COSC264

Data Communications and Networking Assignment

Name: Nobu Sato

Student number: 56572069

Contribution: Responsible for implementing the Channel and Receiver. 50%

Name: Josh Burt

Student number: 92180584

Contribution: Responsible for implementing the Packet and Sender. 50%

1. The protocol between **sender** and **receiver** as described above has (at least) one weakness: it has a **deadlock**. Please explain the notion of a deadlock in the context of networking protocols and describe the particular deadlock situation in our case. A guiding question is: what can go wrong and when in case certain packets are lost?

A deadlock in networking terms is when two differant sockets are both waiting to use resources held by the other, they also wont free their current resources until they have the new resources. This can occur if two non-blocking calls were trying to run at the same time, say the sender and reciever were trying to send simultaneously.

2. What is the **magicno** field good for?

The magicno field is implemented in many operating systems. The magicno field implement strongly typed data and controller programs to read the data types at program run-time. Many packets have such a constants that identify the contained data. For an example of this program, a magino field is used to check whether the packet received in the Channel, and the Receiver received the correct packet.

3. How have you solved the issue with the bit errors? Please explain what you have added to the packet and to the **sender** and **receiver** modules.

Each packet contains a 32 bit checksum trailer. This is calculated at the sending end and is then checked at the recieving end. The checksum is not changed through the sending process so if these two values do not match it is assumed that a bit error has occured and the packet is dropped.

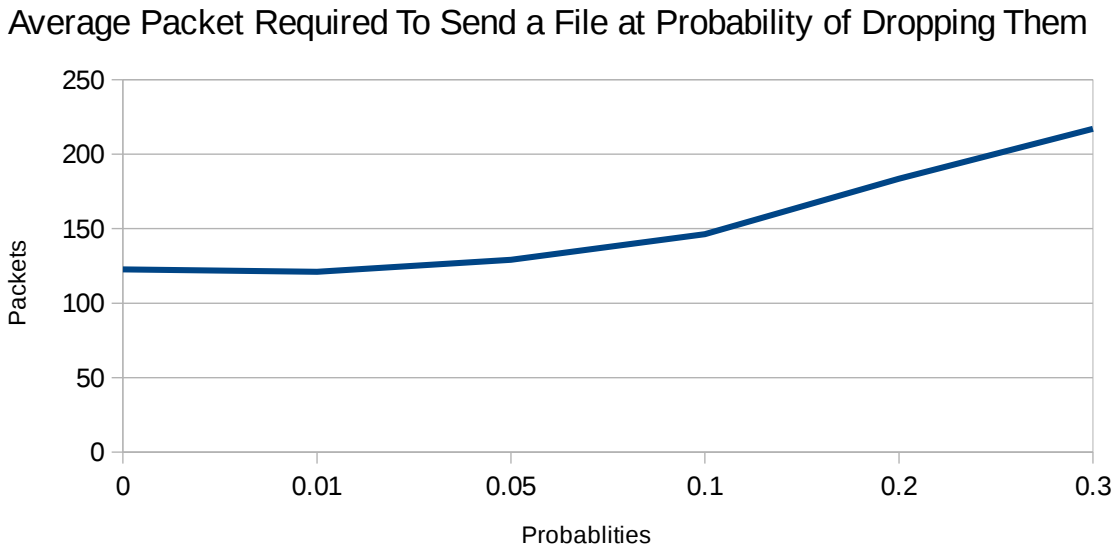
4. Please explain what the select() function is doing and why it is useful for the channel (and in another way for the sender).

The select function is a straightforward interface to the Unix system call. The function takes in three separate arguments where the arguments wait for reading, writing and an exceptional condition respectively. The Channel uses the select function to read each packet received by either the Sender or the Receiver. The channel checks if the packet has the correct magicno header number and if so the packet is sent to the Sender or Receiver depending on the packet's origin.

5. Please explain how you have checked whether or not the file was transferred correctly (i.e. the receivers copy is identical to the transmitters copy).

The checksum is calculated on the senders end upon transferring a chunk of file data. This checksum is then computed again and rechecked when the packet is recieved. If the checksum is differant the file is assumed to be corrupt and the packet is dropped and retransmission is asked for.

