py:**

import socket import sys import time

def createFrame(data): countOnes = 0 for ch in data:

if ch == '1':

countOnes += 1 data += str(countOnes%2) return data

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1:

endidx = i break

return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1: startidx = i+1

break return frame[startidx:]

def Main(senderno):

count = 0 sentframes = []

print('Initiating Sender #',senderno) host = '127.0.0.1'

port = 8080

mySocket = socket.socket() mySocket.connect((host, port))

while True:

print()

data = input("Enter $ ") #prevtime = time.time()

data = createFrame(data) + '/' + str(count) + '/' msg = extractMessage(data)

print('Sending to channel :',str(msg)) mySocket.send(data.encode()) sentframes.append(data)

count += 1

if not msg:

break

if msg == 'q0':

break mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of receiver.py:

import socket import sys import time import random

def waitRandomtime():

x = random.randint(0,5) if x <= 1:

time.sleep(2)

def checkError(frame): countOnes = 0 for ch in frame:

if ch == '1':

countOnes += 1 return countOnes%2

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1: endidx = i

break return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1: startidx = i+1

break return frame[startidx:]

def Main(senderno):

print('Initiating Receiver #',senderno) host = '127.0.0.2'

port = 9090

mySocket = socket.socket() mySocket.connect((host, port))

while True:

print()

data = mySocket.recv(1024).decode() data = str(data)

msg = extractMessage(data) if not msg:

break

if msg == 'q0':

break

print('Received from channel :', str(data)) waitRandomtime()

if checkError(msg) == 0: rdata = 'ACK'

else:

rdata = 'NAK'

print('Sending to channel :',str(rdata)) mySocket.send(rdata.encode())

mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of channel.py:

import socket import time import subprocess import random import os

def injectRandomError(frame):

pos = random.randint(0, len(frame)-1) frame = frame[:pos]+'1'+frame[pos+1:] return frame

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1: endidx = i

break return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1:

startidx = i+1 break

return frame[startidx:] class Channel():

def init (self, totalsender, totalreceiver, windowsize): self.totalsender = totalsender

self.senderhost = '127.0.0.1'

self.senderport = 8080 self.senderconn = []

self.totalreceiver = totalreceiver self.receiverhost = '127.0.0.2'

self.receiverport = 9090 self.receiverconn = []

self.windowsize = windowsize self.slidingwindow = [] self.currentcount = 0 #self.statuswindow = []

def initSenders(self):

senderSocket = socket.socket() senderSocket.bind((self.senderhost, self.senderport)) senderSocket.listen(self.totalsender)

for i in range(1, self.totalsender+1): conn = senderSocket.accept() self.senderconn.append(conn)

print('Initiated all sender connections')

def closeSenders(self):

for conn in self.senderconn: conn[0].close()

print('Closed all sender connections')

def initReceivers(self): receiverSocket = socket.socket()

receiverSocket.bind((self.receiverhost, self.receiverport)) receiverSocket.listen(self.totalreceiver)

for i in range(1, self.totalreceiver+1): conn = receiverSocket.accept() self.receiverconn.append(conn)

print('Initiated all receiver connections')

def closeReceivers(self):

for conn in self.receiverconn: conn[0].close()

print('Closed all receiver connections')

def processData(self): while True:

for i in range(len(self.senderconn)): print()

conn = self.senderconn[i]

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

data = str(data)

origmsg = extractMessage(data) if not origmsg:

break

if origmsg == 'q0':

break

print('Received from Sender',i+1,':',str(data))

recvno = random.randint(0,len(self.receiverconn)-1) print('Sending to Receiver',recvno+1)

rconn = self.receiverconn[recvno] cnt = extractCount(data)

msg = injectRandomError(origmsg) newdata = msg + '/' + str(cnt) + '/'

rconn[0].sendto(newdata.encode(), rconn[1])

#received from receiver

rdata = rconn[0].recv(1024).decode() rdata = str(rdata)

time.sleep(0.5) curtime = time.time()

if curtime-prevtime > 2: timeout = 1

newdata += 'TIMEOUT'

else:

timeout = 0 newdata += rdata

self.slidingwindow.append([data, newdata, i, recvno])

windowsize)+1))

