Computer Networks

Name: Bikash Sah

Roll No: 002010501018

Class: BCSE-3

Semester: 5th

Section: A1

Institute: Jadavpur University

Index

Computer Networks Lab Report - Assignment 1

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 1

Problem Statement: Design and implement an error detection module.

Design and implement an error detection module which has four schemes namely LRC, VRC, Checksum and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0,1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, codeword will be prepared. Sender will send the codeword to the Receiver. Receiver will extract the dataword from codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases (not limited to).

(a) Error is detected by all four schemes. Use a suitable CRC polynomial (list is given in next page).

(b) Error is detected by checksum but not by CRC.

(c) Error is detected by VRC but not by CRC.

[Note: Inject error in random positions in the input data frame. Write a separate method for that.]

Submission Date: **7 August 2022**

Deadline: **7 August 2022**

Introduction

First question that comes in our mind is that, why even do we need an error detection module? Answer to that is, when we transmit/receive data it is in bits. These bits travel through a medium, and these stream of bits are subjected to electromagnetic (optical) interference due to noise signals (light sources). Thus, the data transmitted may be prone to errors.

So, we need some technique that would help us determine errors in the signal(in the receiver side) and ask the transmitter to retransmit it.

In this project, we will mainly focus on 4 error detection techniques.

1. **Vertical Redundancy Check(VRC)**
    - It is also knows as **Parity Check.**
    - In this method, a redundant bit also called parity bit is added to each data unit.
    - This method includes even parity and odd parity.
    - Even parity means the total number of 1s in data(Including redundant bits) is to be even and odd parity means the total number of 1s in data(Including redundant bits) is to be odd.
    - We will use **Even Parity** in this project.

    First the data bits are split into 4 bit groups.

    For example, for the message `1 0 0 1 0 0 0 1 1 1 1 1` with even parity bit scheme, the message to be transmitted is `1 0 0 1 0` `0 0 0 1 1` `1 1 1 1 0`
    The 5th bit at the end of the nibble represents the even parity bit corresponding to that nibble.

    Advantages:

    - *VRC can detect all single bit error.*
    - It can also detect burst errors but only in those cases where number of bits changed is odd, i.e. 1, 3, 5, 7, …….etc.

    Disadvantages:

- It is not able to detect burst error if the number of bits changed is even, i.e. 2, 4, 6, 8, …….etc.

2. **Longitudinal Redundancy Check(LRC)**

   - It is also know as 2 Dimensional Parity Check.
   - Organize data into a table and create a parity for each column.

   Example:

   Suppose the message to be transmitted is `1 0 1 1  1 0 0 0  1 0 0 1`
   Then, we compute the even parity nibble as follows:

   `1 0 1 1`

   `1 0 0 0`

   `1 0 0 1`

   `1 0 1 0`

   We note that in this scheme, the number of 1's in each column including the bit in the parity nibble must be even.

   Data is send along with parity bits : `1 0 1 1  1 0 0 0  1 0 0 1` `1 0 1 0`

   Advantages:

   - LRC is used to detect burst errors.

   Disadvantage:

   - The main problem with LRC is that, it is not able to detect error if two bits in a data unit are damaged and two bits in exactly the same position in other data unit are also damaged.

3. **Check Sum**

   - In check sum method, we divide the stream of bits in 'k' blocks, each block consisting 'n' bits.
   - We add all k blocks, if carry is generated it is again added to the sum.
   - We do 1's complement of the sum and we get the required checkSum value.
   - In receiver's side all the values of codewords are added and 1's complement is done of the sum, if sum==0 then No errors else error.

4. **Cyclic Redundancy Check**

   - CRC performs mod 2 arithmetic (exclusive-OR) on the message using a divisor polynomial. Firstly, the message to be transmitted is appended with

CRC bits and the number of such bits is the degree of the divisor polynomial.

- The divisor polynomial 1 1 0 1 corresponds to the x^3 + x^2 + 1.

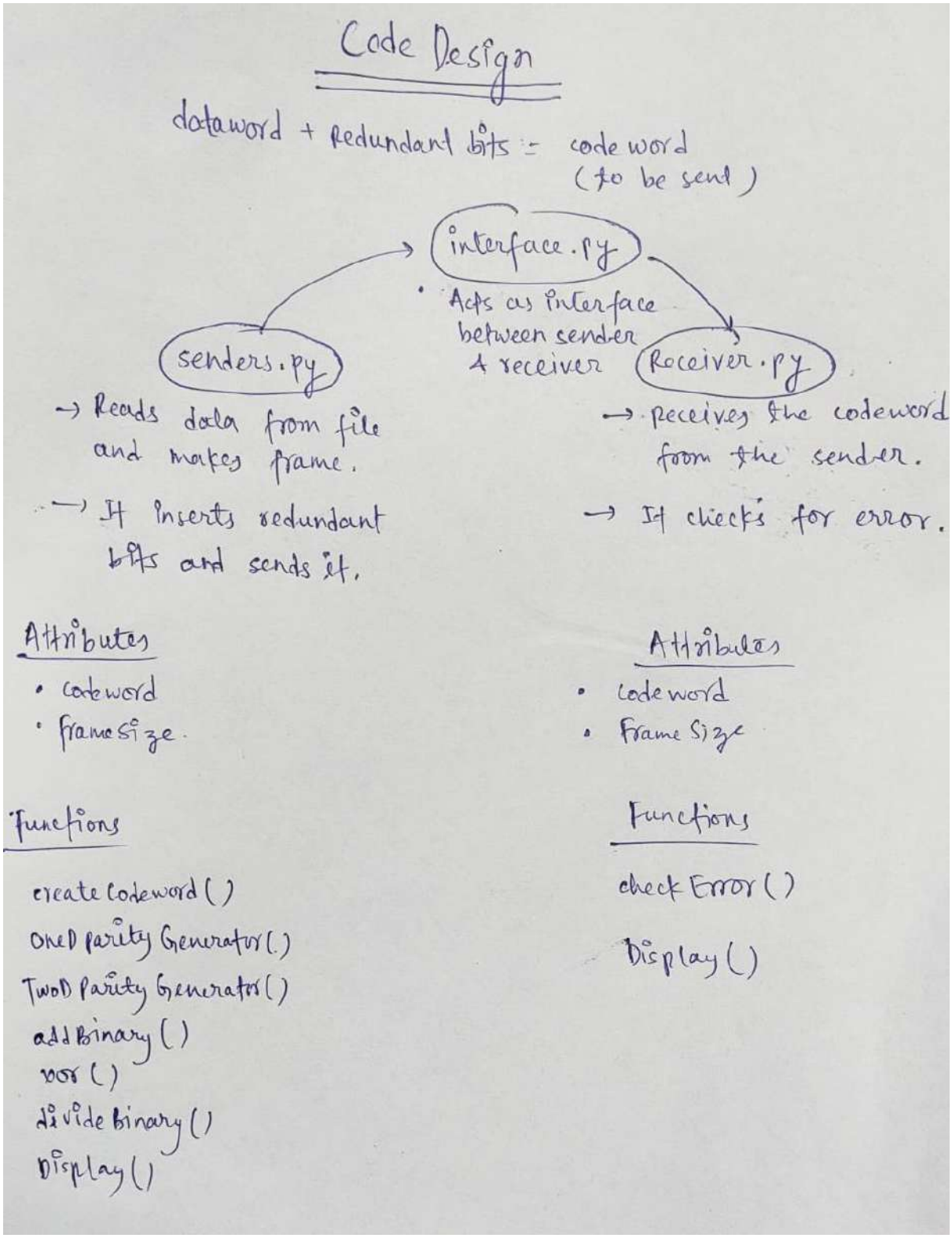
  For example:

  > for the message 1 0 0 1 0 0 with the divisor polynomial 1 1 0 1, the message after appending CRC bits is 1 0 0 1 0 0 0 0 0. We compute CRC on the modified message M.

Design

I have solved the problem using 3 different sub problems:

1. Sending data
2. Receiving data
3. Interface between sender and receiver



1. **senders.py(Task)**
    a. The sequence of 0's ans 1's are read from input file.
    b. This sequence is divided into datawords on the basis of frame size taken as the

input from the user.

c. According to the four schemes namely VRC, LRC, Checksum and CRC, redundant
bits/dataword are added along with the datawords to form codewords.

d. The datawords and codewords which are to be sent are displayed.

e. These encoded codewords are then sent to the receiver.

2. **receiver.py**

a. The codewords are received from the sender.

b. The received codewords are then decoded according to the four schemes namely VRC,
LRC, Checksum and CRC.

c. The result is checked and shown if there is any error detected.

d. The codewords and the datawords extracted from the codewords are also displayed.

3. **interface.py**

a. This program acts as an interface to the above programs – sender.py and receiver.py

b. In this program, the functions for injecting errors are included.

c. There are two separate functions for injecting errors on random positions of codeword, and for injecting errors on specific positions taken as input.

d. A filename is taken as input from command line while executing the program.

e. This is then used as the input file which has a sequence of 0 and 1 stored in it.
The function invocations of Sender class stored in sender.py and Receiver class
stored in receiver.py are done in this file to send and receive the data.

f. The sent data are injected with errors using the above mentioned functions.For
all the following three cases, three different functions have been created to execute a specific case at a moment.
   i. Error is detected by all four scheme
   ii. Error is detected by checksum but not CRC.
   iii. Error is detected by VRC but not CRC.

Implementation

Sender:

```
# sender.py – The following are the tasks performed in this Sender program :
# 1. The input file is read, which contains a sequence of 0 and 1(From input.txt)
# 2. The message sequence is divided into datawords on the basis of frame size(k) taken as
#    the input from the user.
# 3. According to the four schemes namely VRC, LRC, Checksum and CRC, redundant
#    bits/dataword are added along with the datawords to form codewords.
#    datawords + redundant bits = codewords
# 4. The datawords and codewords which are to be sent are displayed.
# 5. These encoded codewords are then sent to the receiver.

class sender:
    # Initialize the sender class
    def __init__(self,size):
        self.codeword = []
        self.frame_size = size

    # Generate the codeword from the dataword according to the scheme
    def createCodeword(self,filename,schemeType,poly=""):
        fileinput=open(filename,"r")
        dataword=fileinput.readline()
        fileinput.close()
        tempword=""
        if schemeType ==1:
            for i in range(len(dataword)):
                if(i!=0 and i%self.frame_size==0):
                    self.codeword.append(tempword)
                    tempword=""
                tempword+=dataword[i]
            self.codeword.append(tempword)
            self.OneDParityGenerator()
        elif schemeType ==2:
            for i in range(len(dataword)):
                if(i!=0 and i%self.frame_size==0):
                    self.codeword.append(tempword)
                    tempword=""
                tempword+=dataword[i]
            self.codeword.append(tempword)
            self.TwoDParityGenerator()
        elif schemeType ==3:
            # Checksum
            for i in range(len(dataword)):
                if(i!=0 and i%self.frame_size==0):
                    self.codeword.append(tempword)
                    tempword=""
                tempword+=dataword[i]
            self.codeword.append(tempword)

            checkSum=self.codeword[0]
```

```
            for i in range(1,len(self.codeword)):
                checkSum=self.addBinary(checkSum,self.codeword[i])
                while(len(checkSum)>self.frame_size):
                    a=checkSum[:len(checkSum)-self.frame_size]
                    b=checkSum[len(checkSum)-self.frame_size:]
                    checkSum=self.addBinary(a,b)
            # Compliment of the checksum
            finalCheckSum=""
            for j in range(len(checkSum)):
                if(checkSum[j]=='0'):
                    finalCheckSum+='1'
                else:
                    finalCheckSum+='0'
            self.codeword.append(finalCheckSum)
        elif schemeType ==4:

            for i in range(len(dataword)):
                if i>0 and i%self.frame_size==0:
                    tempword+='0'*(len(poly)-1)
                    remainder=self.divideBinary(tempword,poly)
                    remainder=remainder[len(remainder)-(len(poly)-1):]
                    tempword=tempword[:self.frame_size]
                    tempword+=remainder
                    self.codeword.append(tempword)
                    tempword=""
                tempword+=dataword[i]
            tempword+='0'*(len(poly)-1)
            remainder=self.divideBinary(tempword,poly)
            remainder=remainder[len(remainder)-(len(poly)-1):]
            tempword=tempword[:self.frame_size]
            tempword+=remainder
            self.codeword.append(tempword)




    def OneDParityGenerator(self):
        for i in range(len(self.codeword)):
            countOnes=0
            for j in range(len(self.codeword[i])):
                if(self.codeword[i][j]=='1'):
                    countOnes+=1
            if(countOnes%2==0):
                self.codeword[i]+='0'
            else:
               self.codeword[i]+='1'



    def TwoDParityGenerator(self):

        parity=""
        index=0
        while index<self.frame_size:
            countOnes=0
            codeWordIndex=0;
            while codeWordIndex<len(self.codeword):
                if(self.codeword[codeWordIndex][index]=='1'):
                    countOnes+=1
```

```
                codeWordIndex+=1
            if(countOnes%2==0):
                parity+='0'
            else:
                parity+='1'
            index+=1
        self.codeword.append(parity)
    # Add the two binary numbers of the same length
    def addBinary(self,a,b):
        result=""
        a=a[::-1]
        b=b[::-1]
        carry=0

        for i in range(len(a)):
            DigitA=ord(a[i])-ord('0')
            DigitB=ord(b[i])-ord('0')
            total=DigitA+DigitB+carry

            char=str(total%2)
            result=char+result
            carry=total//2

        if carry==1:
            result='1'+result
        return result

    def xor(self, a, b):
        result = ""
        for i in range(1, len(b)):
            if a[i]==b[i]:
                result += '0'
            else:
                result += '1'
        return result

  #Helper function to divide two binary sequence
    def divideBinary(self, dividend, divisor):
        xorlen = len(divisor)
        temp = dividend[:xorlen]
        while len(dividend) > xorlen:
            if temp[0]=='1':
                temp=self.xor(divisor,temp)+dividend[xorlen]
            else:
                temp=self.xor('0'*xorlen,temp)+dividend[xorlen]
            xorlen += 1
        if temp[0]=='1':
            temp=self.xor(divisor,temp)
        else:
            temp=self.xor('0'*xorlen,temp)
        return temp

    def Display(self,schemeType):
        dataword=[]
        if schemeType ==1:
            print("VRC Scheme")
            print("Dataword : ",end="")
            for i in self.codeword:
                dataword.append(i[:self.frame_size])
```

```
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
        elif schemeType ==2:
            parity=self.codeword[len(self.codeword)-1]
            print("LRC Scheme")
            print("Dataword : ",end="")
            for i in range(len(self.codeword)-1):
                dataword.append(self.codeword[i])
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
            print("Parity : ",end="")
            print(parity)
        elif schemeType ==3:
            checkSum=self.codeword[len(self.codeword)-1]
            # push all elements except the checksum
            for i in range(len(self.codeword)-1):
                dataword.append(self.codeword[i])
            print("Checksum Scheme")
            print("Dataword : ",end="")
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
            print("Checksum : ",end="")
            print(checkSum)
        elif schemeType ==4:
            print("CRC Scheme")
            print("Dataword : ",end="")
            for i in range(len(self.codeword)):
                dataword.append(self.codeword[i])
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
```

Receiver:

```
class receiver:
    def __init__ (self,s):
        self.codeword=s.codeword
        self.frame_size=s.frame_size


    def checkError(self,schemeType,poly=""):
        if schemeType ==1:
            self.OneDParityCheck()
        elif schemeType ==2:
            self.TwoDParityCheck()
        elif schemeType ==3:
            self.checksumCheck()
        elif schemeType ==4:
            self.checkCRCError(poly)
        else:
```

```
            print("Invalid Scheme")

    def checkCRCError(self,poly):
        for i in range(len(self.codeword)):
            remainder = self.divideBinary(self.codeword[i], poly)
            error = False
            for j in range(len(remainder)):
                if remainder[j] == '1':
                    error = True
            print("Remainder:",remainder,end=' ')
            if error:
                print("ERROR DETECTED")
            else:
                print("NO ERROR DETECTED")
            # exit the loop if error is detected



    def OneDParityCheck(self):

        flag=True
        for i in range(len(self.codeword)):
            countOnes=0
            for j in range(len(self.codeword[i])):
                if(self.codeword[i][j]=='1'):
                    countOnes+=1
            if(countOnes%2!=0):
                print("ERROR is detected by VRC")
                flag=False
                break
        if flag==True:
            print("No Error is detected by VRC")



    def TwoDParityCheck(self):
        parity=""
        index=0
        while index<self.frame_size:
            countOnes=0
            codeWordIndex=0;
            while codeWordIndex<len(self.codeword)-1:
                if(self.codeword[codeWordIndex][index]=='1'):
                    countOnes+=1
                codeWordIndex+=1
            if(countOnes%2==0):
                parity+='0'
            else:
                parity+='1'
            index+=1
        if(parity==self.codeword[len(self.codeword)-1]):
            print("No Error is detected by LRC")
        else:
            print("ERROR is detected by LRC")
        return parity

    def checksumCheck(self):
        checkSum=self.codeword[0]
        for i in range(1,len(self.codeword)):
```

```
            checkSum=self.addBinary(checkSum,self.codeword[i])
            while(len(checkSum)>self.frame_size):
                a=checkSum[:len(checkSum)-self.frame_size]
                b=checkSum[len(checkSum)-self.frame_size:]
                checkSum=self.addBinary(a,b)
            # Compliment of the checksum
        finalCheckSum=""
        for j in range(len(checkSum)):
            if(checkSum[j]=='0'):
                finalCheckSum+='1'
            else:
                finalCheckSum+='0'
        #convert the finalCheckSum to int
        val=int(finalCheckSum,2)
        if(val==0):
            print("No Error is detected by Checksum")
        else:
            print("ERROR is detected by Checksum")
    def addBinary(self,a,b):
        result=""
        a=a[::-1]
        b=b[::-1]
        carry=0

        for i in range(max(len(a),len(b))):
            DigitA=ord(a[i])-ord('0') if i<len(a) else 0
            DigitB=ord(b[i])-ord('0') if i<len(b) else 0
            total=DigitA+DigitB+carry

            char=str(total%2)
            result=char+result
            carry=total//2

        if carry==1:
            result='1'+result
        return result
    def Display(self,schemeType):
        # display the codeword
        dataword=[]
        if schemeType ==1:
            print("VRC Scheme")
            print("Dataword : ",end="")
            for i in self.codeword:
                dataword.append(i[:self.frame_size])


            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
        elif schemeType ==2:
            parity=self.codeword[len(self.codeword)-1]
            print("LRC Scheme")
            print("Dataword : ",end="")
            for i in range(len(self.codeword)-1):
                dataword.append(self.codeword[i])
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
            print("Parity : ",end="")
```

```
            parity=""
            index=0
            while index<self.frame_size:
                countOnes=0
                codeWordIndex=0;
                while codeWordIndex<len(self.codeword)-1:
                    if(self.codeword[codeWordIndex][index]=='1'):
                        countOnes+=1
                    codeWordIndex+=1
                if(countOnes%2==0):
                    parity+='0'
                else:
                    parity+='1'
                index+=1
            print(parity)

        elif schemeType ==3:
            checkSum=self.codeword[len(self.codeword)-1]
            # push all elements except the checksum
            for i in range(len(self.codeword)-1):
                dataword.append(self.codeword[i])
            print("Checksum Scheme")
            print("Dataword : ",end="")
            print(dataword)
            print("Codeword : ",end="")
            print(self.codeword)
            print("Checksum : ",end="")
            print(checkSum)



    #Helper function to divide two binary sequence
      def divideBinary(self, dividend, divisor):
          xorlen = len(divisor)
          temp = dividend[:xorlen]
          while len(dividend) > xorlen:
              if temp[0]=='1':
                  temp=self.xor(divisor,temp)+dividend[xorlen]
              else:
                  temp=self.xor('0'*xorlen,temp)+dividend[xorlen]
              xorlen += 1
          if temp[0]=='1':
              temp=self.xor(divisor,temp)
          else:
              temp=self.xor('0'*xorlen,temp)
          return temp
      def xor(self, a, b):
          result = ""
          for i in range(1, len(b)):
              if a[i]==b[i]:
                  result += '0'
              else:
                  result += '1'
          return result
```