6. We consider different packet loss probabilities of *P 2 f*0:0; 0:01; 0:05; 0:1; 0:2; 0:3*g* and a source file of length *M* = 512 ∗ 100 = 51; 200 bytes (you need to create such a file). For each value of *P* make ten repetitions of the file transfer and for each repetition record how many packets the sender has sent in total. Draw a graph that shows the different values of *P* on the x-axis and for each such value the *average* number of total packets (the average being taken over the ten repetitions) on the y-axis. Explain the results.
Note: To produce graphs, the tool gnuplot can be useful under Linux. Its main advantage is that it allows for script-based (i.e. non-interactive) creation of graphs, but admittedly its command syntax needs some getting used to. However, you are free to use any tool you like (including Excel, Matlab, etc.) for producing graphs. Under all circumstances you need to make sure that axes and curves are properly labeled. You will lose marks otherwise.



7. Assume the following:
● The probability to loose an individual packet (either a dataPacket or an acknowledgementPacket) is *P*,

● Packet loss events are statistically independent of each other.

● The size of the file to be transmitted requires *N* packets.

Please derive and justify an expression for the average total number of packets that need to be sent (including retransmissions) to transmit the entire file. Compare this to the (average) total number of packets you have observed in your experiments

Using a binomial distrbution for approxiamting this we can use the formula for n trials given, k successes (Cook, John D)

$$\binom{n-1}{k-1} p^k (1-p)^{n-k}$$

Where k = N and n is the nuber of trials required to acheieve that many successes. With some manipulation this gives the mean value of this to be

$$Average = \frac{N}{1-P}$$

The values from this functionare lower than given by this function. However the function above does not take into account a resent packet getting resent again.

Referances:

https://www.johndcook.com//negative_binomial.pdf

Packet.py

```
import hashlib

class Packet:


    checksum = None


    def __init__(self,magicno,packetType, seqno, dataLen, data = None, checksum=None):
        try:
            self.magicno = hex(magicno)
        except:
            self.magicno = magicno
        if packetType == "dataPacket" or packetType == "0" or packetType == 0:
```

```
            self.packetType = 0 # using 0 for data packets and 1 for ack packets
        elif packetType == "acknowledgementPacket" or packetType == "1" or packetType == 1:
            self.packetType = 1
        else:
            raise TypeError
        self.seqno = seqno
        self.dataLen = int(dataLen)
        if data is None:
            self.data = b""
        else:
            self.data = data
        if checksum is not None:
            self.checksum = checksum


    def getChecksum(self):


        encoded_string = b""+str(self.magicno)+str(self.packetType)+str("{}".format(self.seqno))
+str("{0:03}".format(self.dataLen))+self.data
        return hashlib.md5(encoded_string.encode('utf-8')).hexdigest()


    def encode(self, recalculateChecksum=True):
        """
        Encodes the packet data into a bit string for sending
        """


        magicno = str(int(str(self.magicno),0)) #Converting hex decimal to a integer. Then
convert it into a string.
        encoded_string = b""+str(self.magicno)+str(self.packetType)+str(self.seqno)
+str("{0:03}".format(self.dataLen))+self.data #"," used to separate the data length and the
data.
        if recalculateChecksum:
```

```
            checksum = hashlib.md5(encoded_string.encode('utf-8')).hexdigest()
        else:
            checksum = self.checksum
        encoded_string = encoded_string + checksum
        return encoded_string


    @staticmethod
    def decode(encoded_string):
        """
        Takes an encoded string and decodes it back to the original.
        Checks checksum and ensures packet is recieved as expected
        """
        checksum = encoded_string[-32:] #selects the last 32 chars of the string which is the hash
        dataLength = encoded_string[8:11]
        if encoded_string[6] == '0':
            packetType = "dataPacket"
        elif encoded_string[6] == '1':
            packetType = "acknowledgementPacket"
        else:
            packetType = "dataPacket"
        if dataLength == 0:
            data = None
        else:
            data = encoded_string[12:-32]
        return Packet(encoded_string[:6], packetType, encoded_string[7], dataLength, data, checksum)#magicno,packetType,seqno,datalen,data,checksum



    def __str__(self):
```

```
        return("MagicNo: {}\nType: {}\nSequence Number: {}\nData Length:
{}".format(self.magicno,self.packetType,self.seqno,self.dataLen))
```