'TIMEOUT':

msg = extractMessage(newdata) cnt = extractCount(newdata) status = extractStatus(newdata) print(msg,str(cnt),status)

print('Round trip time: ',str(curtime-prevtime)) print('Current frame no:',str((self.currentcount %

if (self.currentcount % windowsize)+1 == self.windowsize: idx = 0

flag = 1

while flag == 1:

idx = 0

flag = 0

while idx < self.windowsize:

currframe = self.slidingwindow[idx][1] msg = extractMessage(currframe)

cnt = extractCount(currframe) status = extractStatus(currframe)

if status == 'NAK' or status == flag = 1

break

idx += 1

print(' ')

NO:',str(idx+1))

if flag==1:

print('RESEND FROM FRAME

else:

SIZE',self.windowsize,'SUCCESSFULLY SENT')

print('BLOCK OF WINDOW

print(' ')

'''fileout = open('flag.txt', "w") fileout.write(str(flag)) fileout.close()'''

while flag == 1 and idx < self.windowsize: print()

prevtime = time.time()

prevframe = self.slidingwindow[idx][0] currframe = self.slidingwindow[idx][1] sendno = self.slidingwindow[idx][2] recvno = self.slidingwindow[idx][3] conn = self.senderconn[sendno]

rconn = self.receiverconn[recvno]

first NAK

Receiver',recvno+1)

#sending all frames to its sender from #conn[0].send(currframe.encode())

#data = conn[0].recv(1024).decode() print('Current frame no:',str(idx+1)) print('Again Sending to

rconn[1])

msg = extractMessage(prevframe) msg = injectRandomError(msg) data = msg + '/' + str(cnt) + '/' rconn[0].sendto(data.encode(),

# receiving ACK or NAK from receiver rdata = rconn[0].recv(1024).decode() rdata = str(rdata)

data += rdata

prevtime))

msg = extractMessage(data) cnt = extractCount(data) stat = extractStatus(data) curtime = time.time() print(msg,str(cnt),stat)

print('Round trip time: ',str(curtime-

self.slidingwindow[idx][1] = data idx += 1

'''fileout = open('flag.txt', "w") fileout.write(str(0)) fileout.close() '''

self.currentcount += 1 if origmsg == 'q0':

break

return

if name == ' main ':

totalsen = int(input('Enter number of senders: ')) totalrecv = int(input('Enter number of receivers: ')) windowsize =int(input('Enter window size: '))

ch = Channel(totalsen, totalrecv, windowsize) ch.initSenders()

ch.initReceivers() ch.processData() ch.closeSenders() ch.closeReceivers()

## SELECTIVE REPEAT SLIDING WINDOW PROTOCOL

**Code Snippet of sender.py:**

import socket import sys import time

def createFrame(data): countOnes = 0 for ch in data:

if ch == '1':

countOnes += 1 data += str(countOnes%2) return data

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1: endidx = i

break return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1: startidx = i+1

break return frame[startidx:]

def Main(senderno):

count = 0 sentframes = []

print('Initiating Sender #',senderno) host = '127.0.0.1'

port = 8080

mySocket = socket.socket() mySocket.connect((host, port))

while True:

print()

data = input("Enter $ ") #prevtime = time.time()

data = createFrame(data) + '/' + str(count) + '/'

msg = extractMessage(data) print('Sending to channel :',str(msg)) mySocket.send(data.encode()) sentframes.append(data)

count += 1

if not msg:

break

if msg == 'q0':

break mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of receiver.py:

import socket import sys import time import random

def waitRandomTime():

x = random.randint(0,5) if x <= 1:

time.sleep(2)

def checkError(frame): countOnes = 0 for ch in frame:

if ch == '1':

countOnes += 1 return countOnes%2

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1: endidx = i

break return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1: startidx = i+1

break return frame[startidx:]

def Main(senderno):