Interface.py

```
from senders import *
from receiver import *
import random
import sys

#function to inject errors in random positions
def injectRandomError(frames):
  for i in range(len(frames)):
    pos = random.randint(0, len(frames[i])-1)
    frames[i] = frames[i][:pos]+'1'+frames[i][pos+1:]
  return frames

def injectSpecificError(frames, zeropos, onepos):
  for i in range(len(zeropos)):
    for j in range(len(zeropos[i])):
      pos = zeropos[i][j]
      frames[i] = frames[i][:pos]+'0'+frames[i][pos+1:]
  for i in range(len(onepos)):
    for j in range(len(onepos[i])):
      pos = onepos[i][j]
      frames[i] = frames[i][:pos]+'1'+frames[i][pos+1:]
  return frames


def case1(filename,size):
    # changing 1 bit in the codeword
    print("CASE 1:")
    print(" VRC ")
    type=1
    s=sender(size)
    s.createCodeword(filename,type)
    s.Display(type)
    s.codeword=injectRandomError(s.codeword)
    r=receiver(s)
    r.checkError(type)
    r.Display(type)

    print(" LRC ")
    type=2
    s=sender(size)
    s.createCodeword(filename,type)
    s.Display(type)
    s.codeword=injectRandomError(s.codeword)
    r=receiver(s)
    r.checkError(type)
    r.Display(type)

    print("CheckSum")
    type=3
    s=sender(size)
    s.createCodeword(filename,type)
    s.Display(type)
    s.codeword=injectRandomError(s.codeword)
    r=receiver(s)
    r.checkError(type)
    r.Display(type)
```

```
    print("CRC")
    poly="1001"
    type=4
    s=sender(size)
    s.createCodeword(filename,type,poly)
    s.Display(type)
    s.codeword=injectRandomError(s.codeword)
    r=receiver(s)
    r.checkError(type,poly)
    r.Display(type)

def case2(filename,size):
    print("Case 2")
    print("CheckSum")
    type=3

    zeropostion=[]
    oneposition=[[5]]
    s=sender(size)
    s.createCodeword(filename,type)
    s.Display(type)
    s.codeword=injectSpecificError(s.codeword,zeropostion,oneposition)
    r=receiver(s)
    r.checkError(type)
    r.Display(type)

    print("CRC")
    poly="100"
    type=4
    s=sender(size)
    s.createCodeword(filename,type,poly)
    s.Display(type)
    s.codeword=injectSpecificError(s.codeword,zeropostion,oneposition)
    r=receiver(s)
    r.checkError(type,poly)
    r.Display(type)

def case3(filename,size):
  print("Case 3")
  print("VRC")
  type=1

  zeropostion=[]
  oneposition=[[5]]
  s=sender(size)
  s.createCodeword(filename,type)
  s.Display(type)
  s.codeword=injectSpecificError(s.codeword,zeropostion,oneposition)
  r=receiver(s)
  r.checkError(type)
  r.Display(type)

  print("CRC")
  poly="100"
  type=4
  s=sender(size)
  s.createCodeword(filename,type,poly)
  s.Display(type)
  s.codeword=injectSpecificError(s.codeword,zeropostion,oneposition)
```

```
  r=receiver(s)
  r.checkError(type,poly)
  r.Display(type)
```

```
if __name__=='__main__':
    print("1. Error is Detected by All Schemes")
    print("2. Error is Detected by Checksum but not CRC ")
    print("3. Error is Detected by VRC but not by CRC")
    case=int(input())
    print("Enter file name ")
    filename=input()
    print("Enter size of frame")
    frameSize=int(input())

    if case==1:
        case1(filename,frameSize)
    elif case==2:
        case2(filename,frameSize)
    elif case==3:
        case3(filename,frameSize)
```

Test Cases

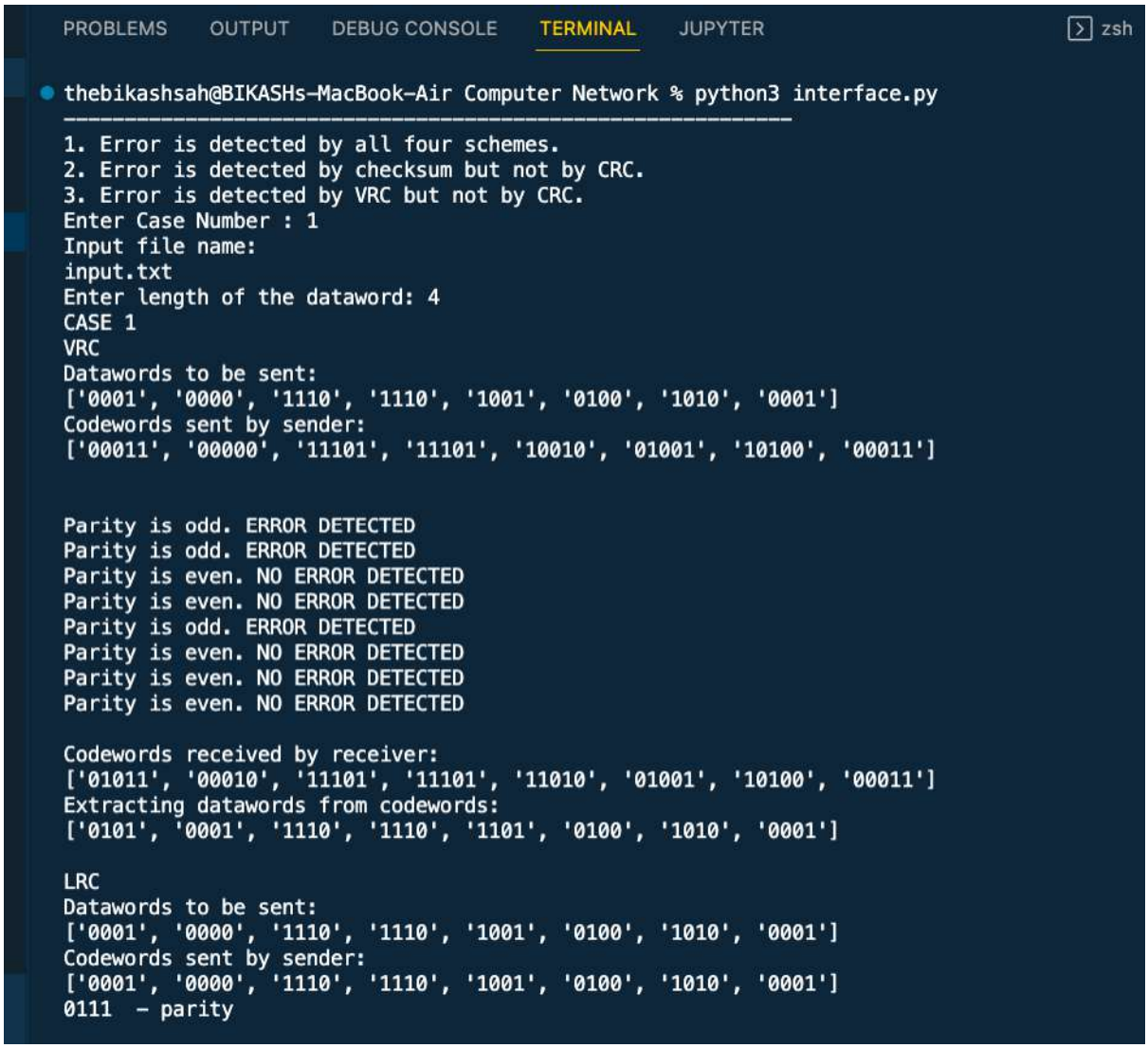
We had to check 3, cases as per as the question.

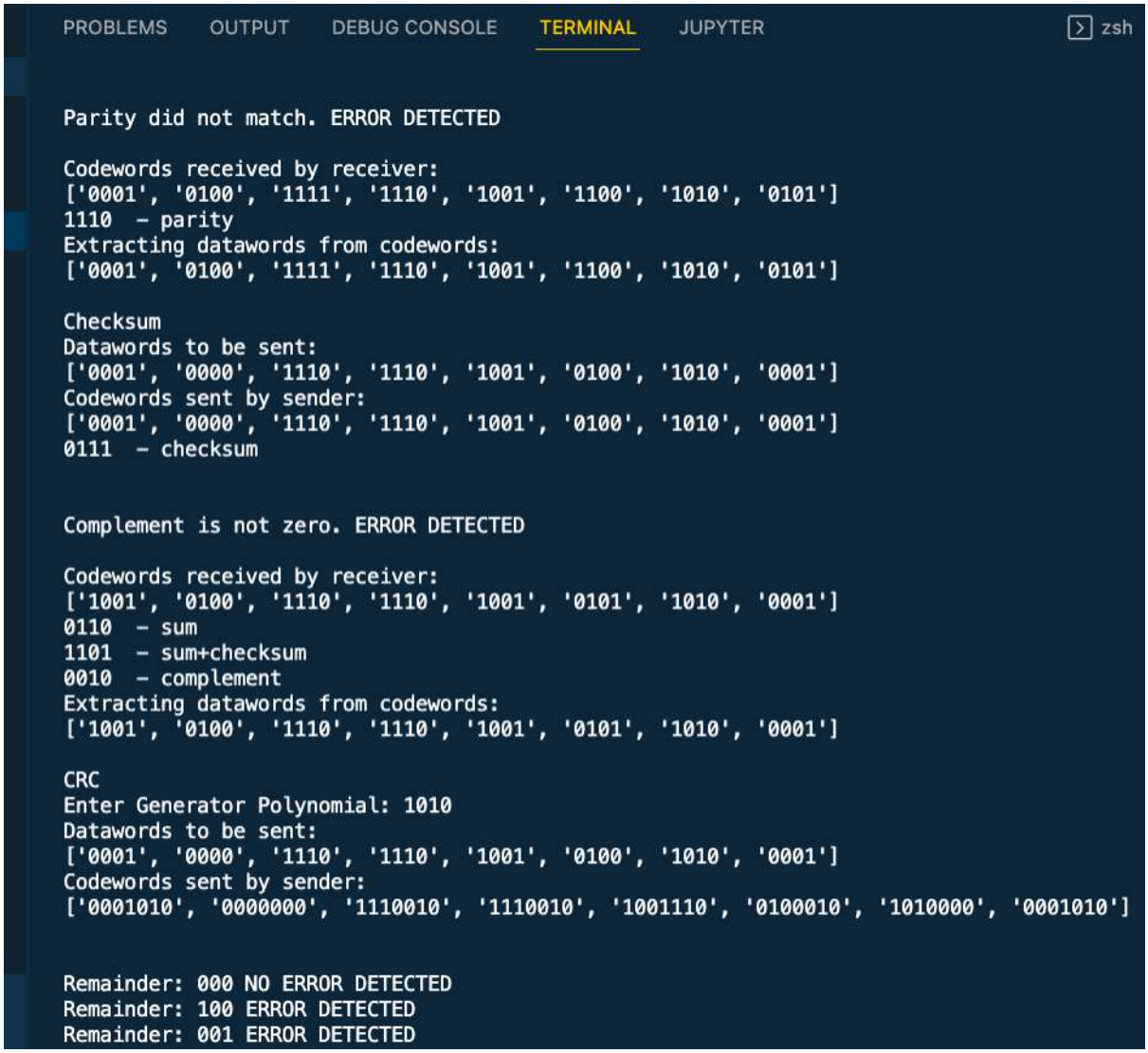
- Error is Detected by All Schemes
    - Here I have injected a random error at a bit position, which is detected by all the schemes.
- Error is Detected by Checksum but not CRC
    - As we know that if the degree of the dividing polynomial is less than the degree of the codeword and the codeword is also divisible by the polynomial then the CRC cannot detect error.
    - So, in the implementation I have designed a test case that checksum and easily detect but due to the choice of polynomial, CRC Couldn’t detect it.
- Error is Detected by VRC but not by CRC

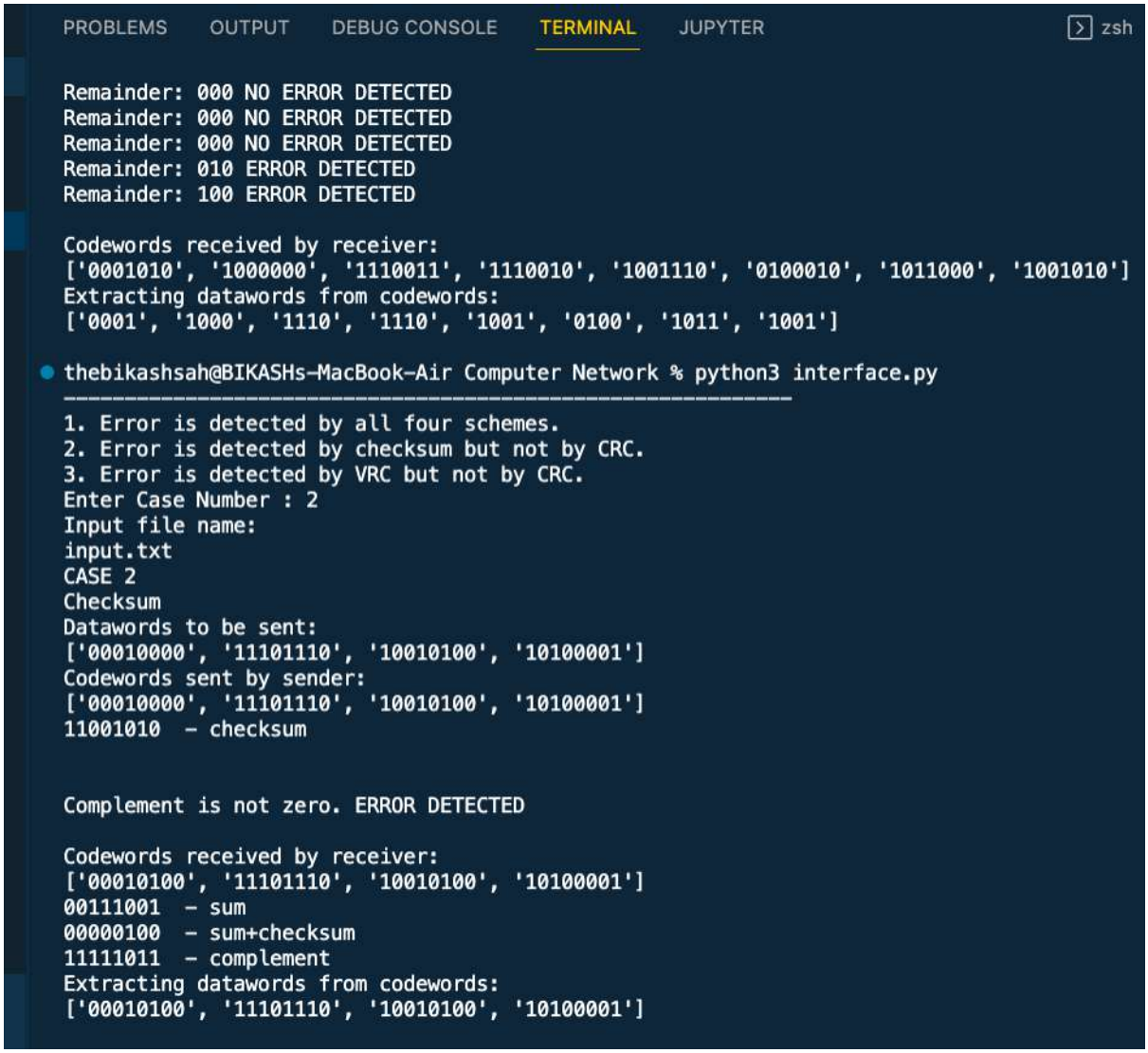
- It is similar to the above test case used in point 2.

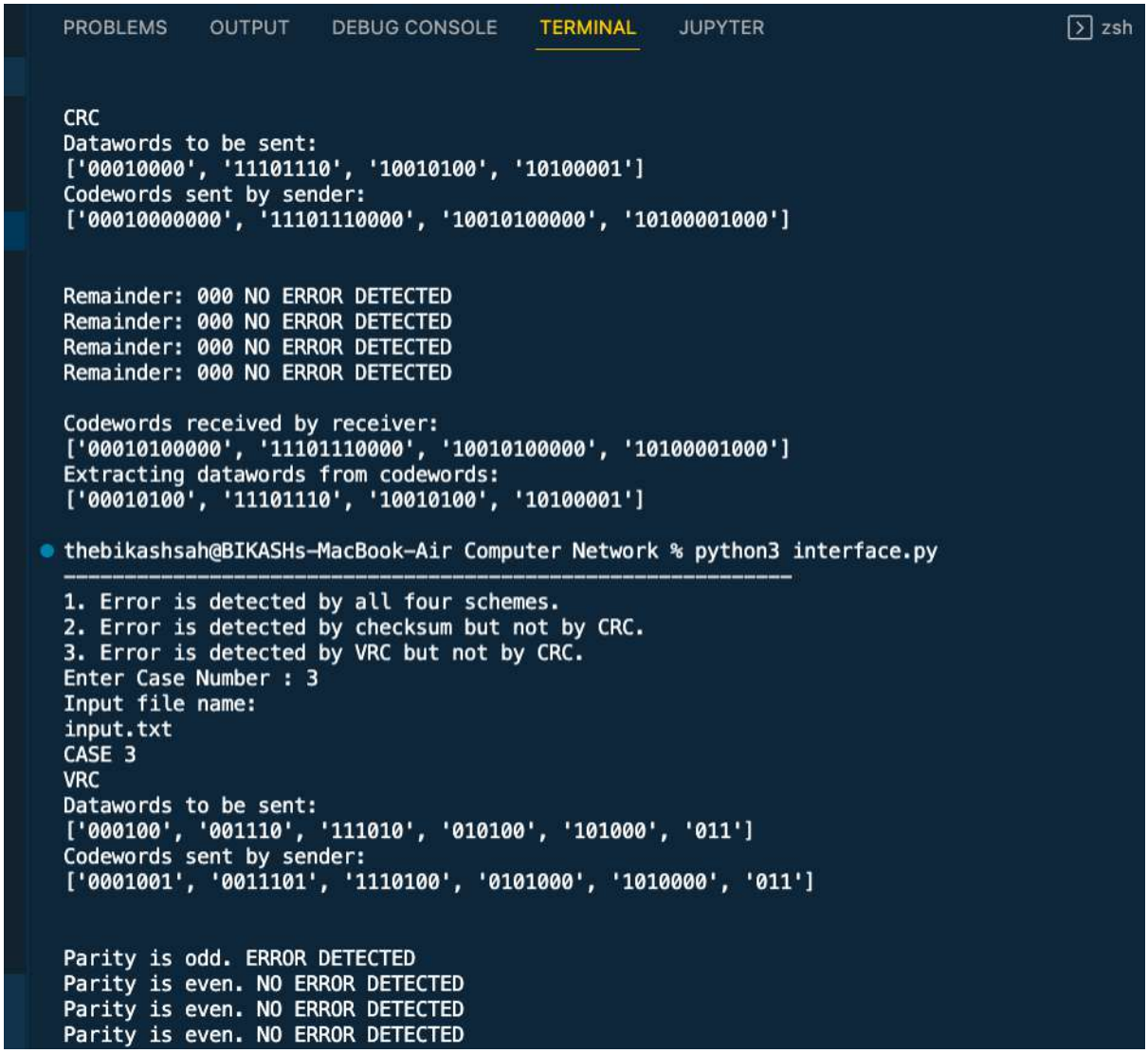
Results

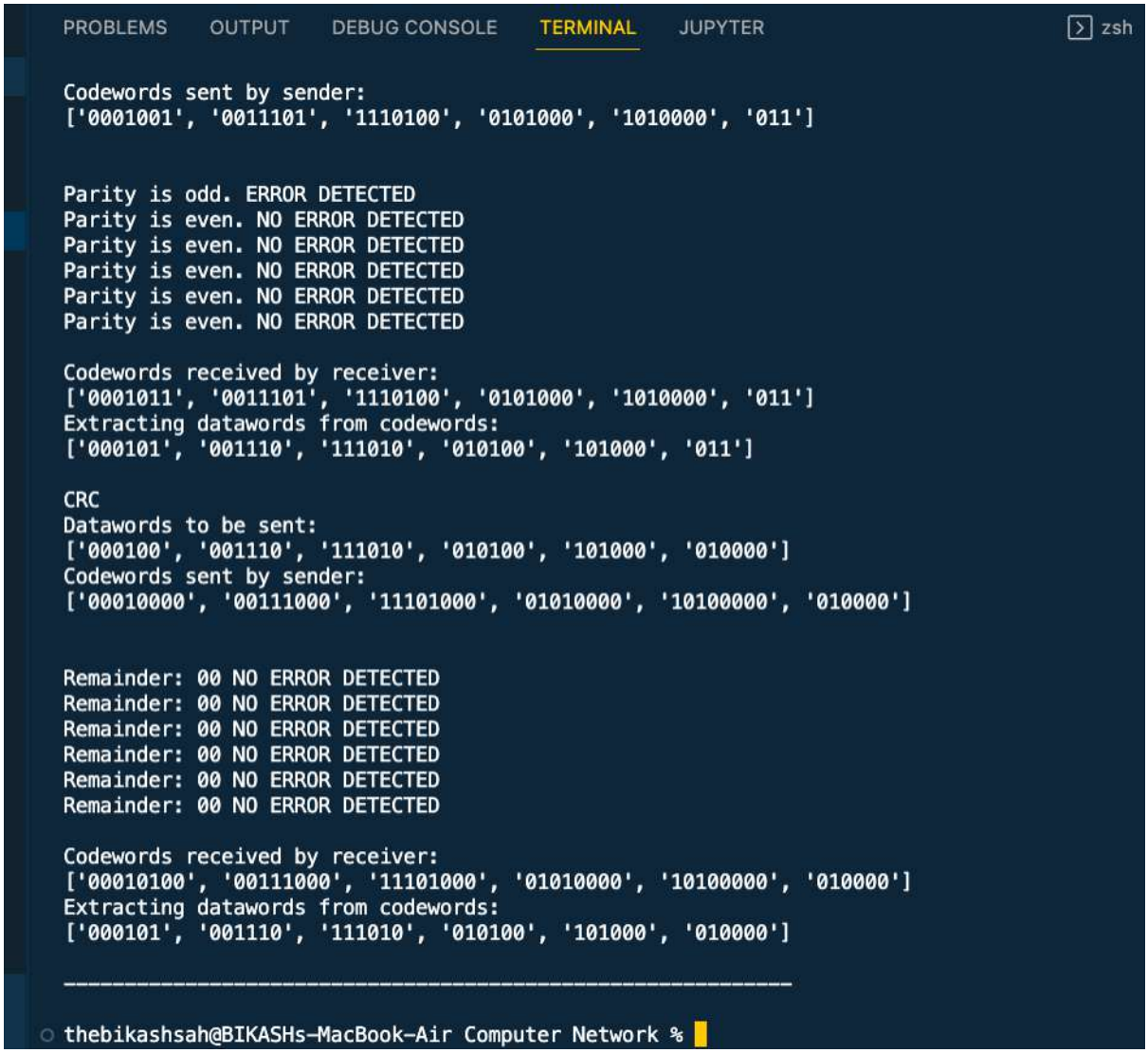
Here, all the test cases have been checked.