Sender.py

```
import sys, socket
from Packet import Packet


class Sender:
    localhost = "127.0.0.1"
    bufferSize = 1024


    def __init__(self):
        """
        Initializes the sender class and reads 4 command line arguments.
        Input port, outputPort, channel input port and file to send
        """
        self.sIn = int(sys.argv[1])
        self.sOut = int(sys.argv[2])
        self.chanSocket = int(sys.argv[3])
        self.sendFile = str(sys.argv[4])
        if (self.checkRange(self.sIn) and self.checkRange(self.sOut)) != True:
            sys.exit("Either a Port is out of range or the specified file does not exist. Please try
again.")
        else:
            print("Preparing sockets:\nInput: {}\nOutput: {}\nChannel: {}\nTo send file:
{}").format(self.sIn, self.sOut, self.chanSocket, self.sendFile)
        self.socketIn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    self.socketOut = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketIn.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketOut.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketIn.bind((self.localhost, self.sIn))
    self.socketOut.bind((self.localhost, self.sOut))
    self.socketOut.connect((self.localhost, self.chanSocket))
    self.packetsSent = 0
    self.next = 0
    self.exitFlag = False



def run(self):
    """
    Once the instance of a Sender is created run can be called to send the file
    Returns void
    """

    raw_packet = None
    readAmount = 512 #max number of bytes to be read into a file
    try:
        infile = open(self.sendFile, "rb")
    except IOError:
        infile.close()
        self.closePorts()
        sys.exit("File Not Found")
    while self.exitFlag == False:
        data = infile.read(readAmount)
        if not data:
            data = ""
```

```
            self.exitFlag = True
        sendPacket = Packet(0x497E,"dataPacket", self.next, len(data), data)
        self.packetsSent += 1
        self.socketOut.send(sendPacket.encode())
        print("packet sent")
        acknowledged = False
        while not acknowledged:
            self.socketIn.settimeout(1.0)
            try:
                self.socketIn.listen(1)
                if raw_packet == None:
                    raw_packet, addr = self.socketIn.accept()
                    raw_packet.settimeout(1)
                packet = raw_packet.recv(self.bufferSize)
                if packet == "": #No incoming packets from the Channel.
                    break
                print("packet Received")
                ackPacket = Packet.decode(packet)
                if ackPacket.packetType != "0x497E" and ackPacket.packetType != 1 and
ackPacket.seqno != self.next and ackPacket.checksum != ackPacket.getChecksum():
                    sendPacket = Packet(0x497E,"dataPacket", self.next, len(data), data)
                    self.socketOut.send(sendPacket.encode())
                    print("packet resent- corrupted")
                    self.packetsSent += 1
                    continue


                else:
                    acknowledged = True
                    self.next = 1 - self.next
            except socket.timeout:
```

```
            print("packet resent - timed out")
            sendPacket = Packet(0x497E,"dataPacket", self.next, len(data), data)
            self.socketOut.send(sendPacket.encode())
            self.packetsSent += 1
    infile.close()
    print("Total packets sent including resends: {}".format(self.packetsSent))


def checkRange(self, portnumber):
    """
    takes a Port number provided and checks to see if it is in the valid
    range.
    Returns a true if it is
    """
    if((portnumber > 1024) and (portnumber < 64000)):
        return True
    else:
        return False



def closePorts(self):
    """
    Closes all open ports that have been created
    Returns void
    """
    self.socketIn.close()
    self.socketOut.close()
    sys.exit("All the open ports in Sender is now closed")
```

```
sender = Sender()
sender.run()
sender.closePorts()
```