print('Initiating Receiver #',senderno) host = '127.0.0.2'

port = 9090

mySocket = socket.socket() mySocket.connect((host, port))

while True:

print()

data = mySocket.recv(1024).decode() data = str(data)

msg = extractMessage(data) if not msg:

break

if msg == 'q0':

break

print('Received from channel :', str(data)) waitRandomTime()

if checkError(msg) == 0: rdata = 'ACK'

else:

rdata = 'NAK'

print('Sending to channel :',str(rdata)) mySocket.send(rdata.encode())

mySocket.close()

if name == ' main ': if len(sys.argv) > 1:

senderno = int(sys.argv[1])

else:

senderno = 1

Main(senderno)

## Code Snippet of channel.py:

import socket import time import subprocess import random import os

def injectRandomError(frame):

pos = random.randint(0, len(frame)-1) frame = frame[:pos]+'1'+frame[pos+1:] return frame

def extractMessage(frame): endidx = -1

for i in range(len(frame)-1):

if frame[i] == '/' and endidx == -1: endidx = i

break return frame[:endidx]

def extractCount(frame): startidx = -1

endidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

if startidx == -1:

startidx = i+1

else:

endidx = i

cnt = frame[startidx:endidx] return int(cnt)

def extractStatus(frame): count = 0

startidx = -1

for i in range(len(frame)-1): if frame[i] == '/':

count += 1

if count == 2 and startidx == -1: startidx = i+1

break return frame[startidx:]

class Channel():

def init (self, totalsender, totalreceiver, windowsize): self.totalsender = totalsender

self.senderhost = '127.0.0.1'

self.senderport = 8080 self.senderconn = []

self.totalreceiver = totalreceiver self.receiverhost = '127.0.0.2'

self.receiverport = 9090 self.receiverconn = []

self.windowsize = windowsize self.slidingwindow = [] self.currentcount = 0 #self.statuswindow = []

def initSenders(self):

senderSocket = socket.socket() senderSocket.bind((self.senderhost, self.senderport)) senderSocket.listen(self.totalsender)

for i in range(1, self.totalsender+1): conn = senderSocket.accept() self.senderconn.append(conn)

print('Initiated all sender connections')

def closeSenders(self):

for conn in self.senderconn: conn[0].close()

print('Closed all sender connections')

def initReceivers(self): receiverSocket = socket.socket()

receiverSocket.bind((self.receiverhost, self.receiverport)) receiverSocket.listen(self.totalreceiver)

for i in range(1, self.totalreceiver+1):

conn = receiverSocket.accept() self.receiverconn.append(conn)

print('Initiated all receiver connections')

def closeReceivers(self):

for conn in self.receiverconn: conn[0].close()

print('Closed all receiver connections')

def processData(self):

'''fileout = open('flag.txt', "w") fileout.write(str(0)) fileout.close()'''

while True:

for i in range(len(self.senderconn)): print()

conn = self.senderconn[i]

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

data = str(data)

origmsg = extractMessage(data) if not origmsg:

break

if origmsg == 'q0':

break

print('Received from Sender',i+1,':',str(data))

recvno = random.randint(0,len(self.receiverconn)-1) print('Sending to Receiver',recvno+1)

rconn = self.receiverconn[recvno] cnt = extractCount(data)

msg = injectRandomError(origmsg) newdata = msg + '/' + str(cnt) + '/'

rconn[0].sendto(newdata.encode(), rconn[1])

#received from receiver

rdata = rconn[0].recv(1024).decode() rdata = str(rdata)

time.sleep(0.5) curtime = time.time()

if curtime-prevtime > 2: timeout = 1

newdata += 'TIMEOUT'

else:

timeout = 0 newdata += rdata

self.slidingwindow.append([data, newdata, i, recvno])

windowsize)+1))

msg = extractMessage(newdata) cnt = extractCount(newdata) status = extractStatus(newdata) print(msg,str(cnt),status)

print('Round trip time: ',str(curtime-prevtime)) print('Current frame no:',str((self.currentcount %

if (self.currentcount % windowsize)+1 == self.windowsize: idx = 0

flag = 1

while flag == 1:

idx = 0

flag = 0 nakframes = [] indices = []

while idx < self.windowsize:

currframe = self.slidingwindow[idx][1] msg = extractMessage(currframe)

cnt = extractCount(currframe) status = extractStatus(currframe)