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
zsh
Parity did not match. ERROR DETECTED
Codewords received by receiver:
['0001', '0100', '1111', '1110', '1001', '1100', '1010', '0101']
1110 - parity
Extracting datawords from codewords:
['0001', '0100', '1111', '1110', '1001', '1100', '1010', '0101']
Checksum
Datawords to be sent:
['0001', '0000', '1110', '1110', '1001', '0100', '1010', '0001']
Codewords sent by sender:
['0001', '0000', '1110', '1110', '1001', '0100', '1010', '0001']
0111 - checksum
Complement is not zero. ERROR DETECTED
Codewords received by receiver:
['1001', '0100', '1110', '1110', '1001', '0101', '1010', '0001']
0110 - sum
1101 - sum+checksum
0010 - complement
Extracting datawords from codewords:
['1001', '0100', '1110', '1110', '1001', '0101', '1010', '0001']
CRC
Enter Generator Polynomial: 1010
Datawords to be sent:
['0001', '0000', '1110', '1110', '1001', '0100', '1010', '0001']
Codewords sent by sender:
['0001010', '0000000', '1110010', '1110010', '1001110', '0100010', '1010000', '0001010']
Remainder: 000 NO ERROR DETECTED
Remainder: 100 ERROR DETECTED
Remainder: 001 ERROR DETECTED


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
zsh
Remainder: 000 NO ERROR DETECTED
Remainder: 000 NO ERROR DETECTED
Remainder: 000 NO ERROR DETECTED
Remainder: 010 ERROR DETECTED
Remainder: 100 ERROR DETECTED
Codewords received by receiver:
['0001010', '1000000', '1110011', '1110010', '1001110', '0100010', '1011000', '1001010']
Extracting datawords from codewords:
['0001', '1000', '1110', '1110', '1001', '0100', '1011', '1001']
thebikashsah@BIKASHs-MacBook-Air Computer Network % python3 interface.py
1. Error is detected by all four schemes.
2. Error is detected by checksum but not by CRC.
3. Error is detected by VRC but not by CRC.
Enter Case Number : 2
Input file name:
input.txt
CASE 2
Checksum
Datawords to be sent:
['00010000', '11101110', '10010100', '10100001']
Codewords sent by sender:
['00010000', '11101110', '10010100', '10100001']
11001010 - checksum
Complement is not zero. ERROR DETECTED
Codewords received by receiver:
['00010100', '11101110', '10010100', '10100001']
00111001 - sum
00000100 - sum+checksum
11111011 - complement
Extracting datawords from codewords:
['00010100', '11101110', '10010100', '10100001']


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
zsh
CRC
Datawords to be sent:
['00010000', '11101110', '10010100', '10100001']
Codewords sent by sender:
['00010000000', '11101110000', '10010100000', '10100001000']
Remainder: 000 NO ERROR DETECTED
Remainder: 000 NO ERROR DETECTED
Remainder: 000 NO ERROR DETECTED
Remainder: 000 NO ERROR DETECTED
Codewords received by receiver:
['00010100000', '11101110000', '10010100000', '10100001000']
Extracting datawords from codewords:
['00010100', '11101110', '10010100', '10100001']
thebikashsah@BIKASHs-MacBook-Air Computer Network % python3 interface.py
1. Error is detected by all four schemes.
2. Error is detected by checksum but not by CRC.
3. Error is detected by VRC but not by CRC.
Enter Case Number : 3
Input file name:
input.txt
CASE 3
VRC
Datawords to be sent:
['000100', '001110', '111010', '010100', '101000', '011']
Codewords sent by sender:
['0001001', '0011101', '1110100', '0101000', '1010000', '011']
Parity is odd. ERROR DETECTED
Parity is even. NO ERROR DETECTED
Parity is even. NO ERROR DETECTED
Parity is even. NO ERROR DETECTED



Analysis

Analysis of senders.py

- `createCodeword(self,filename,schemeType,poly="")`

  We take input as the filename type(type of technique) and the polynomial if it is CRC.

  Here we read the entire sequence of 0's ans 1's form the file and then divide the

  sequence into data frames of the required length that is also given by the user.

  After dividing into frames we append each message with the redundant bits receptive to

  the type of error technique;

- `Display(self,schemeType)`
  This method simply displays the frames with the corresponding redundant bits on console.
- `OneDParityGenerator(self)`
  This method is used to generate paraty in case of VRC.
- `TwoDParityGenerator(self)`
  This method is used to generate paraty in case of LRC.
- `addBinary(self,a,b)`
  This method has been used as a helper function for implementing Checksum. It takes
  two binary sequence as parameters, and returns the result of wrapped addition of them.
- `xor(self, a, b)`
  This method has been used as a helper function for implementing CRC. In this method,
  two binary sequence is taken as input from its arguments, and return the xor of them.
- `divideBinary(self, dividend, divisor)`
  This method has been used as a helper function for implementing CRC. In this method,
  dividend and divisor is taken as input from its arguments. The division is performed
  using mod 2 arithmetic (exclusice-OR) on the message using divisor polynomial, and
  the remainder is returned.

Analysis of receivers.py

- `checkError(self,schemeType,poly="")`

  This method takes type (scheme) of error detection method as one of its argument. On
  the basis of the scheme the codewords received is checked for error. If there is an error,
  appropriate message is displayed on the screen.

Rest functions are similar to senders.py

Analysis of interface.py

- `injectRandomError(frames)`

  This method is used to inject errors in random
  positions of the codewords which are sent by sender program. The randint() function of
  random python module has been used for generating random position within the size of
  the codewords.

- `injectSpecificError(frames, zeropos, onepos)`

  This method is used to inject errors in specific positions of the codewords. The positions in which the error is to be inserted is passed as arguments, as zeropos and onepos. Zeropos contains the list of positions in which the value of those positions will be made 0, whereas Onepos containes the list of positions in which the value of those positions will be made 1.

- `case1(filename,size)`

  This method is implemented to make function invocations and displaying the results of all schemes, considering the case 1 of the problem statement (Error is detected by all four schemes). The dataword size and input file name is taken as function arguments.

- `case2(filename,size)`

  This method is implemented to make function invocations and displaying the results of Checksum and CRC, such that the case 2 of the problem statement satisfies (Error is detected by Checksum but not CRC). The dataword size and input file name is taken as function arguments.

- `case3(filename,size)`

  This method is implemented to make function invocations and displaying the results of VRC and CRC, such that the case 3 of the problem statement satisfies (Error is detected by VRC but not CRC). The dataword size and input file name is taken as function arguments.

Comments

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot,

> I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.

Bikash_Sah_002010501018_Computer Networks Lab Report_2

Name: Bikash Sah

Class: BCSE-3

Group: A1

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.

Assignment 2: Implement three data link layer protocols, Stop and Wait, Go Back N Sliding Window and Selective Repeat Sliding Window for flow control.

Submission due: 22-26 August 2022 (in your respective lab classes)

Report submission due on: 28 August 2022

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 re-compute 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: **23 August 2022**

Deadline: **28 August 2022**

Introduction

Flow control is a speed matching mechanism between the receiver and sender. Flow control coordinates the amount of data that can be send before receiving an acknowledgement.

Some flow control protocols are:

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**

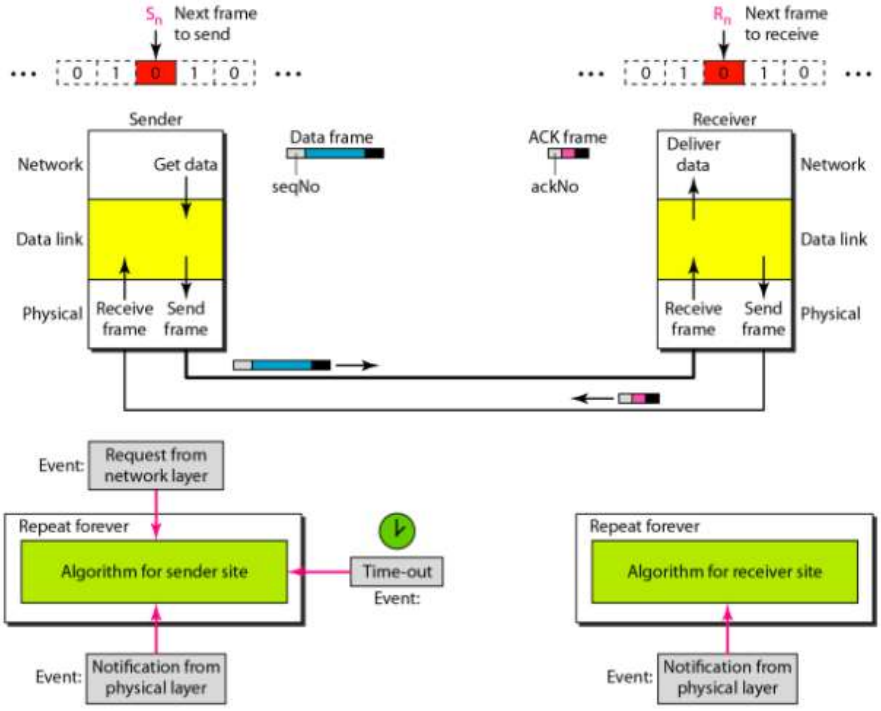
Design

1. **Stop and Wait ARQ (Automatic Repeat Request Protocol)**
    - After sending one frame, the sender waits for an acknowledgement before sending the next frame.
    - If the acknowledgement doesn't arrive after a certain amount of time then the sender times out and retransmit the original frame.
    - Stop and Wait ARQ = Simple stop and wait + timeout timer + sequence number.

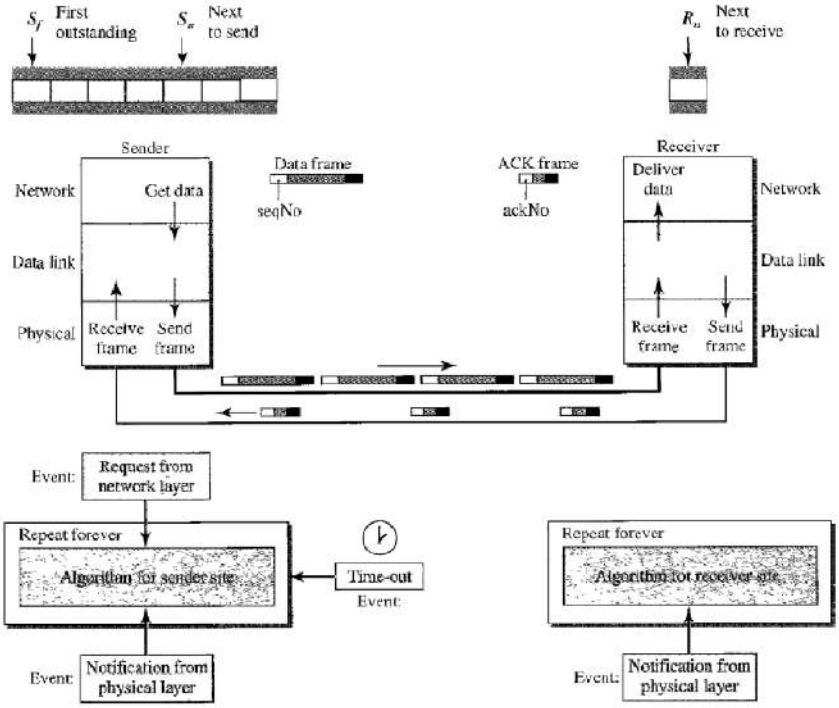
    4 Cases

    1. Frame and ACK are sent and received within time.
    2. Frame is lost.
    3. ACK is lost.
    4. ACK is received after timeout.

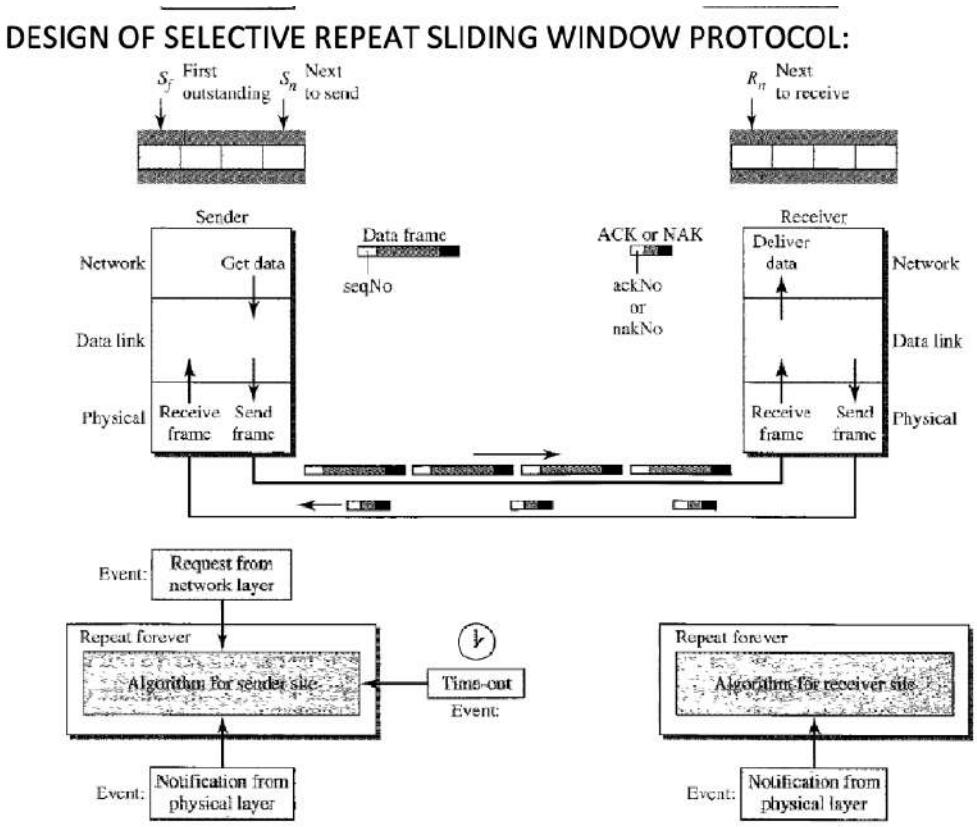
Design of the Stop-and-Wait ARQ Protocol



DESIGN OF GO BACK N SLIDING WINDOW PROTOCOL:



DESIGN OF SELECTIVE REPEAT SLIDING WINDOW PROTOCOL:



Implementation

Stop and Wait:

Sender:

```
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)
```

Receiver:

```
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)
```

Interface:

```
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))
          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,':', str(rdata))
          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__':
  #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:

Sender:

```
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
    '''rdata = mySocket.recv(1024).decode()
    print('Received from channel :',str(rdata))
    curtime = time.time()
    print('Round trip time: ',str(curtime-prevtime))
    '''
    '''
    time.sleep(0.005)
    filein = open('flag.txt',"r")
    flag = int(filein.read())
    filein.close()
    while flag == 1:
      rdata = mySocket.recv(1024).decode()
      print(rdata)

      cnt = extractCount(rdata)
      msg = extractMessage(sentframes[cnt])
      stat = extractStatus(rdata)
      print(msg,cnt,stat)
      print('Again Sending to channel :',str(msg))
      data = msg +'/'+cnt+'/'
      mySocket.send(data.encode())

      time.sleep(0.005)
      filein = open('flag.txt',"r")
      flag = int(filein.read())
      filein.close()

    '''
  mySocket.close()

if __name__ == '__main__':
  if len(sys.argv) > 1:
    senderno = int(sys.argv[1])
  else:
    senderno = 1
  Main(senderno)
```

Receiver:

```
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)
```

Interface:

```
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])


        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 % windowsize)+1))
        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 == 'TIMEOUT':
                flag = 1
                break
              idx += 1
            print(' ------------------------------ ')
            if flag==1:
              print('RESEND FROM FRAME NO:',str(idx+1))
            else:
              print('BLOCK OF WINDOW SIZE',self.windowsize,'SUCCESSFULLY SENT')
            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]

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

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

```
                print('Current frame no:',str(idx+1))
                print('Again Sending to Receiver',recvno+1)

                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

                msg = extractMessage(data)
                cnt = extractCount(data)
                stat = extractStatus(data)
                curtime = time.time()
                print(msg,str(cnt),stat)
                print('Round trip time: ',str(curtime-prevtime))
                self.slidingwindow[idx][1] = data
                idx += 1

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

if __name__ == '__main__':
  #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:

Sender:

```
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)
```

Receiver:

```
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)
```

Interface:

```
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])


          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 % windowsize)+1))
          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)

                if status == 'NAK' or status == 'TIMEOUT':
                  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)

                print(' ------------------------------ ')
                print('RESENDING FRAME NO:',str(indices[idx]))
                print(' ------------------------------ ')
                #sending all frames to its sender from first NAK
                #conn[0].send(currframe.encode())

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

                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

                msg = extractMessage(data)
                cnt = extractCount(data)
                stat = extractStatus(data)
                curtime = time.time()
                print(msg,str(cnt),stat)
                print('Round trip time: ',str(curtime-prevtime))
                self.slidingwindow[indices[idx]-1][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__':
  #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()
```

Computer Networks Lab Report - Assignment 3

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 3

Problem Statement:

Assignment 3: Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

Submission due: 12th -16th *September* 2022

In this assignment, you have to implement 1-persistent, non-persistent and p-persistent CSMA techniques. Measure the performance parameters like throughput (i.e., average amount of data bits successfully transmitted per unit time) and forwarding delay (i.e., average end-to-end delay, including the queuing delay and the transmission delay) experienced by the CSMA frames (IEEE 802.3). Plot the comparison graphs for throughput and forwarding delay by varying p. State your observations on the impact of performance of different CSMA techniques.

Submission Date: **21 Nov, 2022**

Deadline: 16th September, 2022

Introduction

1. 1-Persistent
2. Non-persistent
3. p-persistent

1-Persistent

The 1-persistent method is simple and straightforward. In this method, after the station finds the line idle, it sends its frame immediately (with probability 1). This method has the highest
chance of collision because two or more stations may find the line idle and send their frames immediately.

Non-Persistent

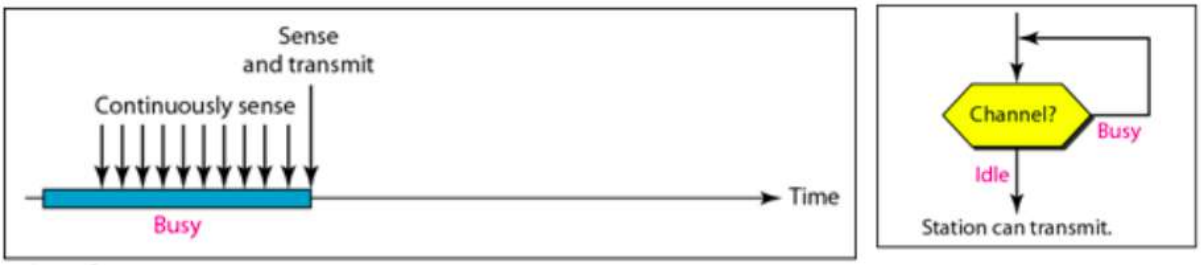
In the nonpersistent method, a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits for a random amount of time and then senses the line again. The non persistent approach reduces the chance of collision because it is
unlikely that two or more stations will wait the same amount of time and retry to send simultaneously.

P-Persistent

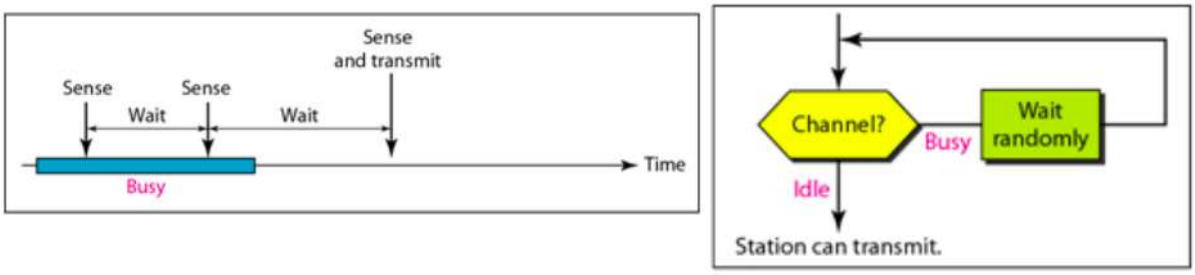
The p-persistent method uses advantages of both other strategies. It reduces the collision and improves the efficiency.

Design

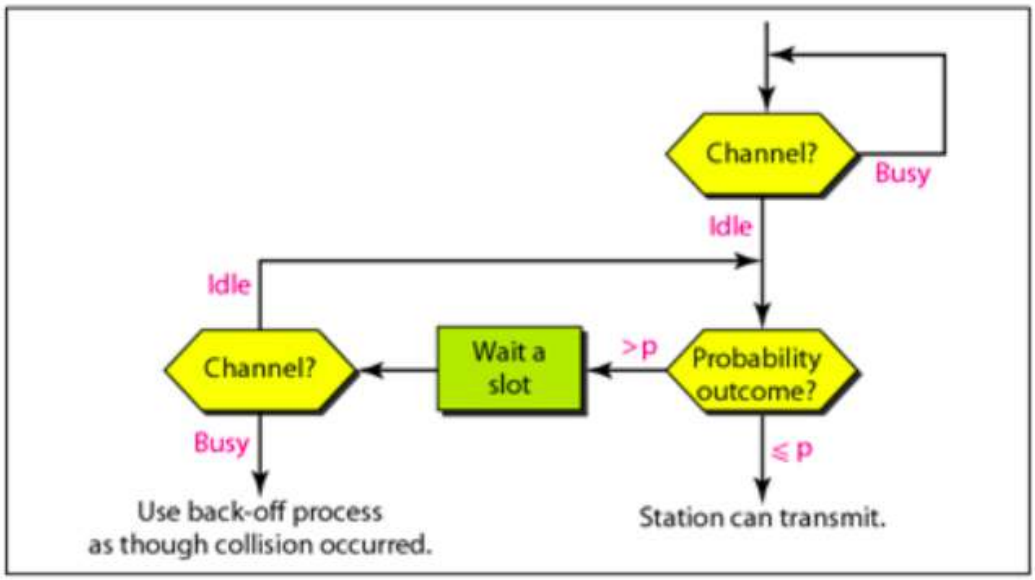
1-Persistent



Non-Persistent



P-Persistent



Implementation

P-Persistent:

```
import threading
import time

frameTime = 3
interFrameTime = 1

numFrames = 2
totalFrames = 0


# function to determine channel utilization of 1p CSMA
# Channel utilization is the percentage of time that the channel is busy (i.e. the per
centage of time that the channel is not idle)
def utility(lock, total): # It is the fraction of time used to transmit data packets.
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    channelUtil = used / tot
    # Channel Utilization to 2 decimal places
    print("-----------------------------------------------------")
    print("Channel Utilization: " + str(round(channelUtil, 4)))
    print("-----------------------------------------------------")

    print()


# CSMA class using threading
class CSMA(threading.Thread):
    def __init__(self, lock, k, index): # initialize class with lock, k, and index val
ues
        super().__init__()
        self.lock = lock
        self.k = k
        self.index = index

    def run(self):
        cnt = 1
        global numFrames
        global totalFrames
        while cnt <= numFrames:
            print("-----------------------------------------------------")
            print("Thread " + str(self.index) + " is trying to acquire lock")
            print(f"Attempting to send frame {cnt} of node {self.index}")
            print()

            while self.lock.locked():
                pass

            self.lock.acquire()
            time.sleep(frameTime)
            print(f"Successfully sent frame {cnt} of node {self.index}")
            print()
```

```
            self.lock.release()
            totalFrames += 1
            time.sleep(interFrameTime)
            cnt += 1
        return


if __name__ == '__main__':
    numberNodes = int(input("Enter number of nodes: ")) # Take number of nodes as inpu
t

    lock = threading.Lock() # Create lock object which will be used by all threads
    met = threading.Thread(target=utility, args=[lock, numberNodes * numFrames]) # Cre
ate thread to calculate channel utilization

    Nodes = [CSMA(lock, 4, i + 1) for i in range(0, numberNodes)] # Create list of thr
eads for each node in the network
    met.start() # Start thread to calculate channel utilization
    for node in Nodes: # Start all threads
        node.start()

    for node in Nodes: # Wait for all threads to finish
        node.join()
    met.join() # Wait for channel utilization thread to finish
```

Non-Persistent:

```
import threading
import random
import time

frameTime = 3
interFrameTime = 1
numFrames = 2
totalFrames = 0


def metrics(lock, total):
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    print("-------------------------------------------------")
    # Channel Utilization to 4 decimal places
    print("Channel Utilization: " + str(round(used / tot, 4)))
    print()


class CSMA(threading.Thread):
    def __init__(self, lock, index):
        super().__init__()
        self.lock = lock
```

```
        self.index = index

    def run(self):
        global numFrames
        global totalFrames

        cnt = 1
        while cnt <= numFrames:
            print("--------------------------------------------------")
            print(f"Attempting to send frame {cnt} of node {self.index}")
            print("--------------------------------------------------")
            print()

            while self.lock.locked():
                backOffTime = random.randint(2, 5)
                print("--------------------------------------------------")
                print(f"Node {self.index} waiting for time : {backOffTime}")
                print("--------------------------------------------------")
                print()
                time.sleep(backOffTime)

            self.lock.acquire()
            time.sleep(frameTime)
            print("--------------------------------------------------")
            print(f"Successfully sent frame {cnt} of node {self.index}")
            print("--------------------------------------------------")
            print()
            self.lock.release()
            totalFrames += 1
            time.sleep(interFrameTime)

            cnt += 1
        return


if __name__ == '__main__':
    numberNodes = int(input("Enter number of nodes: "))
    lock = threading.Lock()
    met = threading.Thread(target=metrics, args=[lock, numberNodes * numFrames])

    Nodes = [CSMA(lock, i + 1) for i in range(0, numberNodes)]
    met.start()
    for node in Nodes:
        node.start()

    for node in Nodes:
        node.join()
    met.join()
```

P-Persistent:

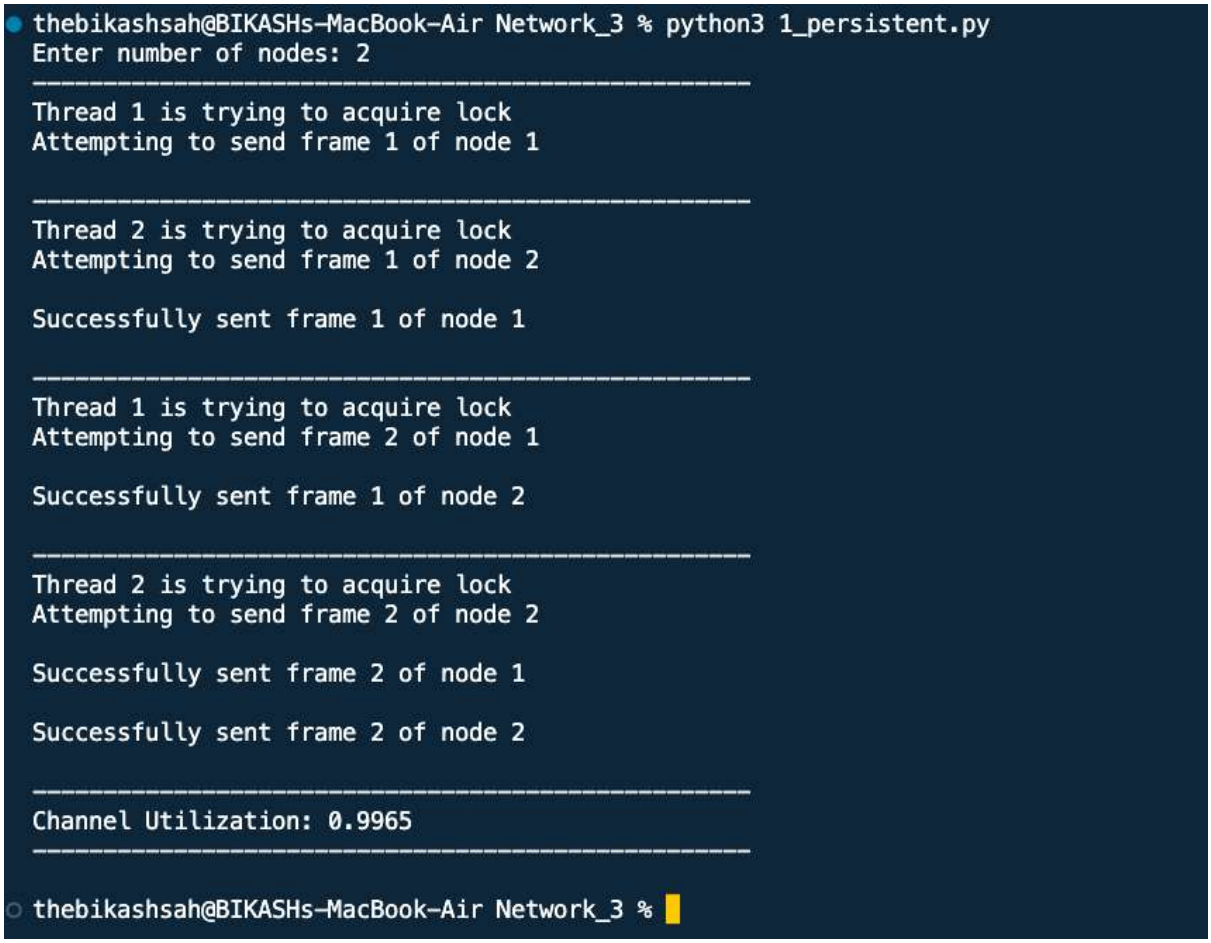
```
import threading
import random
import time
```

```
frameTime = 3
interFrameTime = 1
backOffTime = 2
numFrames = 2
totalFrames = 0


def metrics(lock, total):
    global totalFrames
    tot = 0
    used = 0
    while totalFrames < total:
        if lock.locked():
            used += 1
        tot += 1
    print("--------------------------------------------------")
    print("Channel Utilization: " + str(round(used / tot, 4)))



class CSMA(threading.Thread):
    def __init__(self, lock, k, index, prob):
        super().__init__()
        self.lock = lock
        self.k = k
        self.index = index
        self.prob = prob

    def run(self):
        global numFrames
        global totalFrames

        cnt = 1
        while cnt <= numFrames:
            print("--------------------------------------------------")
            print(f"Attempting to send frame {cnt} of node {self.index}")
            print("--------------------------------------------------")

            while self.lock.locked():
                pass

            decision = random.randint(0, 1);
            while decision > self.prob:
                print("--------------------------------------------------")
                print(f"Node {self.index} backing off")
                print("--------------------------------------------------")


                time.sleep(backOffTime)

                while self.lock.locked():
                    pass

                decision = random.randint(0, 1);

            self.lock.acquire()
            time.sleep(frameTime)
            print(f"Successfully sent frame {cnt} of node {self.index}")
```

```
            self.lock.release()
            totalFrames += 1
            time.sleep(interFrameTime)

            cnt += 1
        return


if __name__ == '__main__':
    numberNodes = int(input("Enter number of nodes: "))
    lock = threading.Lock()
    met = threading.Thread(target=metrics, args=[lock, numberNodes * numFrames])

    Nodes = [CSMA(lock, 4, i + 1, 1 / numberNodes) for i in range(0, numberNodes)]
    met.start()
    for node in Nodes:
        node.start()

    for node in Nodes:
        node.join()
    met.join()
```

Test Cases

1-persistent:



Non-persistent:

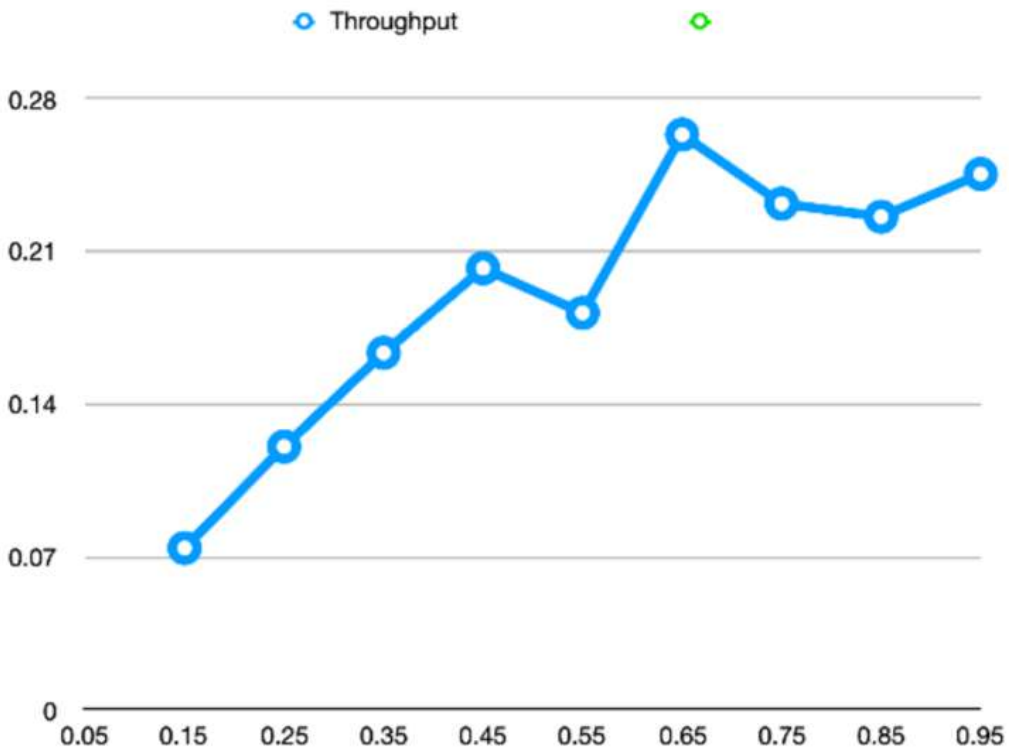
```
thebikashsah@BIKASHs-MacBook-Air Network_3 % python3 non_persistent.py
Enter number of nodes: 2
--------------------------------------------------
Attempting to send frame 1 of node 1
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 1 of node 2
--------------------------------------------------
--------------------------------------------------
Node 2 waiting for time : 4
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 1 of node 1
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 2 of node 1
--------------------------------------------------
--------------------------------------------------
Node 2 waiting for time : 4
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 2 of node 1
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 1 of node 2
--------------------------------------------------
--------------------------------------------------
Attempting to send frame 2 of node 2
--------------------------------------------------
--------------------------------------------------
Successfully sent frame 2 of node 2
--------------------------------------------------
--------------------------------------------------
Channel Utilization: 0.7474
thebikashsah@BIKASHs-MacBook-Air Network_3 %
```

P-persistent:

```
thebikashsah@BIKASHs-MacBook-Air Network_3 % python3 p_persistent.py
Enter number of nodes: 2
------------------------------------------------------
Attempting to send frame 1 of node 1
------------------------------------------------------
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Attempting to send frame 1 of node 2
------------------------------------------------------
Successfully sent frame 1 of node 2
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Attempting to send frame 2 of node 2
------------------------------------------------------
------------------------------------------------------
Node 2 backing off
------------------------------------------------------
------------------------------------------------------
Node 1 backing off
------------------------------------------------------
------------------------------------------------------
Node 2 backing off
------------------------------------------------------
Successfully sent frame 1 of node 1
------------------------------------------------------
Attempting to send frame 2 of node 1
------------------------------------------------------
Successfully sent frame 2 of node 2
Successfully sent frame 2 of node 1
------------------------------------------------------
Channel Utilization: 0.6476
thebikashsah@BIKASHs-MacBook-Air Network_3 %
```

Results

Comparing the throughput by varing p of the p-persistent CSMA approach. The p is varied according to the X-axis and the throughput is along the Y-axis. The number of nodes considered for this simulation is 12.



Discussion:

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot, I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.

Computer Networks Lab Report - Assignment 4

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 4

Problem Statement:

Assignment 4: Implement CDMA with Walsh code.

Submission due: 19th-23th September 2022

In this assignment you have to implement CDMA for multiple access of a common channel by n stations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations.

Submission Date: **21 November, 2022**

Deadline: 23rd Sept, 2022

Introduction

There are n number of servers. Each server has a data bit to be sent to the reciever. The data bits are sent through a channel. The channel is lossy. The channel can drop the data bits. The channel can also add noise to the data bits. The reciever has to decode the data bits from the servers. The reciever has to detect the dropped data bits and the noise added to the data bits. The reciever has to correct the data bits. The reciever has to display the data bits from the servers.

Design

```
The program should have the following classes:
1. sender
2. Channel
3. receiver

The sender class should have the following functions:

1. sender(int n, vector<vector<int> > arr) - constructor to initialize the number of s
ervers and the walsh table for the servers.

2. vector<int> getData() - to get the data bits from the servers. The data bits are ra
ndomly generated.

3. vector<int> encodeData() - to encode the data bits from the servers using the walsh
table. The encoded data bits are sent to the channel.

The channel class should have the following functions:
1. Channel(sender *s) - constructor to initialize the sender object.

2. vector<int> getFromSender() - to get the encoded data bits from the sender.

3. vector<int> sendToreciver() - to send the encoded data bits to the receiver.

The receiver class should have the following functions:
1. receiver(int n, vector<vector<int> > arr, Channel *ch) - constructor to initialize
 the number of servers, the walsh table for the servers and the channel object.

2. vector<int> reciveData() - to recieve the encoded data bits from the channel.

3. vector<int> decodeData() - to decode the encoded data bits using the walsh table. T
he decoded data bits are sent to the display function.

 The main function should have the following:
 1. The main function should create the sender, channel and receiver objects.
 2. The main function should call the decodeData() function of the receiver object.
 3. The main function should display the data bits from the servers.
```

Implementation

```
#include <bits/stdc++.h>
using namespace std;

class sender
{
    int senders;
    int size;
    int startIndex;
    vector<vector<int> > walshTable;
    vector<vector<int> > data;

public:
    sender(int n, vector<vector<int> > arr)
    {
        this->startIndex = 0;
        this->senders = n;
        this->walshTable = arr;
        this->size = rand() % 20;

        //GENERATING RANDOM DATA FOR EVERY SENDER
        cout<<endl;
        cout<<"____Data table_______"<<endl;
        for (int i = 0; i < senders; i++)
        {
            cout<<"Data for sender "<<i<<":   ";
            vector<int> temp;
            for (int j = 0; j < size; j++)
            {
                int bit = rand() % 3;
                temp.push_back(bit);
                if(bit==2)
                {
                    cout << "*" <<"  ";
                }
                else
                {cout << bit <<"  ";}
            }
            cout << endl;
            data.push_back(temp);
        }
        cout<<"_________________"<<endl;
    }

    vector<int> getData()
    {
        if (startIndex == size)
        {
            cout << "no more data";
        }
        vector<int> res;
        for (int i = 0; i < senders; i++)
        {
            res.push_back(data[i][startIndex]);
        }
        startIndex++;
        cout << endl
```

```
             << endl
             << endl;
        cout<<"_____sender side______"<<endl;
        for (int i = 0; i < res.size(); i++)
        {
            cout <<"Data bit from Sender "<<i<<":  ";
            if(res[i]==2)
            {
                cout<<"*"<<endl;
            }
            else
             {cout<<res[i]<<endl;}
        }
        cout<<"_________________"<<endl;
        return res;
    }
    vector<int> encodeData()
    {
        vector<int> res(walshTable.size(), 0);
        vector<int> rawData = getData();
        for (int i = 0; i < rawData.size(); i++)
        {
            if (rawData[i] == 0)
                for (int k = 0; k < res.size(); k++)
                {

                    res[k] += (-1 * walshTable[i][k]);
                }
            else if (rawData[i] == 1)
                for (int k = 0; k < res.size(); k++)
                {

                    res[k] += (1 * walshTable[i][k]);
                }
            else
                for (int k = 0; k < res.size(); k++)
                {

                    res[k] += (0 * walshTable[i][k]);
                }
        }
        cout << endl;
        cout<<"____DATA SENT______"<<endl;
        for (int i = 0; i < res.size(); i++)
        {
            cout << res[i] << "    ";
        }
        cout<<endl;
        cout<<"_____________"<<endl;
        cout<<endl;
        return res;
    }
};

class Channel
{
public:
    sender *st;
```

```
    Channel(sender *s)
    {
        st = s;
    }
    vector<int> getFromSender()
    {
        return st->encodeData();
    }
    vector<int> sendToreciver()
    {
        return this->getFromSender();
    }
};

class reciver
{
    int stations;
    vector<vector<int> > walshTable;
    Channel *ch;

public:
    reciver(int s, vector<vector<int> > arr, Channel *ch)
    {
        this->stations = s;
        this->walshTable = arr;
        this->ch = ch;
    }
    vector<int> reciveData()
    {
        return ch->sendToreciver();
    }
    vector<int> decodeData()
    {
        vector<int> data = reciveData();
        return data;
    }
};
vector<vector<int> > fun(int n)
{
    if (n == 1)
    {
        vector<vector<int> > a(1, vector<int>(1, 1));
        return a;
    }

    vector<vector<int> > arr(n, vector<int>(n));
    vector<vector<int> > temp = fun(n / 2);
    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            arr[i][j] = temp[i][j];
            arr[i + (n / 2)][j] = temp[i][j];
            arr[i][j + (n / 2)] = temp[i][j];
            arr[i + n / 2][j + (n / 2)] = -temp[i][j];
        }
    }
    return arr;
```