Receiver.py

```
import sys, socket, select
from Packet import Packet

class Receiver:
    localHost = "127.0.0.1"
    def __init__(self):
        self.rin = int(sys.argv[1])
        self.rout = int(sys.argv[2])
        self.crin = int(sys.argv[3])
        self.fileName = sys.argv[4]
        self.portNumbers = [self.rin, self.rout]
        self.validNumber()

        self.expected = 0 #Local integer
        self.bufferSize = 1024
```

```
    self.magicno = '0x497e'

    self.files = []

    localHost = socket.gethostbyname(socket.gethostname())
    self.socketrin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketrout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketrin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketrout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketrin.bind((self.localHost, self.rin))
    self.socketrout.bind((self.localHost, self.rout))

    self.socketrout.settimeout(10)
    '''
    try:
        self.socketrout.connect((localHost, self.crin))
    except:
        sys.exit("Socket error (Transport endpoint is not connected)")
    '''

def validNumber(self):
    """Checks if the port numbers are within the range 1024 and 64000.
       Also checks if the number is integer and all the port numbers are unique.
    """
    nums = []
    for pnum in self.portNumbers:
        if pnum > 64000 or pnum < 1024:
            sys.exit("Port number must be between 1024 and 64000")
        if pnum in nums: #Checks if the port numbers are unique.
```

```
            sys.exit("Port number must be unique!")
        nums.append(pnum)
    #Change the type to integer since the port number should be a integer.


def run(self):
    """Once the instance of a Receiever is created. "run" can be called to
      to receive and store the file
      """
    raw_packet = None
    try:
        outfile = open(self.fileName, "w") #Receiver opens a file with supplied filename for writing
    except IOError:
        sys.exit("File Not Found")
    for file in self.files: #Checks if the file already exists.
        if file == self.fileName:
            sys.exit("File already exists")
    self.files.append(self.fileName)
    expected = 0
    while True:
        self.socketrin.listen(5)
        if raw_packet == None:
            raw_packet, addr = self.socketrin.accept()
            self.socketrout.connect((self.localHost, self.crin))
        packet = raw_packet.recv(self.bufferSize)
        if packet == "": #No more incoming packets from Channel.
            return
        print("Packet received")
        decodedPacket = Packet.decode(packet)
```

```
        if decodedPacket.seqno != expected and decodedPacket.packetType != 0 and
decodedPacket.checksum != decodedPacket.getChecksum() and decodedPacket.magicno !=
self.magicno:
            newPacket = Packet(0x497E, "acknowledgementPacket", int(decodedPacket.seqno),
0, decodedPacket.data)
            self.socketrout.send(newPacket.encode())
            print("Sent Packet, Packet corrupted")
        else:
            newPacket = Packet(0x497E, "acknowledgementPacket", int(decodedPacket.seqno),
0) #Datafield is empty.
            self.socketrout.send(newPacket.encode())
            print("Sent Packet Packet all good")
            expected = 1 - expected
            if decodedPacket.dataLen != 0:
                outfile.write(decodedPacket.data)
            else:
                outfile.close()
                self.closePorts()




    def closePorts(self):
        """
        Closes all open ports that have been created
        Returns void
        """
        self.socketrin.close()
        self.socketrout.close()
        sys.exit("All the open ports in Receiver is now closed")
```

```
receiver = Receiver()
receiver.run()
receiver.closePorts()
```