'TIMEOUT':

if status == 'NAK' or status == flag = 1

nakframes.append(self.slidingwindow[idx])

indices.append(idx+1) #break

idx += 1

if flag==0:

print('

')

print('BLOCK OF WINDOW

SIZE',self.windowsize,'SUCCESSFULLY SENT')

')

print('

'''fileout = open('flag.txt', "w") fileout.write(str(flag)) fileout.close()'''

idx = 0

while flag == 1 and idx < len(nakframes): print()

prevtime = time.time()

prevframe = nakframes[idx][0] currframe = nakframes[idx][1] sendno = nakframes[idx][2] recvno = nakframes[idx][3] conn = self.senderconn[sendno]

rconn = self.receiverconn[recvno] stat = extractStatus(currframe)

')

NO:',str(indices[idx])) ')

first NAK

no:',str(indices[idx])) Receiver',recvno+1)

print(' print('RESENDING FRAME

print(' #sending all frames to its sender from #conn[0].send(currframe.encode())

#data = conn[0].recv(1024).decode() print('Current frame

print('Again Sending to

msg = extractMessage(prevframe) msg = injectRandomError(msg) data = msg + '/' + str(cnt) + '/'

rconn[0].sendto(data.encode(),

rconn[1])

# receiving ACK or NAK from receiver rdata = rconn[0].recv(1024).decode() rdata = str(rdata)

data += rdata

prevtime)) data

msg = extractMessage(data) cnt = extractCount(data) stat = extractStatus(data) curtime = time.time() print(msg,str(cnt),stat)

print('Round trip time: ',str(curtime- self.slidingwindow[indices[idx]-1][1] = idx += 1

self.currentcount += 1 if origmsg == 'q0':

break

return

if name == ' main ':

totalsen = int(input('Enter number of senders: ')) totalrecv = int(input('Enter number of receivers: ')) windowsize =int(input('Enter window size: '))

ch = Channel(totalsen, totalrecv, windowsize) ch.initSenders()

ch.initReceivers() ch.processData() ch.closeSenders() ch.closeReceivers()

# TEST CASES

## Stop and Wait ARQ:

**channel.py :**

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\stopnwait>python channel.py Enter number of senders: 2**

**Enter number of receivers: 2 Initiated all sender connections Initiated all receiver connections**

**Received from Sender 1 : 10010 Sending to Receiver 2**

**Received from Receiver 2 : TIMEOUT Sending to Sender 1**

**1**

**Again Received from Sender 1 : 10010 Again Sending to Receiver 2**

**Again Received from Receiver 2 : TIMEOUT Again Sending to Sender 1**

**1**

**Again Received from Sender 1 : 10010 Again Sending to Receiver 2**

**Again Received from Receiver 2 : ACK Again Sending to Sender 1**

**0**

**Received from Sender 2 : 00101 Sending to Receiver 2**

**Received from Receiver 2 : ACK Sending to Sender 2**

**0**

## Sender.py (1st sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\stopnwait>python sender.py 1 Initiating Sender # 1**

**Enter $ 1001**

**Sending to channel : 10010 Received from channel : TIMEOUT**

**Round trip time: 2.031153678894043**

**TIMEOUT of 2s EXPIRED !!**

**Again Sending to channel : 10010**

**Again Received from channel : TIMEOUT Round trip time: 4.031170129776001 seconds**

**TIMEOUT of 2s EXPIRED !!**

**Again Sending to channel : 10010 Again Received from channel : ACK**

**Round trip time: 0.015580177307128906 seconds**

## Sender.py (2nd sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\stopnwait>python sender.py 2**

**Initiating Sender # 2**

**Enter $ 0010**

**Sending to channel : 00101 Received from channel : ACK Round trip time: 0.0**

## Receiver.py (1st receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\stopnwait>python receiver.py 1 Initiating Receiver # 1**