```
}

int main()
{
    srand(time(0));

    int n;
    cout<<"enter the number of servers: ";
    cin >> n;
    int ser = n;
    n = pow(2, ceil(log2(n)));//maximum size of walash table required

    vector<vector<int> > arr = fun(n);
    cout<<"_____WALASH TABLE_______"<<endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if(arr[i][j]==1)
            {
                cout<<" "<<arr[i][j]<<"   ";
            }
            else{
            cout << arr[i][j] << "   ";}
        }
        cout << endl;
    }
    cout<<"___________________"<<endl;

    sender st(ser, arr);
    Channel ch(&st);
    reciver rec(ser, arr, &ch);
    int flag=1;
    while (flag == 1)
    {
        vector<int> temp = rec.decodeData();
        cout<<"_____RECIVER SIDE__________"<<endl;
        for(int i=0;i<ser;i++)
        {
            int sum=0;
            for(int k=0;k<arr.size();k++)
            {
                sum+=(temp[k]*arr[i][k]);
            }
            int bit = sum / n;
            if (bit == 0)
            {
                   cout << "Data bit from server " << i << ": "<< " *"<<endl;
            }
            else if (bit == 1)
            {
                {
                    cout << "Data bit from server " << i << ": "<< " 1"<<endl;
                }
            }
            else
            {
                {
```

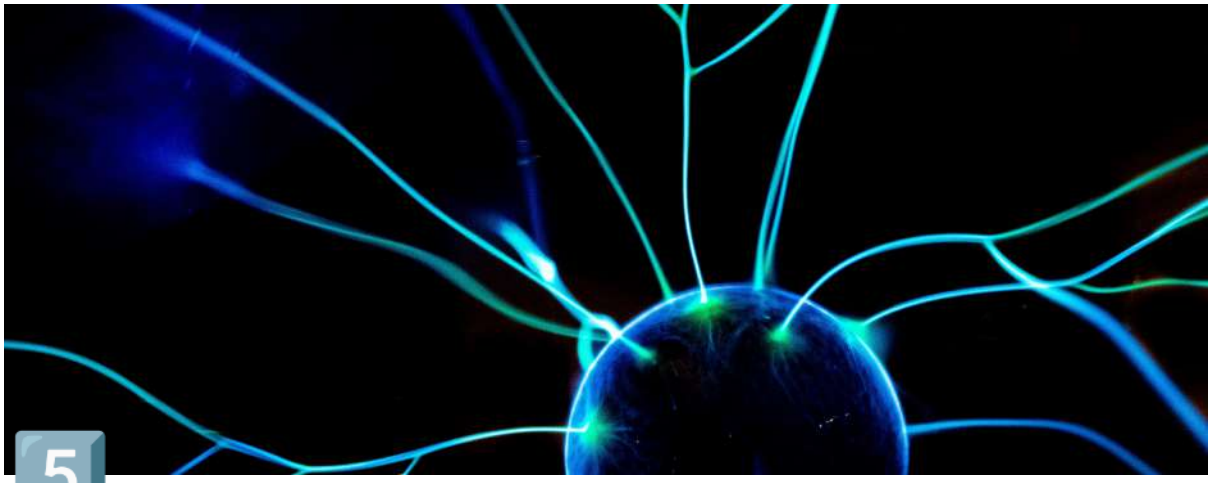
```
                    cout << "Data bit from server " << i << " "<< "0"<<endl;
                }
            }
        }
        cout<<"_____________________"<<endl;
        cout << "enter y to continue" << endl;
        cin >> flag;
    }
}
```

Test Cases and Results

```
thebikashsah@BIKASHs-MacBook-Air Network_4 % ./a.out
enter the number of servers: 2
_____WALASH TABLE_______
 1    1
 1   -1
_________________

____Data table_______
Data for sender 0:   0  1  *  *  0  0  1  *  1  1  *  0  *  1  0  *  *  *
Data for sender 1:   0  0  1  1  0  *  0  0  *  0  0  1  *  1  1  *  0  0
_________________



_____sender side______
Data bit from Sender 0:  0
Data bit from Sender 1:  0
_______________

____DATA SENT______
-2    0
_____________

_____RECIVER SIDE__________
Data bit from server 0 0
Data bit from server 1 0
____________________
enter y to continue
```

Discussion

This assignment has helped me to understand the how Walsh Table is built for a given number of stations, and how CDMA channelization protocol encodes and decodes the data bits sent by all stations simultaneously.



Bikash_Sah_002010501018_Computer Networks Lab Report_5

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 5

Problem Statement:

Assignment 5: Packet tracer and traffic analysis with Wireshark.

Submission due: 10th-14th October 2022

Questions (Please take screenshots and explain)

1. **Generate some ICMP traffic by using the Ping command line tool to check the connectivity of a neighbouring machine (or router). Note the results in Wireshark. The initial ARP request broadcast from your PC determines the physical MAC address of the network IP Address, and the ARP reply from the neighboring system. After the ARP request, the pings (ICMP echo request and replies) can be seen.**
2. **Generate some web traffic and**
    a. **find the list the different protocols that appear in the protocol column in the unfiltered packet-listing window of Wireshark.**
    b. **How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.)**
    c. **What is the Internet address of the website? What is the Internet address of your computer?**
    d. **Search back through your capture, and find an HTTP packet containing a GET command. Click on the packet in the Packet List Panel. Then expand the HTTP layer in the Packet Details Panel, from the packet.**
    e. **Find out the value of the Host from the Packet Details Panel, within the GET command.**
3. **Highlight the Hex and ASCII representations of the packet in the Packet Bytes Panel.**

---

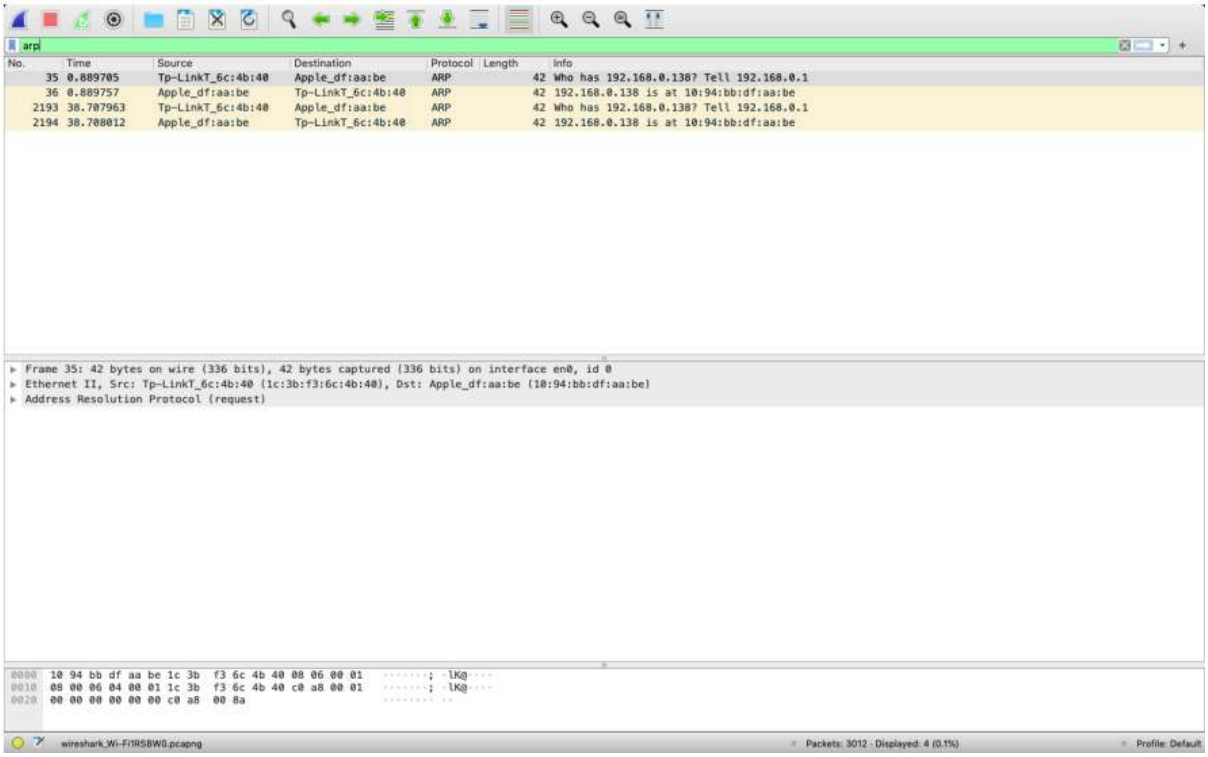
4. **Find out the first 4 bytes of the Hex value of the Host parameter from the Packet Bytes Panel.**
5. **Filter packets with http, TCP, DNS and other protocols.**
    a. **Find out what are those packets contain by following one of the conversations (also called network flows), select one of the packets and press the right mouse button..click on follow.**
6. **Search through your capture, and find an HTTP packet coming back from the server (TCP Source Port == 80). Expand the Ethernet layer in the Packet Details Panel.**
7. **What are the manufacturers of your PC's Network Interface Card (NIC), and the servers NIC?**
8. **What are the Hex values (shown the raw bytes panel) of the two NICS Manufacturers OUIs?**
9. **Find the following statistics:**
    a. **What percentage of packets in your capture are TCP, and give an example of the higher level protocol which uses TCP?**
    b. **What percentage of packets in your capture are UDP, and give an example of the higher level protocol which uses UDP?**
10. **Find the traffic flow Select the Statistics->Flow Graph menu option. Choose General Flow and Network Source options, and click the OK button.**

Submission Date: **21 November, 2022**

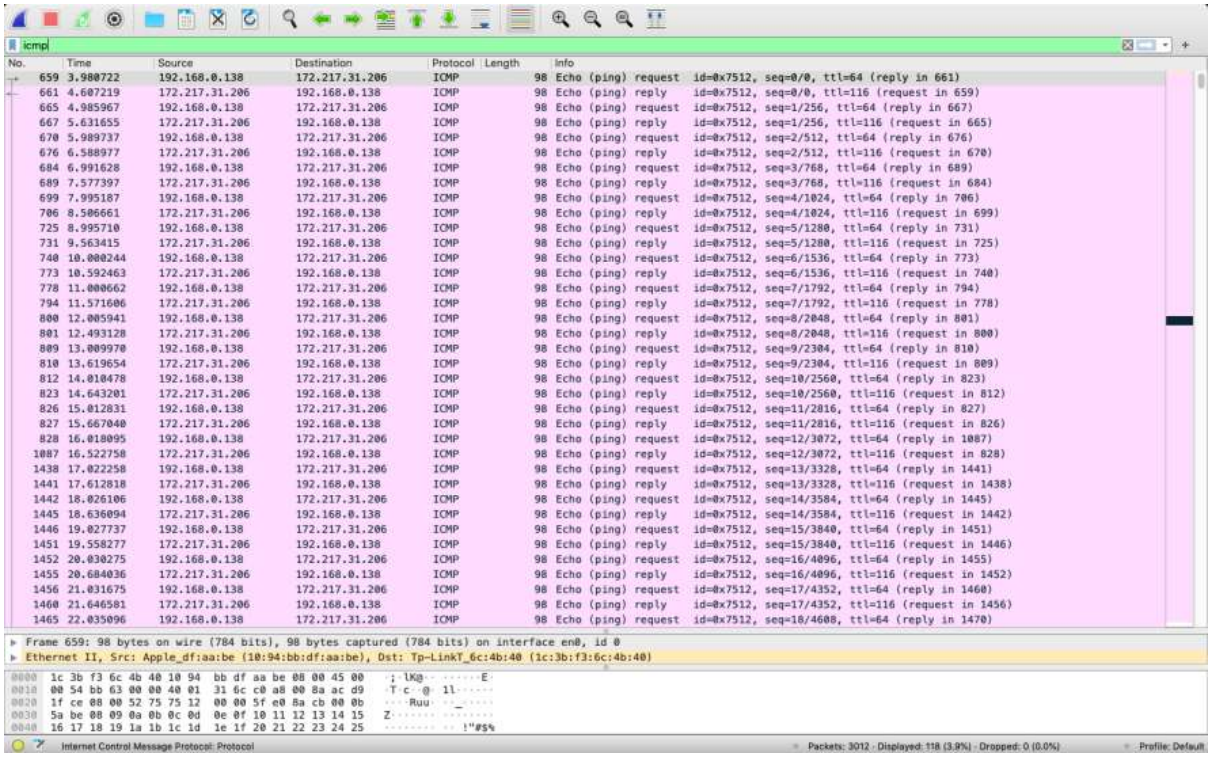
Deadline: 14th October, 2022

Q1

1. **Generate some ICMP traffic by using the Ping command line tool to check the connectivity of a neighbouring machine (or router). Note the results in Wireshark. The initial ARP request broadcast from your PC determines the physical MAC address of the network IP Address, and the ARP reply from the neighboring system. After the ARP request, the pings (ICMP echo request and replies) can be seen.**



The different ARP requests generated as a part of locating MAC addresses of neighbouring machines.Tp-LinkT_6c is the Wifi-adaptor and the origin machine while Apple_df is the destination (Macbook Air).



The different ICMP request-replies generated as a part of pinging google.com

---

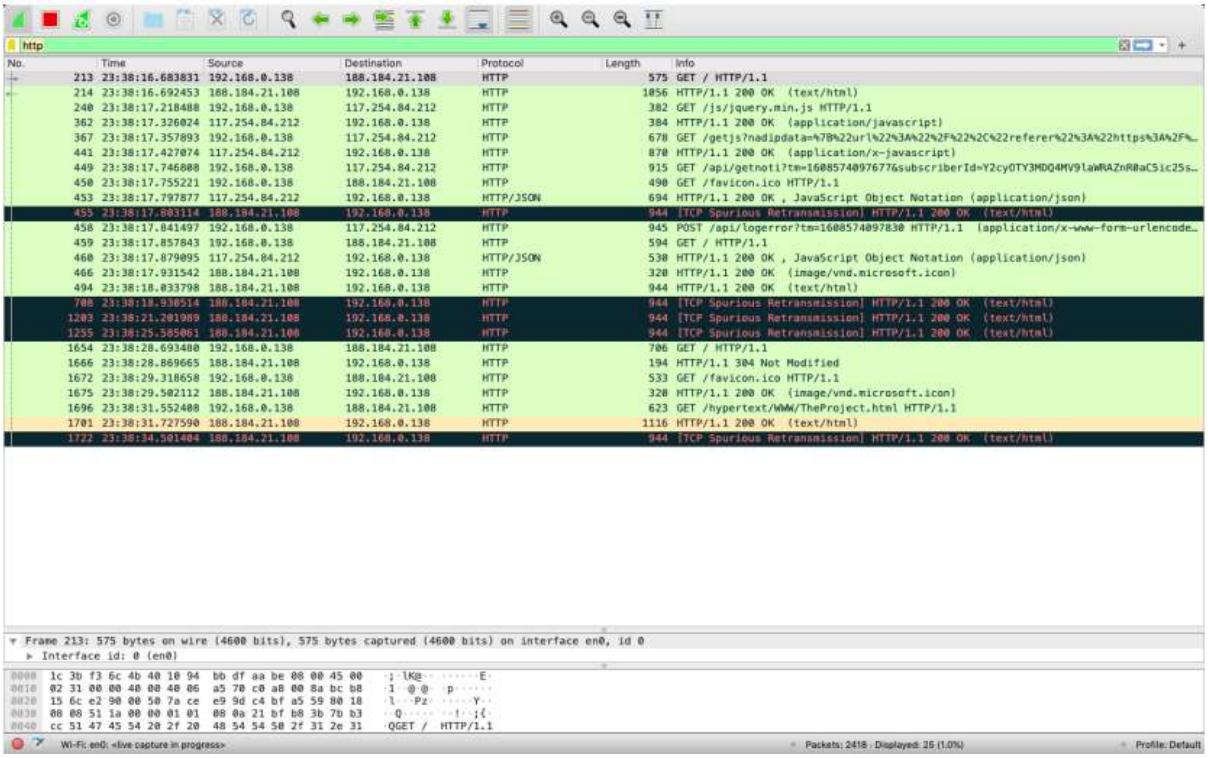
Q2

a. find the list the different protocols that appear in the protocol column in the unfiltered packet-listing window of Wireshark.

1. ARP
2. DCP-AF
3. DNS
4. DHCP
5. ICMP
6. IGMPv2
7. MDNS
8. TCP
9. UDP
10. TLSv1.3
11. TLSv1.2

---

b. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.)



The first HTTP GET request was sent at 23:38:16.218488 and the HTTP OK response was received at 23:38:17.326034. and thus it took 1.11 seconds.

---

c. What is the Internet address of the website? What is the Internet address of your computer?

The internet address of the website http://info.cern.ch/(had to look up an ancient website) is 188.184.21.108 and the internet address of my computer is 192.168.0.138.

---

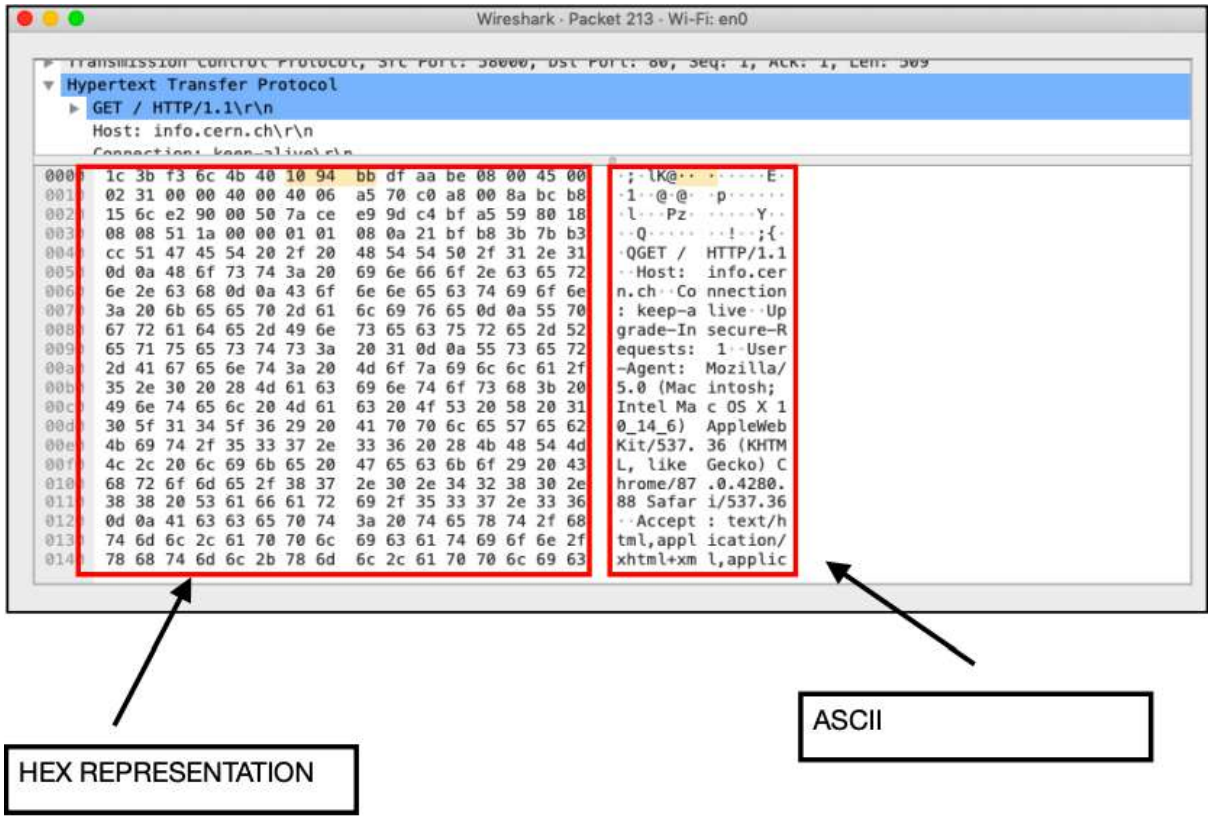
d. Search back through your capture, and find an HTTP packet containing a GET command. Click on the packet in the Packet List Panel. Then expand the HTTP layer in the Packet Details Panel, from the packet.

e. Find out the value of the Host from the Packet Details Panel, within the GET command.

The figure 2d shows the hostname.- http://info.cern.ch/

Q3

3. Highlight the Hex and ASCII representations of the packet in the Packet Bytes Panel.



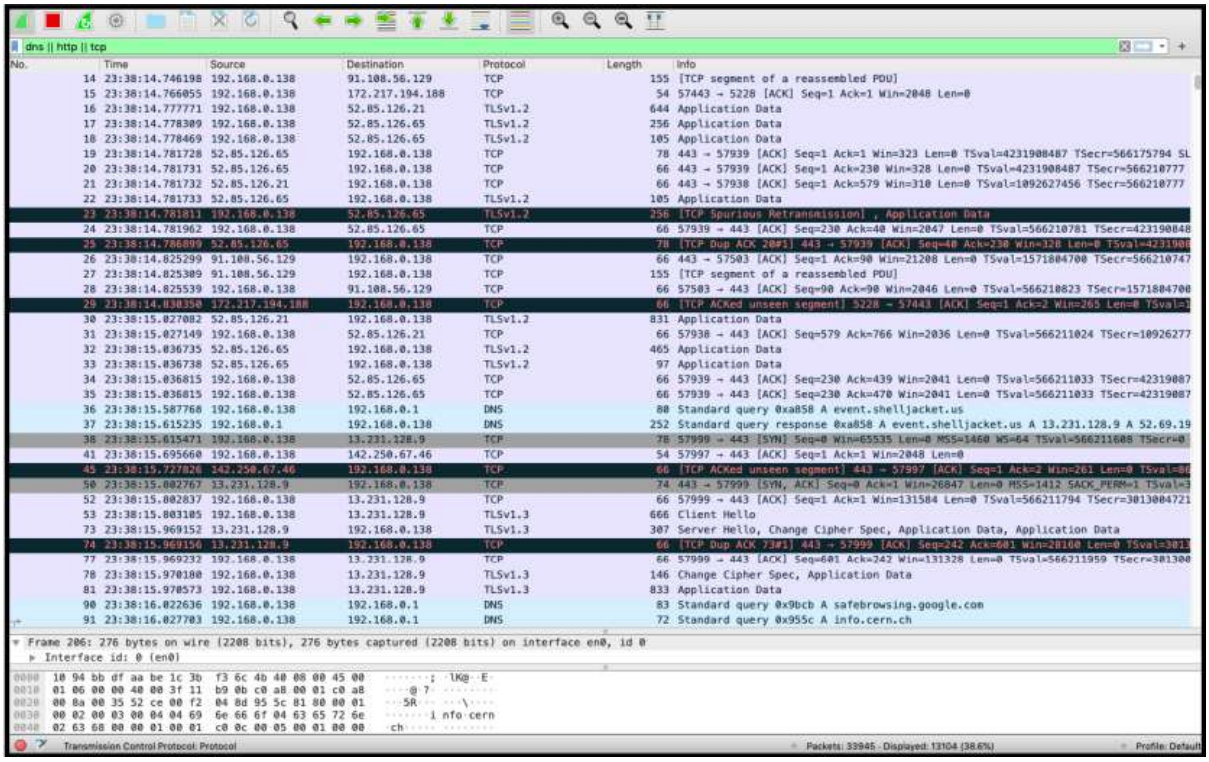
Q4

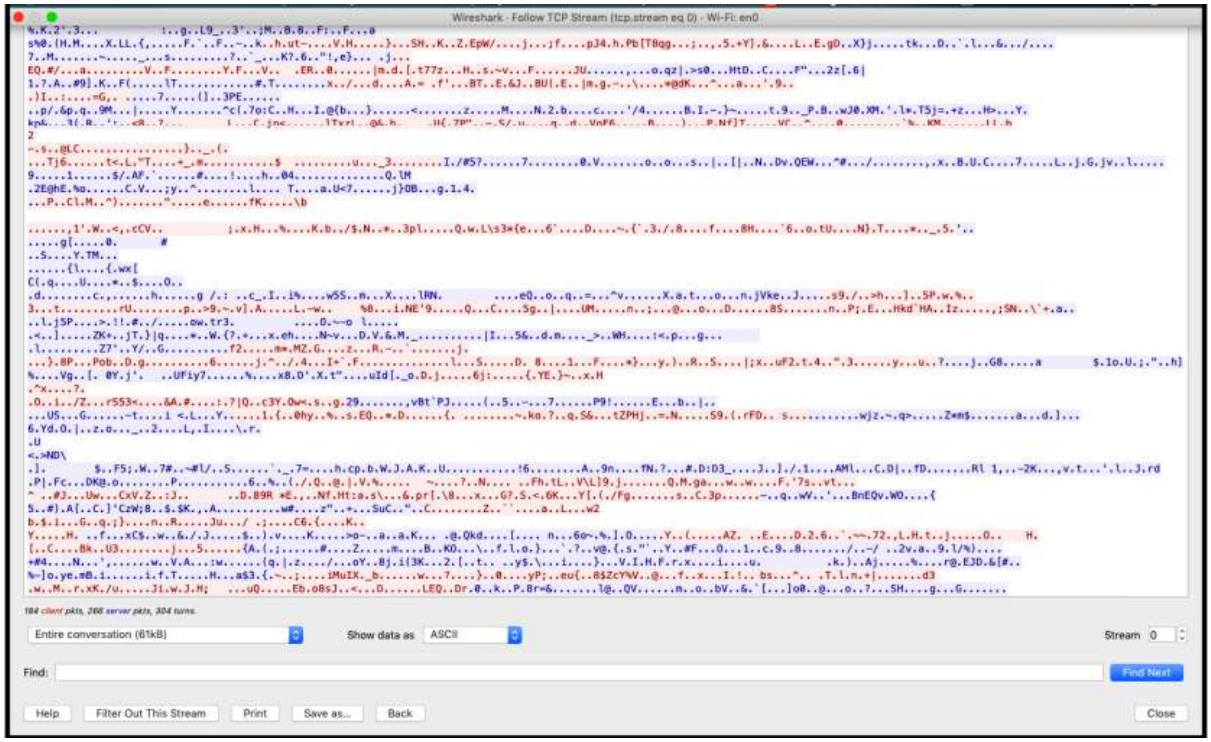
4. **Find out the first 4 bytes of the Hex value of the Host parameter from the Packet Bytes Panel.**