Channel.py

```
import sys, socket, select, random
from Packet import Packet
class Channel:
    localHost = "127.0.0.1"
    def __init__(self):
        self.csin = int(sys.argv[1])
```

```
    self.csout = int(sys.argv[2])
    self.crin = int(sys.argv[3])
    self.crout = int(sys.argv[4])
    self.sin = int(sys.argv[5])
    self.rin = int(sys.argv[6])
    self.valueP = float(sys.argv[7])
    self.portNumbers = [self.csin, self.csout, self.crin, self.crout]
    self.magicno = "0x497e"
    self.maxBuffSize = 1024 #The maximum size of the buffer
    self.validNumber()
    self.pRange()
    self.done = False
    self.halfDone = False

    self.socketcsin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketcsout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketcrin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketcrout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socketcsin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketcsout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketcrin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketcrout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    self.socketcsin.bind((self.localHost, self.csin))
    self.socketcsout.bind((self.localHost, self.csout))
    self.socketcrin.bind((self.localHost, self.crin))
    self.socketcrout.bind((self.localHost, self.crout))

def pRange(self):
    """Checks if the floating point/ souble precision is within the range
```

```
        0 <= P < 1
    """
    if self.valueP >= 1 or self.valueP < 0:
        print("P value should be in the range 0 or more and less than 1")


def validNumber(self):
    """Checks if the port numbers are within the range 1024 and 64000.
       Also checks if the number is integer and all the port numbers are unique.
    """
    nums = []
    for pnum in self.portNumbers:
        if pnum > 64000 or pnum < 1024:
            sys.exit("Port number must be between 1024 and 64000")
        if pnum in nums: #Checks if the port numbers are unique.
            sys.exit("Port number must be unique!")
        nums.append(pnum)


def run(self):
    """Program go through infinite loop and wait for input on any one of
       the of the two sockets (csout and crout) and set their default
       receiver to the port numbers used by sender and receiver for the
       sockets sin and rin.
    """


    rlist = [self.socketcsin,self.socketcrin]
    sendConn = None
    recvConn = None
    while True:
        read, write, error = select.select(rlist, [], [])
```

```
for socket in read:
    if socket == self.socketcsin:
        socket.listen(10)
        sendConn, addr = socket.accept()
        print("Packet received from sender")
        rlist.append(sendConn)
        packet = sendConn.recv(self.maxBuffSize)
        self.socketcrout.connect((self.localHost, self.rin))
        print("Packet processed and ready to send")
        self.socketcrout.send(packet)
        print("Packet Sent to receiver")
    elif socket == self.socketcrin:
        socket.listen(10)
        recvConn, addr = socket.accept()
        print("Packet received from receiver")
        rlist.append(recvConn)
        packet = recvConn.recv(self.maxBuffSize)
        self.socketcsout.connect((self.localHost, self.sin))
        print("Packet processed and ready to send")
        self.socketcsout.send(packet)
        print("Packet Sent to sender")
    elif socket == sendConn:
        packet = sendConn.recv(self.maxBuffSize)
        if packet == "": #No incoming packets
            return
        decodedPacket = Packet.decode(packet)
        print("Packet recieved from sender sc")
        packet = self.introduceBitError(packet)
        if packet is not None:
```

```
                self.socketcrout.send(packet)
                print("Packet Sent to receiver sc")

            elif socket == recvConn:
                packet = recvConn.recv(self.maxBuffSize)
                if packet == "": #No incoming packets
                    return
                decodedPacket = Packet.decode(packet)
                print("Packet received from receiver rc")
                packet = self.introduceBitError(packet)
                if packet is not None:
                    self.socketcsout.send(packet)
                    print("Packet Sent to sender rc")



    def introduceBitError(self, packet):
        """
        Needs to be implemented here so the send structure works. SHould process the packet
        and then return the packet to go.
        """
        randomU = random.uniform(0,1) #Random variate u
        randomV = random.uniform(0,1) #Random variate v


        decodedPacket = Packet.decode(packet)
        if decodedPacket.magicno != self.magicno: #If the packet header does not equal '0x497E' then it should return back to the start of the loop.
            print("The magicno header " + str(decodedPacket.magicno) + " does not match '0x497E'!")
            return None
```

```
        if randomU < self.valueP: #Drops packet if u (uniformly distributed random variate) is smaller than P rate
            return None #Packets has been lost
        else: #if the packet is not dropped
            if randomV < 0.1:
                randomInt = int(random.uniform(1,10)) #converts floating number to integer.
                decodedPacket.dataLen += randomInt #Increments dataLen field of the packet by a random integer between 1-10
                return decodedPacket.encode(False)
            else:
                return packet



    def closePorts(self):
        """
        Closes all open ports that have been created
        Returns void
        """
        self.socketcsin.close()
        self.socketcsout.close()
        self.socketcrin.close()
        self.socketcrout.close()
        sys.exit("All the open ports in Channel is now closed.")




channel = Channel()
channel.run()
channel.closePorts()
```