## Receiver.py (2nd receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\stopnwait>python receiver.py 2 Initiating Receiver # 2**

**Received from channel : 10011 Sending to channel : TIMEOUT**

**Received from channel : 10110 Sending to channel : TIMEOUT**

**Received from channel : 10010 Sending to channel : ACK**

**Received from channel : 00101 Sending to channel : ACK**

## Go Back N ARQ:

**Channel.py:**

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\gobackn>python channel.py Enter number of senders: 2**

**Enter number of receivers: 2 Enter window size: 3**

**Initiated all sender connections Initiated all receiver connections**

**Received from Sender 1 : 10010/0/ Sending to Receiver 2**

**11010 0 NAK**

**Round trip time: 0.5156002044677734 Current frame no: 1**

**Received from Sender 2 : 01001/0/ Sending to Receiver 2**

**01101 0 NAK**

**Round trip time: 0.5156097412109375 Current frame no: 2**

**Received from Sender 1 : 10100/1/ Sending to Receiver 2**

**10110 1 NAK**

**Round trip time: 0.5000448226928711 Current frame no: 3**

**------------------------------ RESEND FROM FRAME NO: 1**

**------------------------------**

**Current frame no: 1**

**Again Sending to Receiver 2 10011 0 NAK**

**Round trip time: 2.0155763626098633**

**Current frame no: 2**

**Again Sending to Receiver 2 01011 0 NAK**

**Round trip time: 0.0**

**Current frame no: 3**

**Again Sending to Receiver 2 10100 0 ACK**

**Round trip time: 0.0**

**------------------------------ RESEND FROM FRAME NO: 1**

**------------------------------**

**Current frame no: 1**

**Again Sending to Receiver 2 10011 0 NAK**

**Round trip time: 0.0**

**Current frame no: 2**

**Again Sending to Receiver 2 01101 0 NAK**

**Round trip time: 2.000006675720215**

**Current frame no: 3**

**Again Sending to Receiver 2 11100 0 NAK**

**Round trip time: 0.0**

**------------------------------ RESEND FROM FRAME NO: 1**

**------------------------------**

**Current frame no: 1**

**Again Sending to Receiver 2 10011 0 NAK**

**Round trip time: 0.0**

**Current frame no: 2**

**Again Sending to Receiver 2 01001 0 ACK**

**Round trip time: 2.0155766010284424**

**Current frame no: 3**

**Again Sending to Receiver 2 10100 0 ACK**

**Round trip time: 0.0**

**------------------------------ RESEND FROM FRAME NO: 1**

**------------------------------**

**Current frame no: 1**

**Again Sending to Receiver 2 10011 0 NAK**

**Round trip time: 0.0**

**Current frame no: 2**

**Again Sending to Receiver 2 01101 0 NAK**

**Round trip time: 2.0155653953552246**

**Current frame no: 3**

**Again Sending to Receiver 2 10100 0 ACK**

**Round trip time: 0.0**

**------------------------------ RESEND FROM FRAME NO: 1**

**------------------------------**

**Current frame no: 1**

**Again Sending to Receiver 2 10010 0 ACK**

**Round trip time: 0.0**

**Current frame no: 2**

**Again Sending to Receiver 2 01101 0 NAK**

**Round trip time: 2.015582799911499**

**Current frame no: 3**

**Again Sending to Receiver 2 10110 0 NAK**

**Round trip time: 0.0**

**------------------------------ RESEND FROM FRAME NO: 2**

**------------------------------**

**Current frame no: 2**

**Again Sending to Receiver 2 01001 0 ACK**

**Round trip time: 0.0**

**Current frame no: 3**

**Again Sending to Receiver 2 10101 0 NAK**

**Round trip time: 2.0155813694000244**

**------------------------------ RESEND FROM FRAME NO: 3**

**------------------------------**

**Current frame no: 3**

**Again Sending to Receiver 2 11100 0 NAK**

**Round trip time: 2.002156972885132**

**------------------------------ RESEND FROM FRAME NO: 3**

**------------------------------**

**Current frame no: 3**

**Again Sending to Receiver 2 10100 0 ACK**

**Round trip time: 2.0155797004699707**

**------------------------------**

**BLOCK OF WINDOW SIZE 3 SUCCESSFULLY SENT**

**------------------------------**

## Sender.py (1st sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\gobackn>python sender.py 1 Initiating Sender # 1**