```
1c 3b f3 6c 4b 40 10 94  bb df aa be 08 00 45 00   ·;·lK@·· ·····E·
02 31 00 00 40 00 40 06  a5 70 c0 a8 00 8a bc b8   ·1··@·@· ·p·····
15 6c e2 90 00 50 7a ce  e9 9d c4 bf a5 59 80 18   ·l···Pz· ·····Y··
08 08 51 1a 00 00 01 01  08 0a 21 bf b8 3b 7b b3   ··Q····· ··!··;{·
cc 51 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ·QGET /  HTTP/1.1
0d 0a 48 6f 73 74 3a 20  69 6e 66 6f 2e 63 65 72   ··Host:  info.cer
6e 2e 63 68 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e   n.ch··Co nnection
3a 20 6b 65 65 70 2d 61  6c 69 76 65 0d 0a 55 70   : keep-a live··Up
67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52   grade-In secure-R
65 71 75 65 73 74 73 3a  20 31 0d 0a 55 73 65 72   equests:  1··User
2d 41 67 65 6e 74 3a 20  4d 6f 7a 69 6c 6c 61 2f   -Agent:  Mozilla/
35 2e 30 20 28 4d 61 63  69 6e 74 6f 73 68 3b 20   5.0 (Mac intosh;
49 6e 74 65 6c 20 4d 61  63 20 4f 53 20 58 20 31   Intel Ma c OS X 1
```

Q5

5. **Filter packets with http, TCP, DNS and other protocols.**
    a. **Find out what are those packets contain by following one of the conversations (also called network flows), select one of the packets and press the right mouse button..click on follow.**

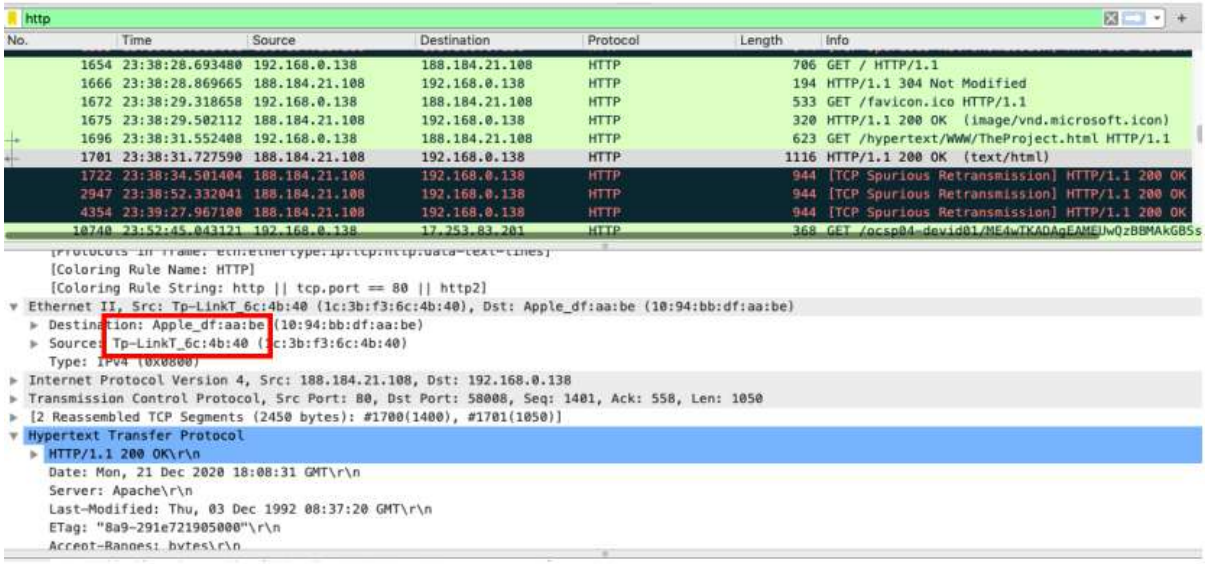




Q6

6. Search through your capture, and find an HTTP packet coming back from the server (TCP Source Port == 80). Expand the Ethernet layer in the Packet Details Panel.

fig6. Showing the ethernet details of a HTTP OK request.



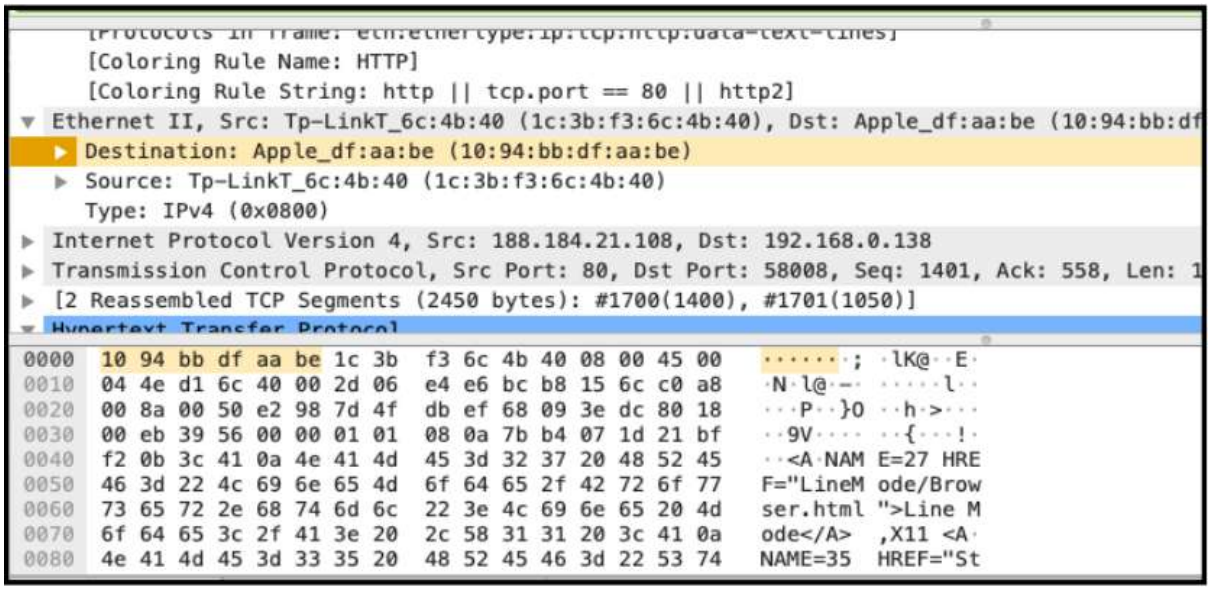
Q7

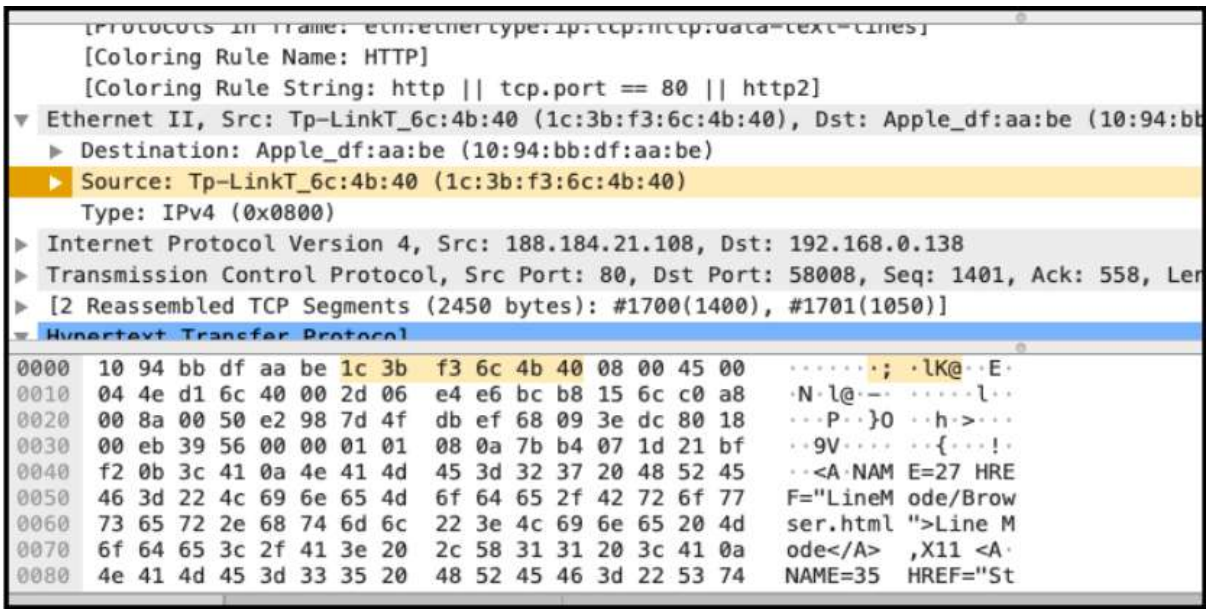
7. **What are the manufacturers of your PC's Network Interface Card (NIC), and the servers NIC?**

PC's NIC : Apple_df
Servers's NIC : Tp-LinkT_6c

Q8

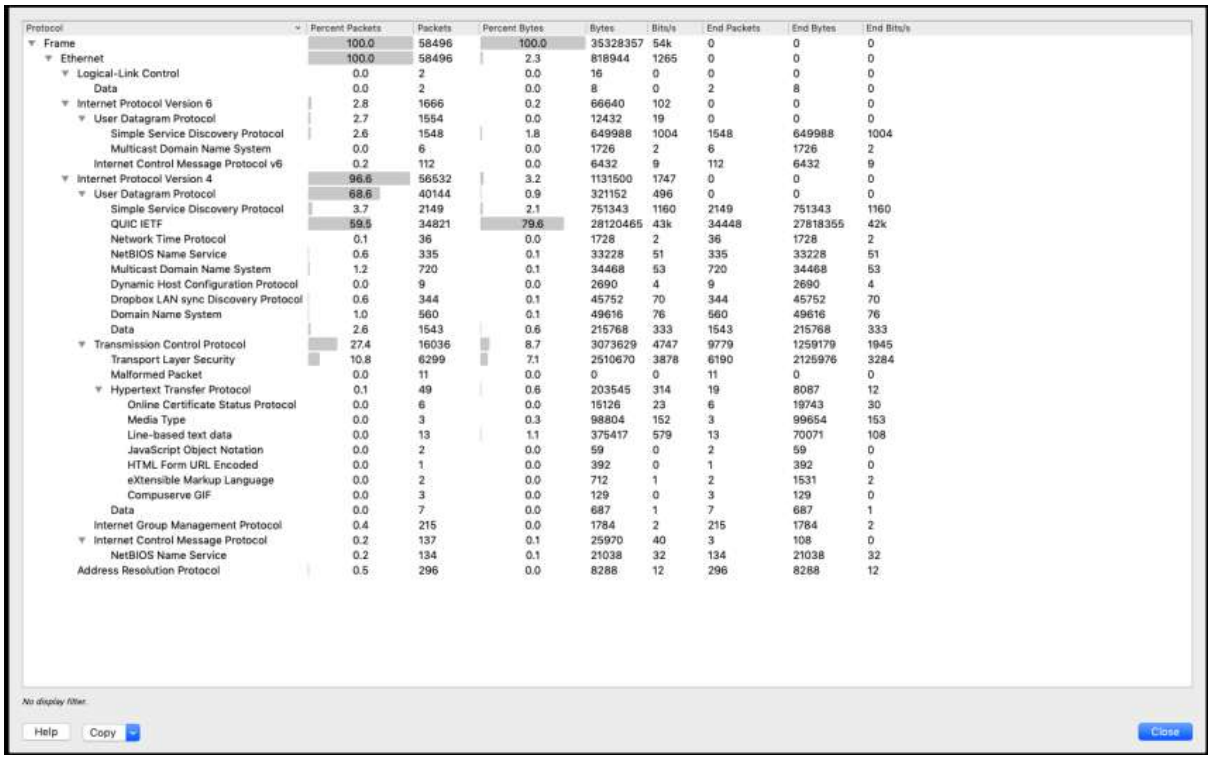
8. **What are the Hex values (shown the raw bytes panel) of the two NICS Manufacturers OUIs?**





Q9

9. **Find the following statistics:**
    a. **What percentage of packets in your capture are TCP, and give an example of the higher level protocol which uses TCP?**
    b. **What percentage of packets in your capture are UDP, and give an example of the higher level protocol which uses UDP?**



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	58496	100.0	35328357	54k	0	0	0
Ethernet	100.0	58496	2.3	818944	1265	0	0	0
Logical-Link Control	0.0	2	0.0	16	0	0	0	0
Data	0.0	2	0.0	8	0	2	8	0
Internet Protocol Version 6	2.8	1666	0.2	66640	102	0	0	0
User Datagram Protocol	2.7	1554	0.0	12432	19	0	0	0
Simple Service Discovery Protocol	2.6	1548	1.8	649988	1004	1548	649988	1004
Multicast Domain Name System	0.0	6	0.0	1726	2	6	1726	2
Internet Control Message Protocol v6	0.2	112	0.0	6432	9	112	6432	9
Internet Protocol Version 4	96.6	56532	3.2	1131500	1747	0	0	0
User Datagram Protocol	68.6	40144	0.9	321152	496	0	0	0
Simple Service Discovery Protocol	3.7	2149	2.1	751343	1160	2149	751343	1160
QUIC IETF	59.5	34821	79.6	28120465	43k	34448	27818355	42k
Network Time Protocol	0.1	36	0.0	1728	2	36	1728	2
NetBIOS Name Service	0.6	335	0.1	33228	51	335	33228	51
Multicast Domain Name System	1.2	720	0.1	34468	53	720	34468	53
Dynamic Host Configuration Protocol	0.0	9	0.0	2690	4	9	2690	4
Dropbox LAN sync Discovery Protocol	0.6	344	0.1	45752	70	344	45752	70
Domain Name System	1.0	560	0.1	49616	76	560	49616	76
Data	2.6	1543	0.6	215768	333	1543	215768	333
Transmission Control Protocol	27.4	16036	8.7	3073629	4747	9779	1259179	1945
Transport Layer Security	10.8	6299	7.1	2510670	3878	6190	2125976	3284
Malformed Packet	0.0	11	0.0	0	0	11	0	0
Hypertext Transfer Protocol	0.1	49	0.6	203545	314	19	8087	12
Online Certificate Status Protocol	0.0	6	0.0	15126	23	6	19743	30
Media Type	0.0	3	0.3	98804	152	3	99654	153
Line-based text data	0.0	13	1.1	375417	579	13	70071	108
JavaScript Object Notation	0.0	2	0.0	59	0	2	59	0
HTML Form URL Encoded	0.0	1	0.0	392	0	1	392	0
eXtensible Markup Language	0.0	2	0.0	712	1	2	1531	2
Compuserve GIF	0.0	3	0.0	129	0	3	129	0
Data	0.0	7	0.0	687	1	7	687	1
Internet Group Management Protocol	0.4	215	0.0	1784	2	215	1784	2
Internet Control Message Protocol	0.2	137	0.1	25970	40	3	108	0
NetBIOS Name Service	0.2	134	0.1	21038	32	134	21038	32
Address Resolution Protocol	0.5	296	0.0	8288	12	296	8288	12

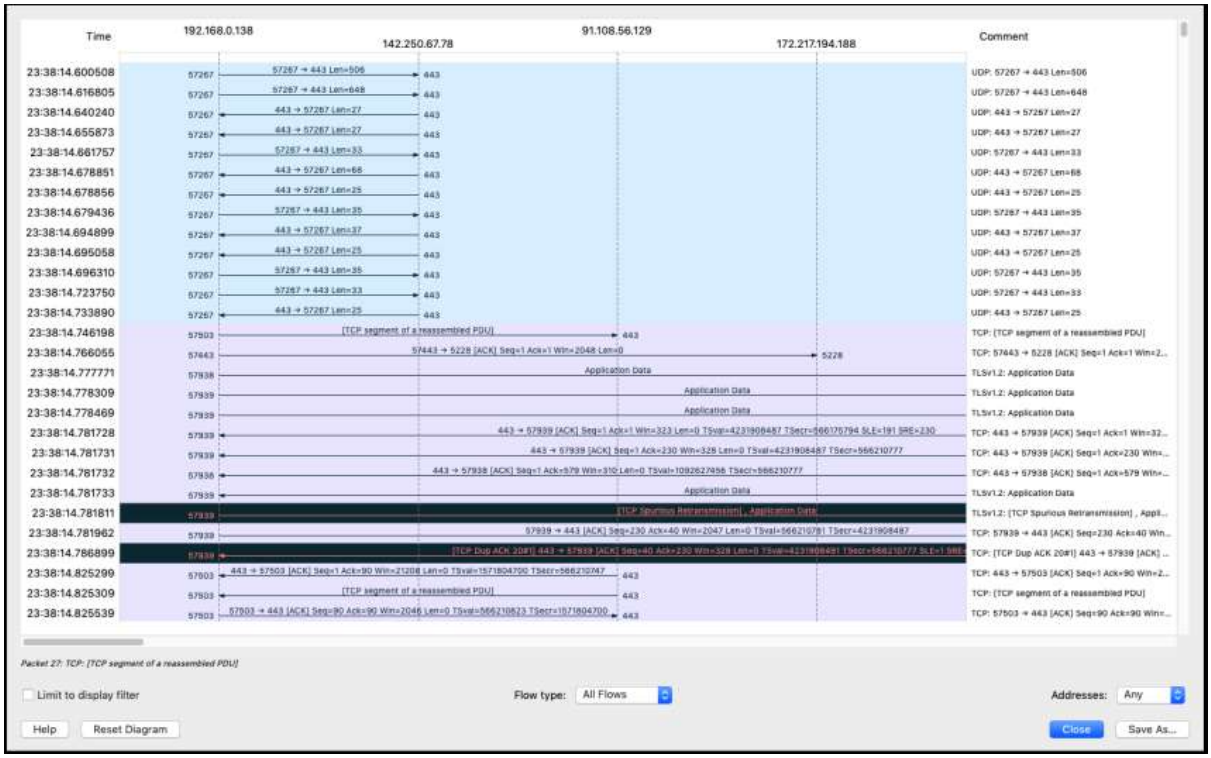
No display filter.

Help Copy Close

a. TCP Packet Percentage : 27.4% . A high level protocol that uses TCP is HTTP.