**Enter $ 1001**

**Sending to channel : 10010**

**Enter $ 1010**

**Sending to channel : 10100 Enter $**

## Sender.py (2nd sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\gobackn>python sender.py 2 Initiating Sender # 2**

**Enter $ 0100**

**Sending to channel : 01001 Enter $**

## Receiver.py (1st receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\gobackn>python receiver.py 1 Initiating Receiver # 1**

## Receiver.py (2nd receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\gobackn>python receiver.py 2 Initiating Receiver # 2**

**Received from channel : 11010/0/ Sending to channel : NAK**

**Received from channel : 01101/0/ Sending to channel : NAK**

**Received from channel : 10110/1/ Sending to channel : NAK**

**Received from channel : 10011/0/ Sending to channel : NAK**

**Received from channel : 01011/0/ Sending to channel : NAK**

**Received from channel : 10100/0/ Sending to channel : ACK**

**Received from channel : 10011/0/ Sending to channel : NAK**

**Received from channel : 01101/0/ Sending to channel : NAK**

**Received from channel : 11100/0/**

**Sending to channel : NAK**

**Received from channel : 10011/0/ Sending to channel : NAK**

**Received from channel : 01001/0/ Sending to channel : ACK**

**Received from channel : 10100/0/ Sending to channel : ACK**

**Received from channel : 10011/0/ Sending to channel : NAK**

**Received from channel : 01101/0/ Sending to channel : NAK**

**Received from channel : 10100/0/ Sending to channel : ACK**

**Received from channel : 10010/0/ Sending to channel : ACK**

**Received from channel : 01101/0/ Sending to channel : NAK**

**Received from channel : 10110/0/ Sending to channel : NAK**

**Received from channel : 01001/0/ Sending to channel : ACK**

**Received from channel : 10101/0/ Sending to channel : NAK**

**Received from channel : 11100/0/ Sending to channel : NAK**

**Received from channel : 10100/0/ Sending to channel : ACK**

## Selective Repeat ARQ:

**Channel.py:**

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\selectiverepeat>python channel.py Enter number of senders: 2**

**Enter number of receivers: 2 Enter window size: 3**

**Initiated all sender connections Initiated all receiver connections**

**Received from Sender 1 : 10010/0/ Sending to Receiver 1**

**10010 0 ACK**

**Round trip time: 0.5156097412109375 Current frame no: 1**

**Received from Sender 2 : 01001/0/ Sending to Receiver 2**

**01001 0 ACK**

**Round trip time: 0.5156137943267822 Current frame no: 2**

**Received from Sender 1 : 10100/1/ Sending to Receiver 1**

**10101 1 NAK**

**Round trip time: 0.5155987739562988 Current frame no: 3**

**------------------------------ RESENDING FRAME NO: 3**

**------------------------------**

**Current frame no: 3**

**Again Sending to Receiver 1 10110 1 NAK**

**Round trip time: 0.0**

**------------------------------ RESENDING FRAME NO: 3**

**------------------------------**

**Current frame no: 3**

**Again Sending to Receiver 1 10100 1 ACK**

**Round trip time: 2.015587329864502**

**------------------------------**

**BLOCK OF WINDOW SIZE 3 SUCCESSFULLY SENT**

**------------------------------**

## Sender.py (1st sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\selectiverepeat>python sender.py 1 Initiating Sender # 1**

**Enter $ 1001**

**Sending to channel : 10010**

**Enter $ 1010**

**Sending to channel : 10100 Enter $**

## Sender.py (2nd sender):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\selectiverepeat>python sender.py 2 Initiating Sender # 2**