b. UDP Packet Percentage : 68.6%. A high level protocol that uses UDP is DNS.

Q10

10. **Find the traffic flow Select the Statistics->Flow Graph menu option. Choose General Flow and Network Source options, and click the OK button.**


Time
192.168.0.138
142.250.67.78
91.108.56.129
172.217.194.188
Comment
Packet 27: TCP: [TCP segment of a reassembled PDU]
Limit to display filter
Flow type:
All Flows
Addresses:
Any
Help
Reset Diagram
Close
Save As...

Bikash_Sah_002010501018_Computer Networks Lab Report_6

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 6

Submission Date: **21 November, 2022**

Deadline: 21st October, 2022

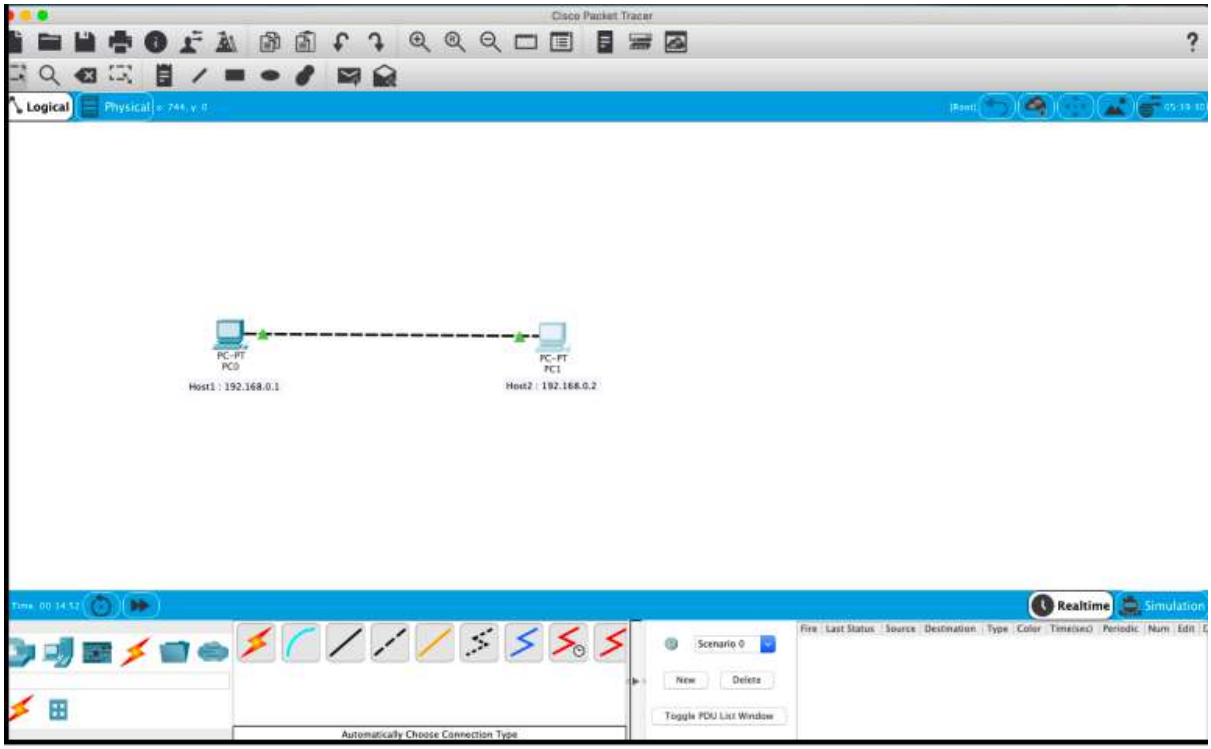
Problem Statement:

Assignment 6: Use Cisco Packet Tracer software to do the following experiments.
Submission due: 17th -21st October 2022

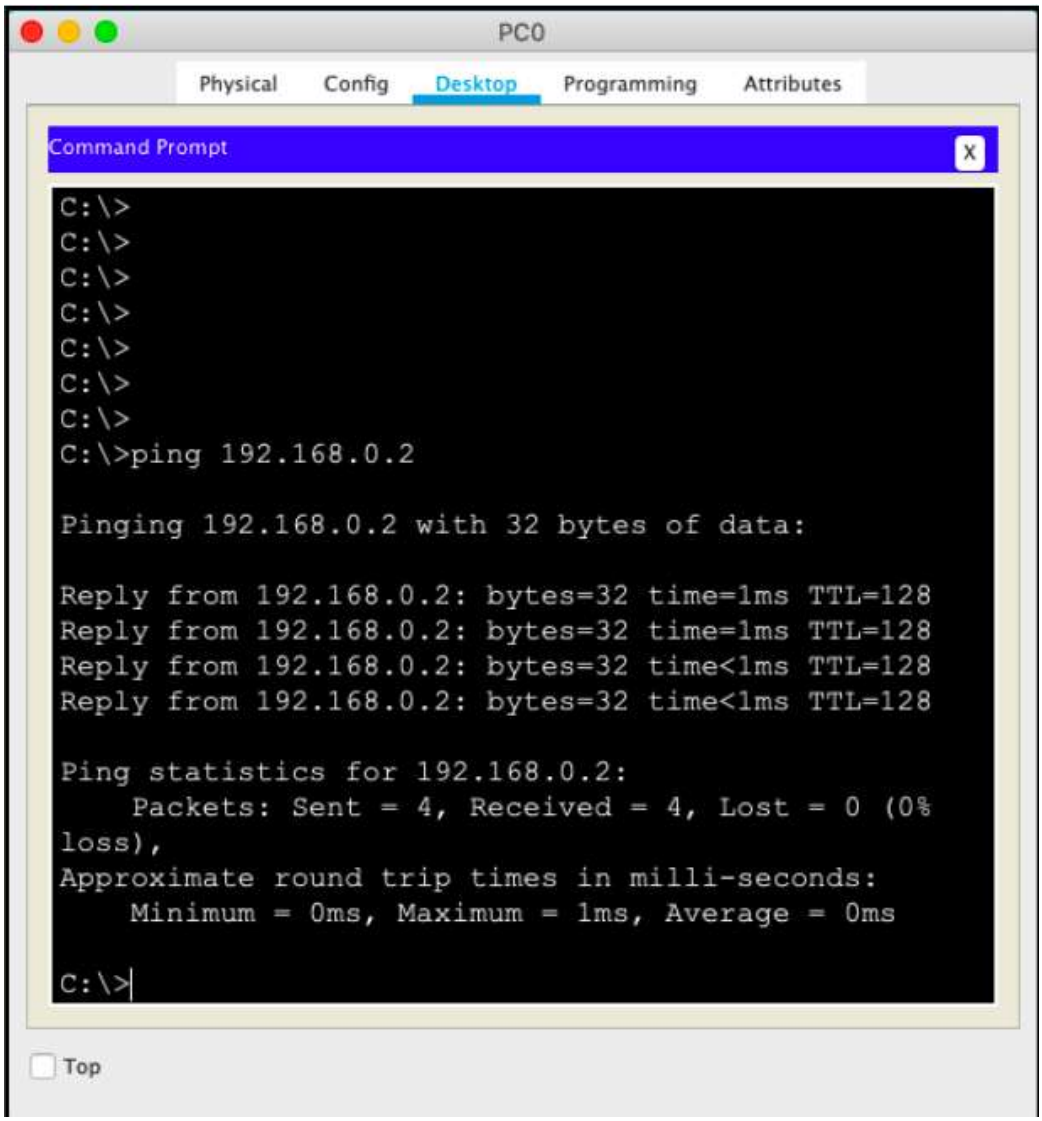
1. Connect two hosts back-to-back with a cross over cable. Assign IP addresses, and see whether they are able to ping each other.
2. Create a LAN (named LAN-A) with 3 hosts using a hub. Ping each pair of nodes.
3. Create a LAN (named LAN-B) with 3 hosts using a switch. Record contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch. Ping each pair of nodes. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.
4. Connect LAN-A and LAN-B by connecting the hub and switch using a cross-over cable. Ping between each pair of hosts of LAN-A and LAN-B. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.
5. Create a LAN (named JU-Main) with three hosts connected via a layer-2 switch (Cisco 2950 switch PC-LAB1-Switch). Connect the switch to a router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.148.0/24. Configure default gateway of each hosts as the IP address of the interface of the router which is connected to the LAN. Create another LAN (named JU-SL) with three hosts connected via a layer-2 switch (Cisco 2950 switch PC-LAB2-Switch). Connect this switch to another router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.149.0/24. Configure default gateway of each hosts as the IP address of the interface of the router which is connected to the LAN. Connect the two routers through appropriate WAN interfaces. Assign IP addresses to the WAN interfaces from network 192.168.150.0/24. Add static route in both of the routers to route packets between two LANs.
6. Add servers to the individual LANs (in problem 5) and configure them as a DHCP server. Configure the hosts in the individual LAN to obtain IP addresses and address of the default gateway via this DHCP server.
7. Create a LAN (CSE) with three hosts connected via a layer-2 switch (Cisco 2950 switch CSE-Switch). Also add a web server and a ftp server to this LAN. The hosts dynamically get their IP addresses from a local DHCP server. Servers are assigned fixed IP addresses. Configure the individual hosts to use the local DNS server for name resolution. Add a Domain Name Server (DNS) to this LAN. Create appropriate records in the DNS server for the individual servers in the LAN. The domain name of the LAN is cse.myuniv.edu. Configure the individual hosts to use the local DNS server for name resolution.

Q1

1. Connect two hosts back-to-back with a cross over cable. Assign IP addresses, and see whether they are able to ping each other.



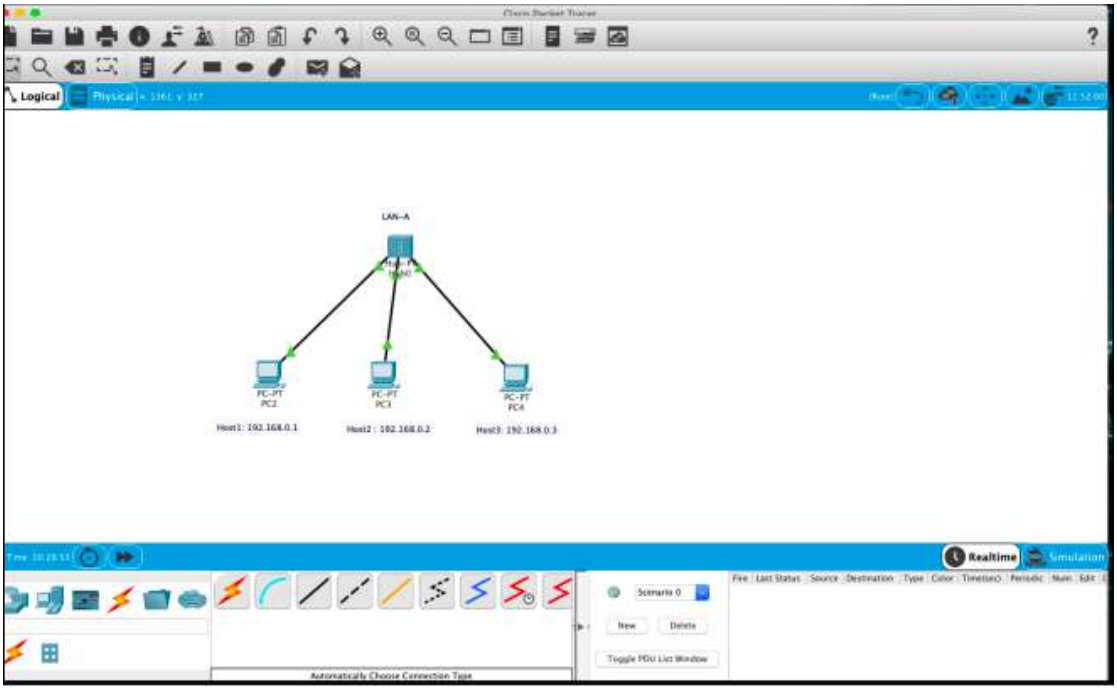
Two hosts connected over a cross-over cable.



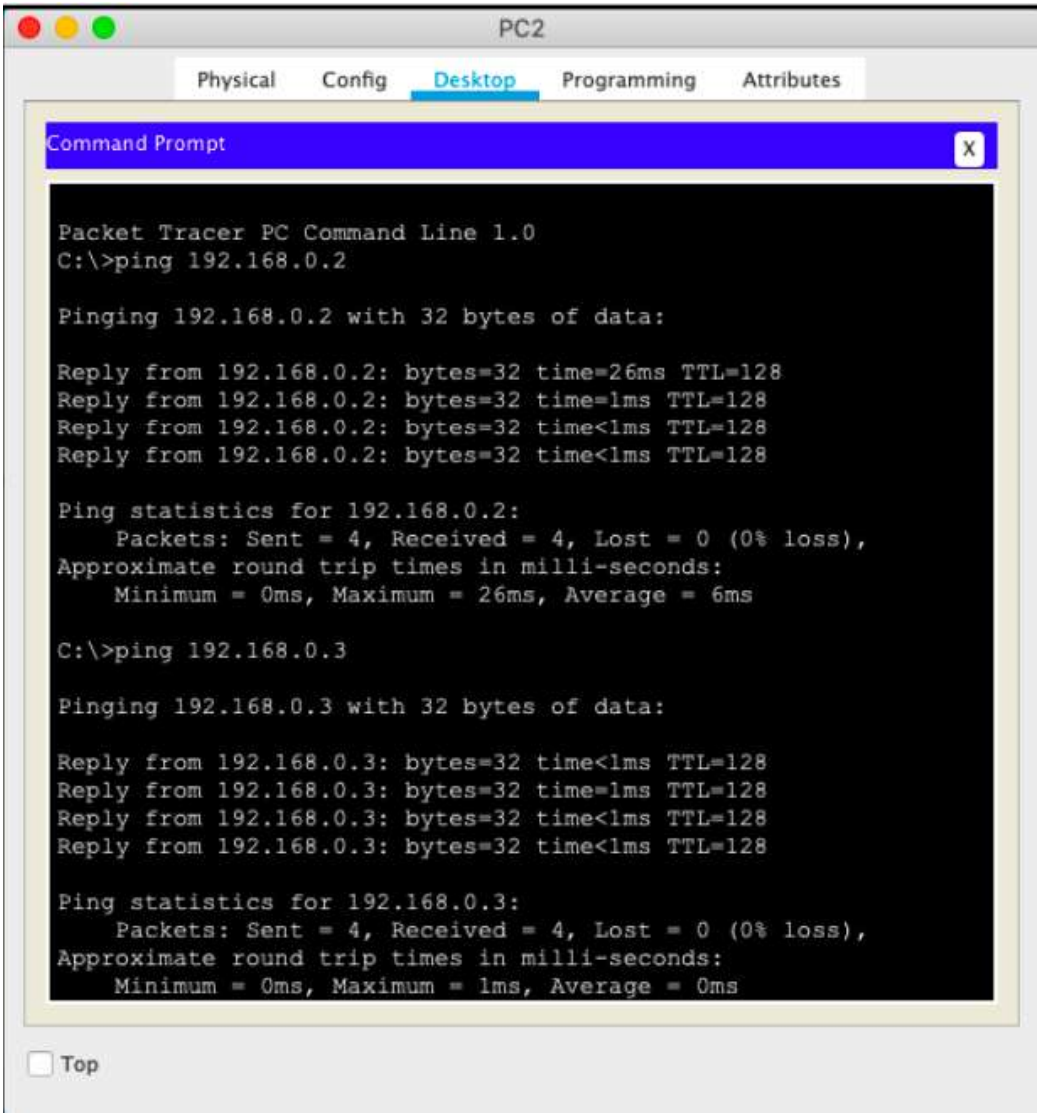
Host 1 with IP: 192.168.0.1 pinging Host 2 with IP : 192.168.0.2.

Q2

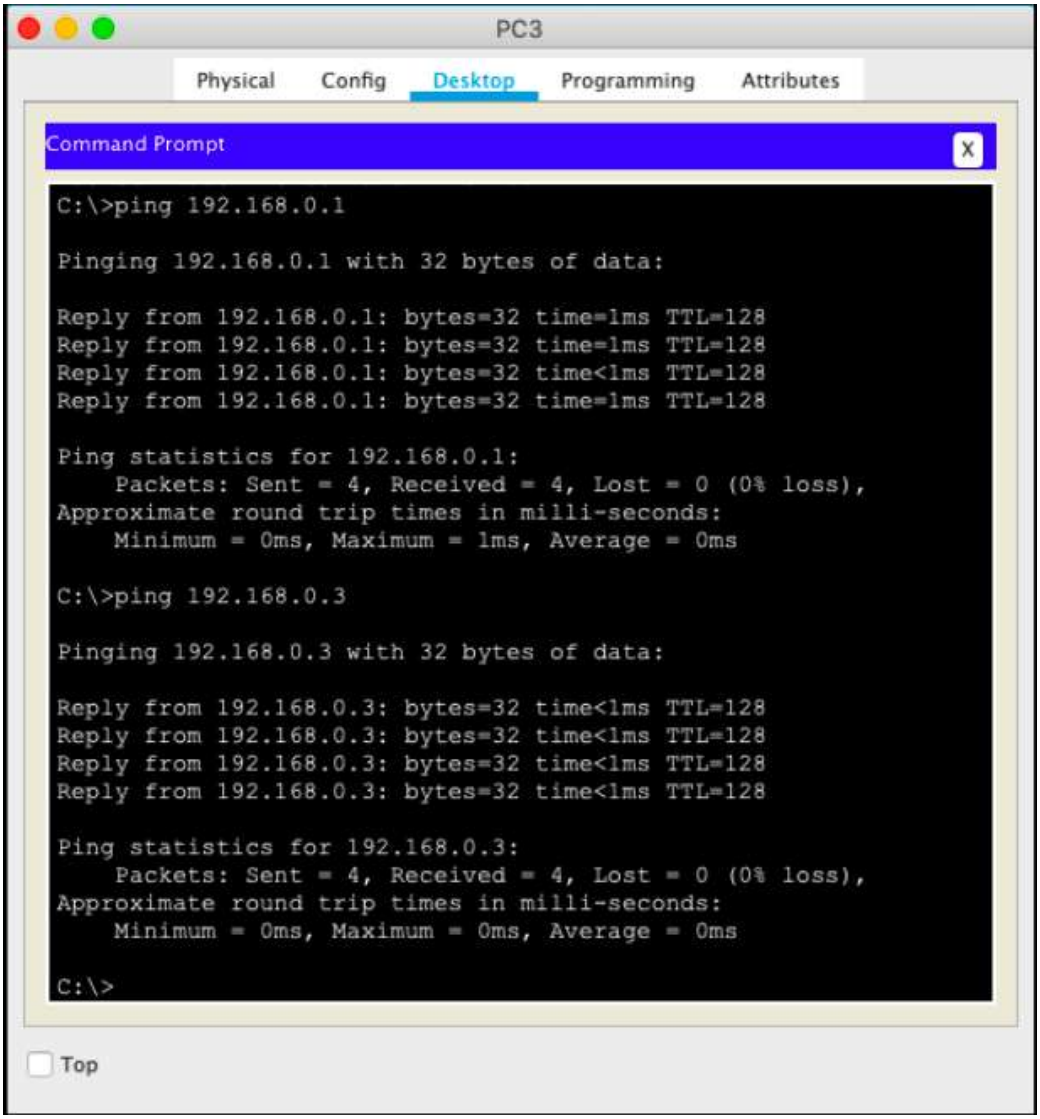
2. Create a LAN (named LAN-A) with 3 hosts using a hub. Ping each pair of nodes.



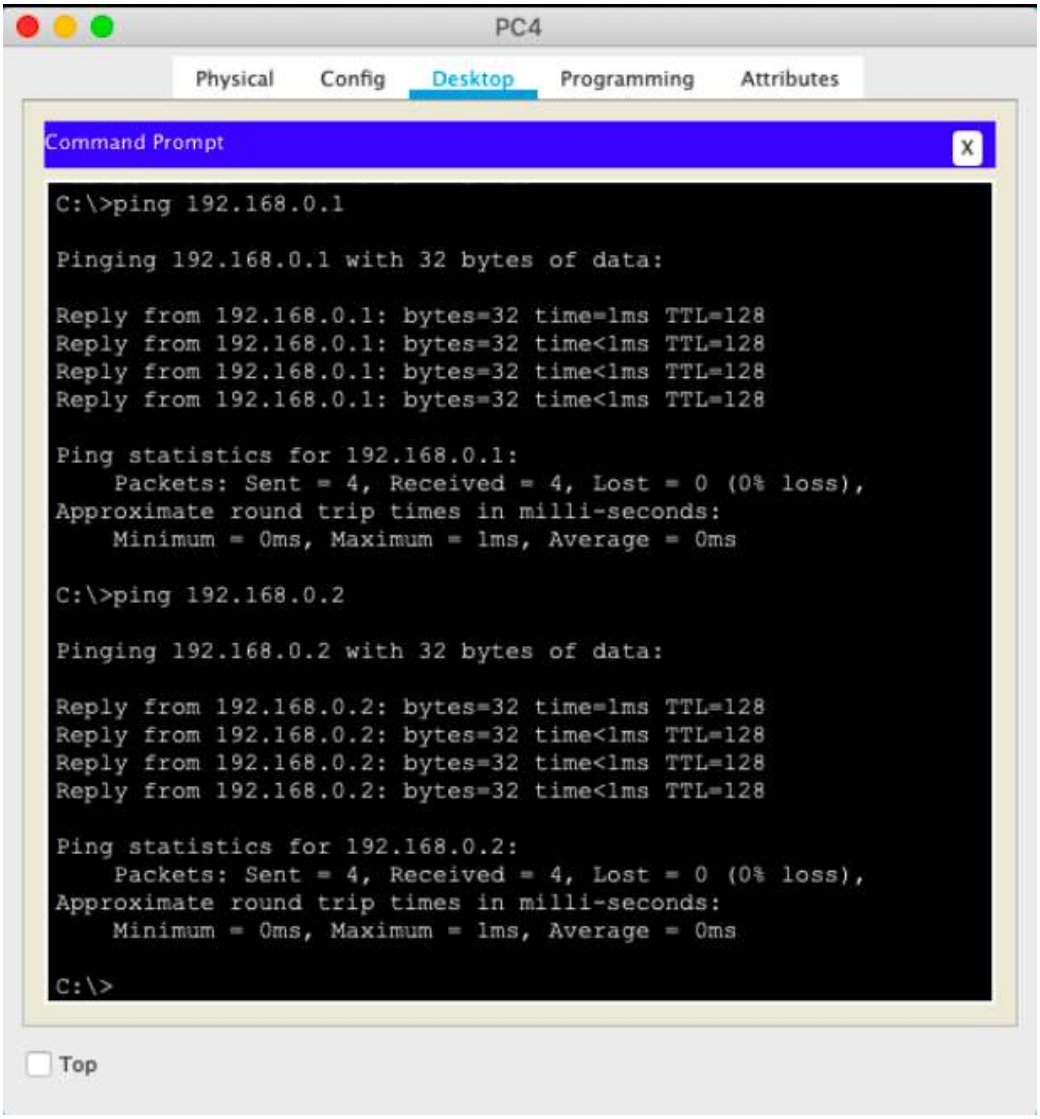
3 hosts connected using a hub creating a LAN.



Host 1 pinging Host 2 and Host 3.



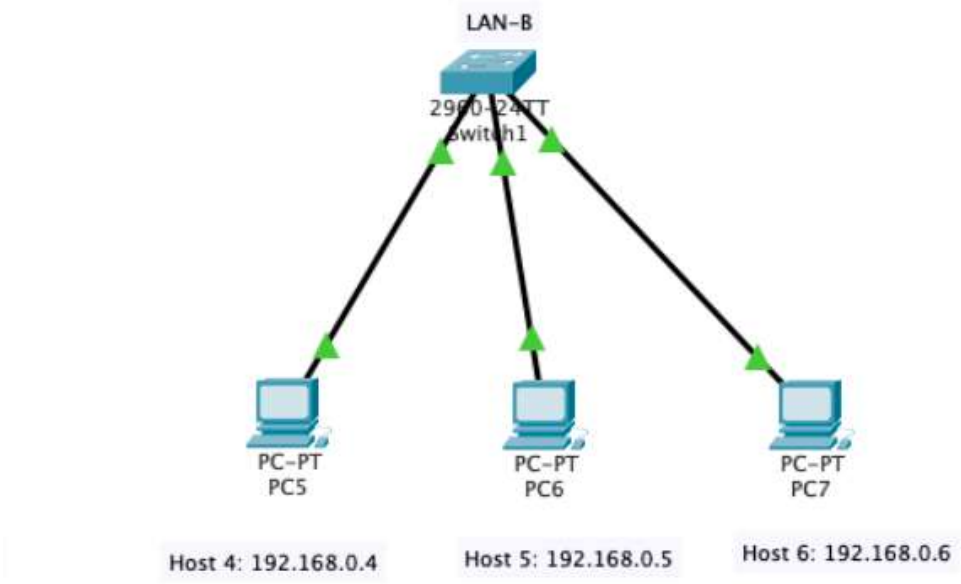
Host 2 pinging Host 1 and Host 3.



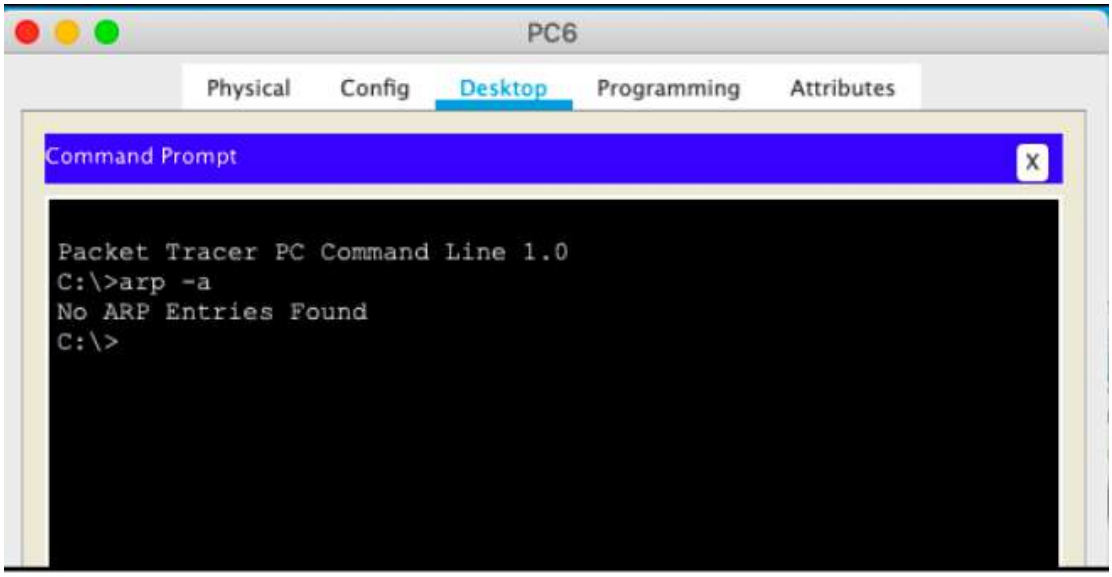
Host3 pinging Host 1 and Host 2.

Q3

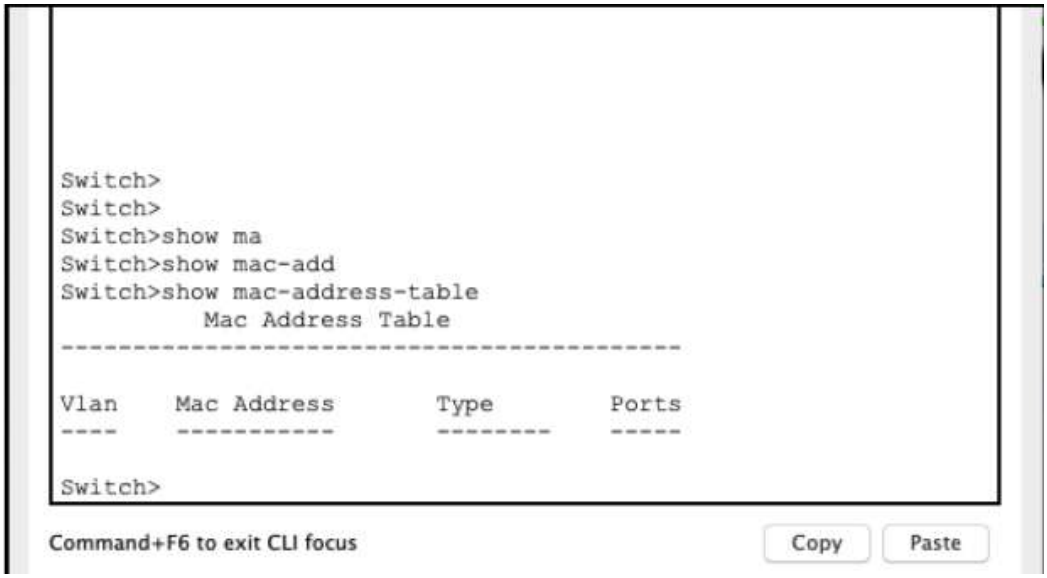
3. Create a LAN (named LAN-B) with 3 hosts using a switch. Record contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch. Ping each pair of nodes. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.



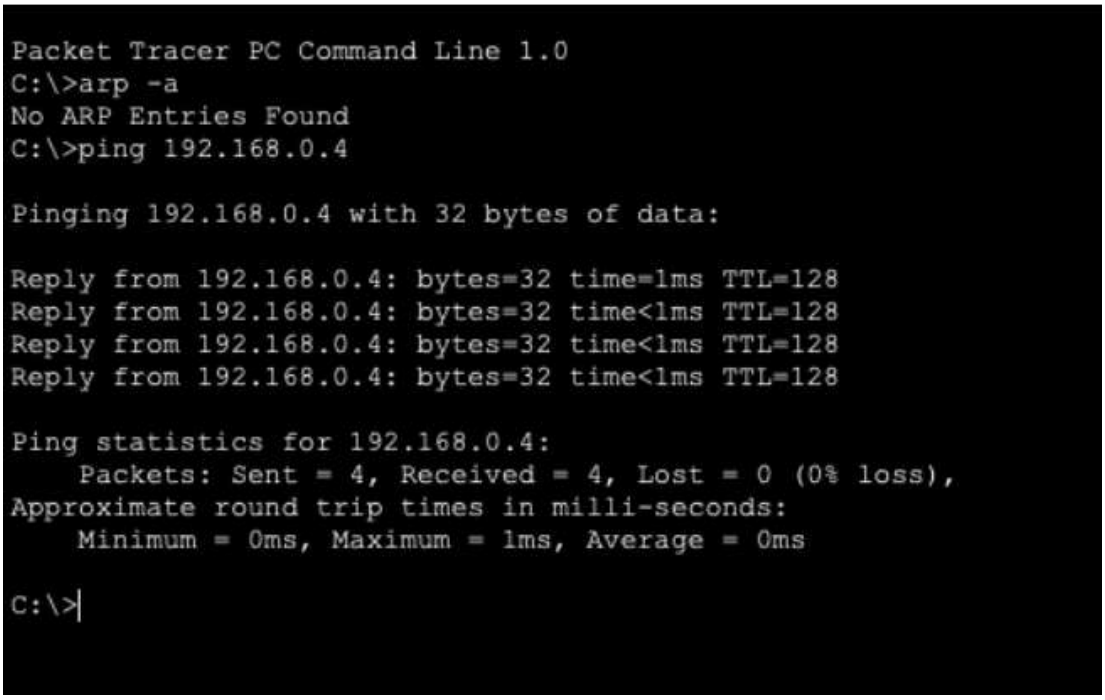
LAN-B created by connecting 3 hosts over a switch.



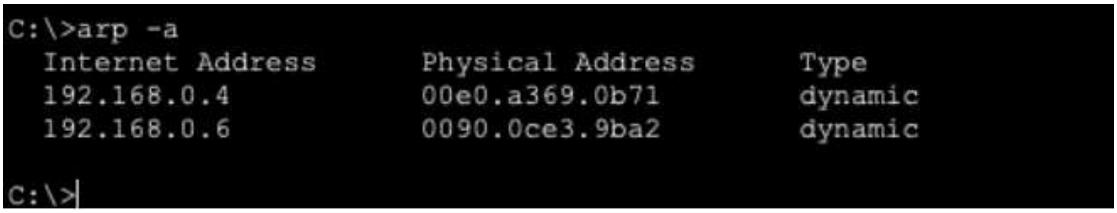
No ARP Table found in END Hosts



No Mac Address Table found in switch



Each pair of hosts pinged.