**Enter $ 0100**

**Sending to channel : 01001 Enter $**

## Receiver.py (1st receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\selectiverepeat>python receiver.py 1 Initiating Receiver # 1**

**Received from channel : 10010/0/ Sending to channel : ACK**

**Received from channel : 10101/1/ Sending to channel : NAK**

**Received from channel : 10110/1/ Sending to channel : NAK**

**Received from channel : 10100/1/ Sending to channel : ACK**

## Receiver.py (2nd receiver):

**C:\Users\SOURAV\Desktop\comp-networks-lab\ass2\selectiverepeat>python receiver.py 2 Initiating Receiver # 2**

**Received from channel : 01001/0/ Sending to channel : ACK**

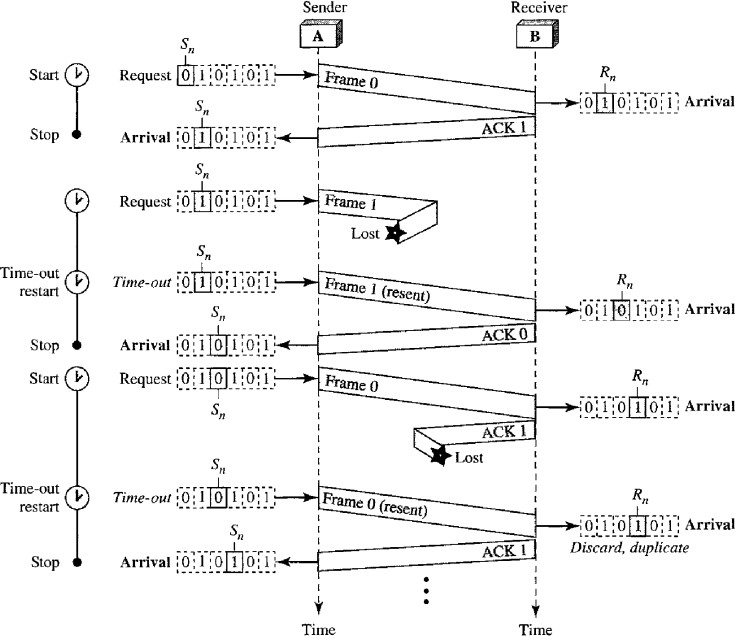
# RESULTS AND ANALYSIS

## Stop and Wait ARQ:

Stop and Wait ARQ adds a simple error control mechanism to the Stop and Wait protocol for noiselsess channel. To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver site, it is checked if it is corrupted.

The corrupted and lost frames need to be resent in this protocol. If the receiver does not respond when there is an error, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted.

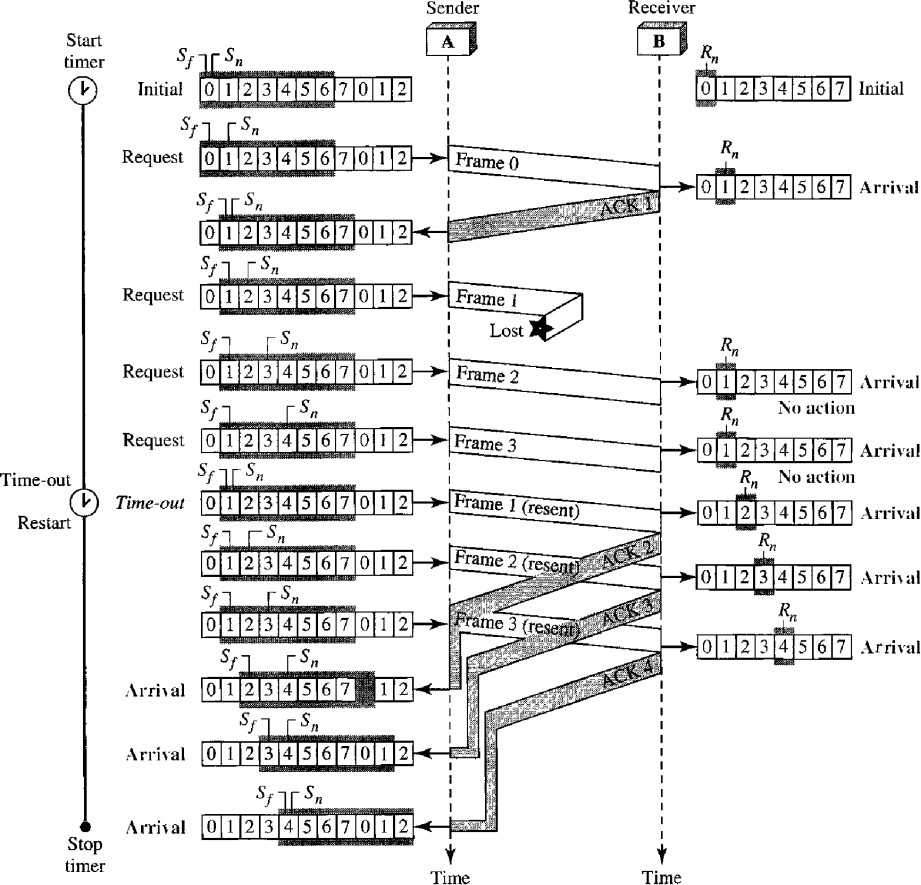
The Stop and Wait ARQ is very inefficient if our channel is thick and long. By thick, we mean that our channel has a large bandwidth; by long, we mean the round-trip delay is long. The product of the two is called the bandwidth-delay product. It is the volume of the pipe in bits.



## Go Back N ARQ:

To improve the efficiency of transmission (filling the pipe), multiple frames are in transition while waiting for acknowledgment. In this protocol, we can send several frames before receiving acknowledgments; we keep a copy of these frames until acknowledgments arrive.

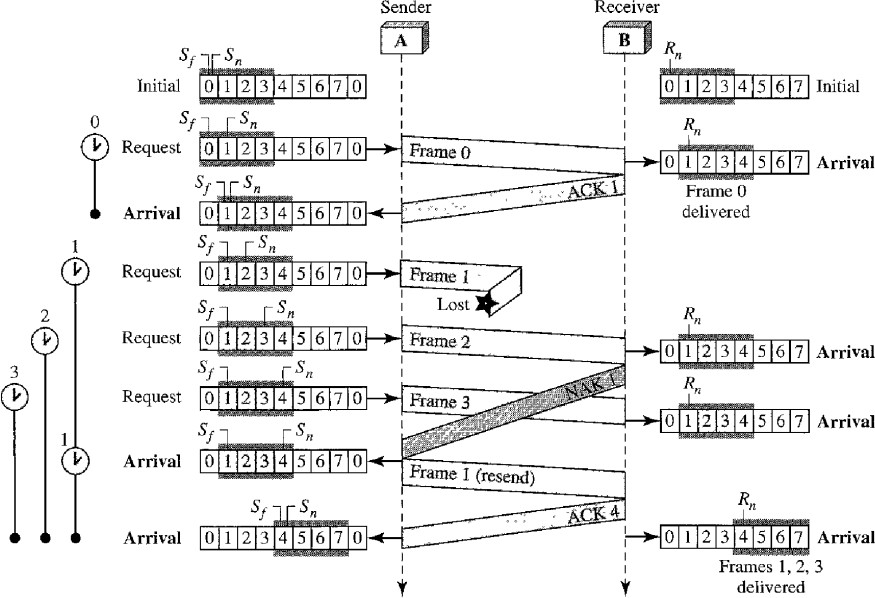
In this protocol, when the timer expires, the sender resends all the outstanding frames. For example, suppose the sender has already sent frame 6, bur the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again.



## Selective Repeat ARQ:

Go Back N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, it is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission.

For noisy links, there is another mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex.



# COMMENTS

This assignment has helped me in understanding the different data link layer protocols immensely, by researching and implementing them. It has also helped in understanding the demerits of a protocol, and how such demerits are overcome by other protocols.