```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.5           00d0.ba01.e615        dynamic
  192.168.0.6           0090.0ce3.9ba2        dynamic
```

ARP table of host 4 and 5

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.4           00e0.a369.0b71        dynamic
  192.168.0.5           00d0.ba01.e615        dynamic
```

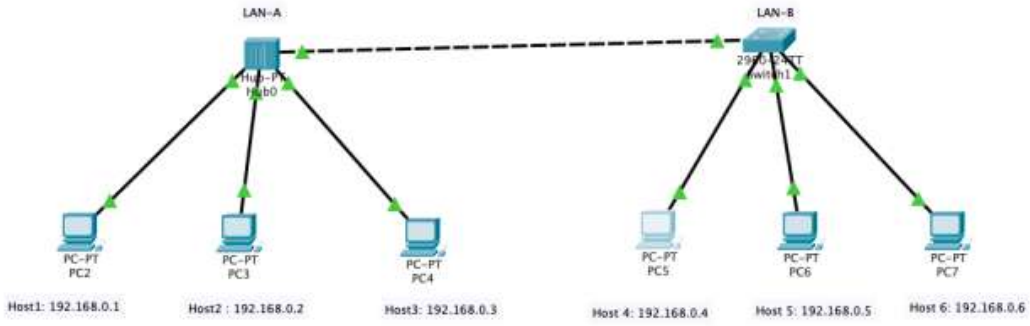
ARP table of host 6.

```
Switch#show mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

   1    0090.0ce3.9ba2    DYNAMIC     Fa0/3
   1    00d0.ba01.e615    DYNAMIC     Fa0/2
   1    00e0.a369.0b71    DYNAMIC     Fa0/1
Switch#
```

Mac Address Table of Switch after respective pings.

Q4

4. Connect LAN-A and LAN-B by connecting the hub and switch using a cross-over cable. Ping between each pair of hosts of LAN-A and LAN-B. Now record the contents of the ARP Table of end hosts and the MAC Forwarding Table of the switch again.



LAN-A and LAN-B connected

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.0.1           0060.3ee7.a69c        dynamic
  192.168.0.2           0004.9a6d.cbb4        dynamic
  192.168.0.3           0006.2a05.0945        dynamic
  192.168.0.5           00d0.ba01.e615        dynamic
  192.168.0.6           0090.0ce3.9ba2        dynamic
```

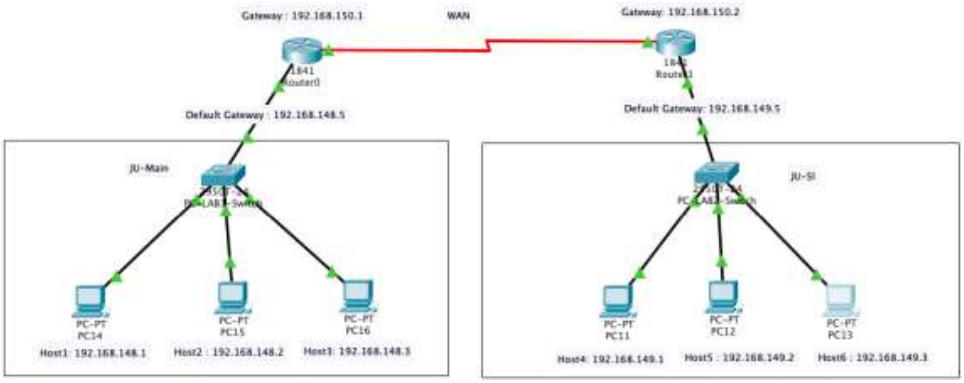
New formed ARP Table of the hosts showing addresses of Hosts 1,2 ,3 connected by the Hub.

```
Switch#
Switch#show mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----

   1    0004.9a6d.cbb4    DYNAMIC     Fa0/4
   1    0006.2a05.0945    DYNAMIC     Fa0/4
   1    0060.3ee7.a69c    DYNAMIC     Fa0/4
   1    0090.0ce3.9ba2    DYNAMIC     Fa0/3
   1    00e0.a369.0b71    DYNAMIC     Fa0/1
Switch#
```

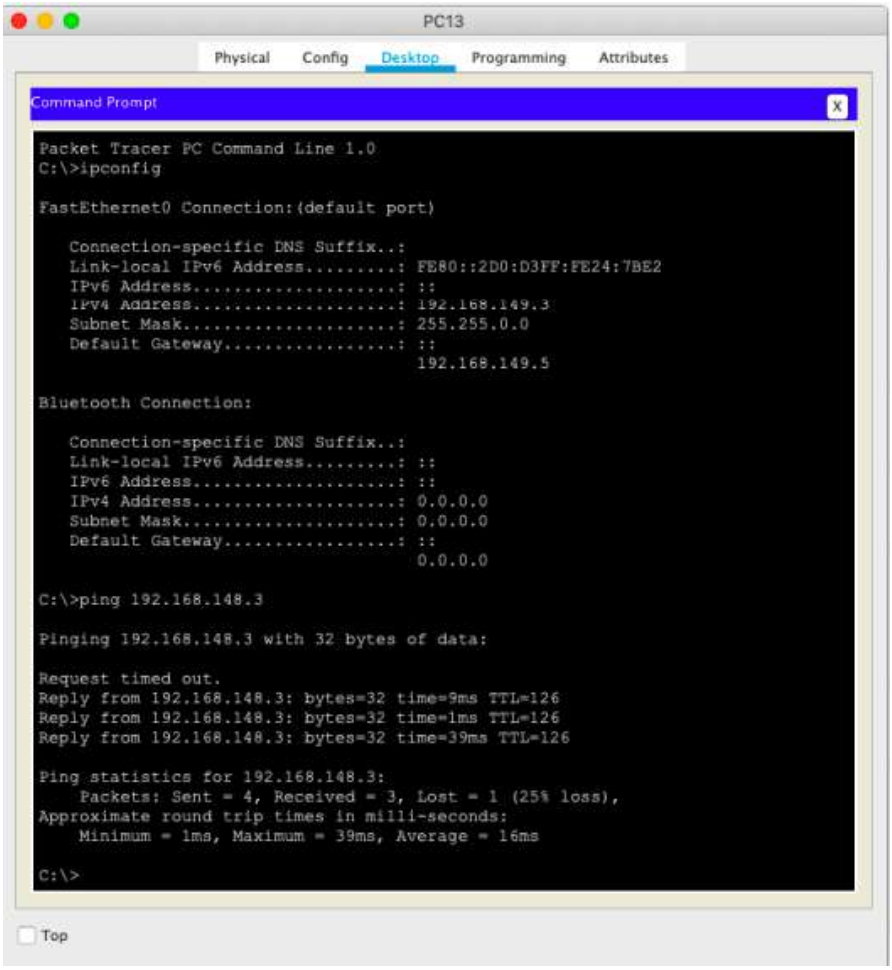
New mac-addresses shown in mac-address table, as Host 1 is pinged to Host 4, 5 and 6 connecting ethernet 4 of hub to switch. As new pings are made the mac address table is overwritten in switch.

Q5

5. Create a LAN (named JU-Main) with three hosts connected via a layer-2 switch (Cisco 2950 switch PC-LAB1-Switch). Connect the switch to a router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.148.0/24. Configure default gateway of each hosts as the IP address of the interface of the router which is connected to the LAN. Create another LAN (named JU-SL) with three hosts connected via a layer-2 switch (Cisco 2950 switch PC-LAB2-Switch). Connect this switch to another router (Cisco 1818). Assign IP addresses to all the hosts and the router interface connected to this LAN from network 192.168.149.0/24. Configure default gateway of each hosts as the IP address of the interface of the router which is connected to the LAN. Connect the two routers through appropriate WAN interfaces. Assign IP addresses to the WAN interfaces from network 192.168.150.0/24. Add static route in both of the routers to route packets between two LANs.



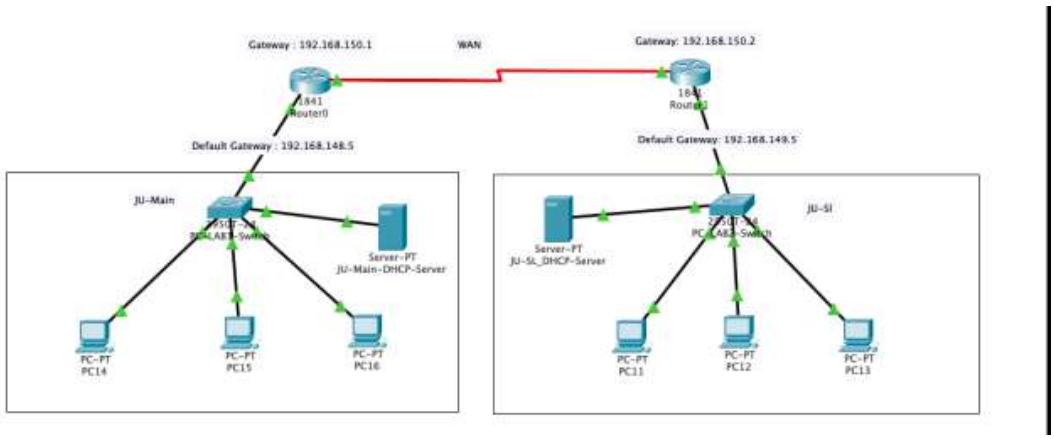
Two Local Area networks consisting of 3 hosts in ranges 192.168.148.0/24 and 192.168.149.0/24 connected by two switches respectively - connected over routers over a WAN with IP ranging from 192.168.150.0/24.



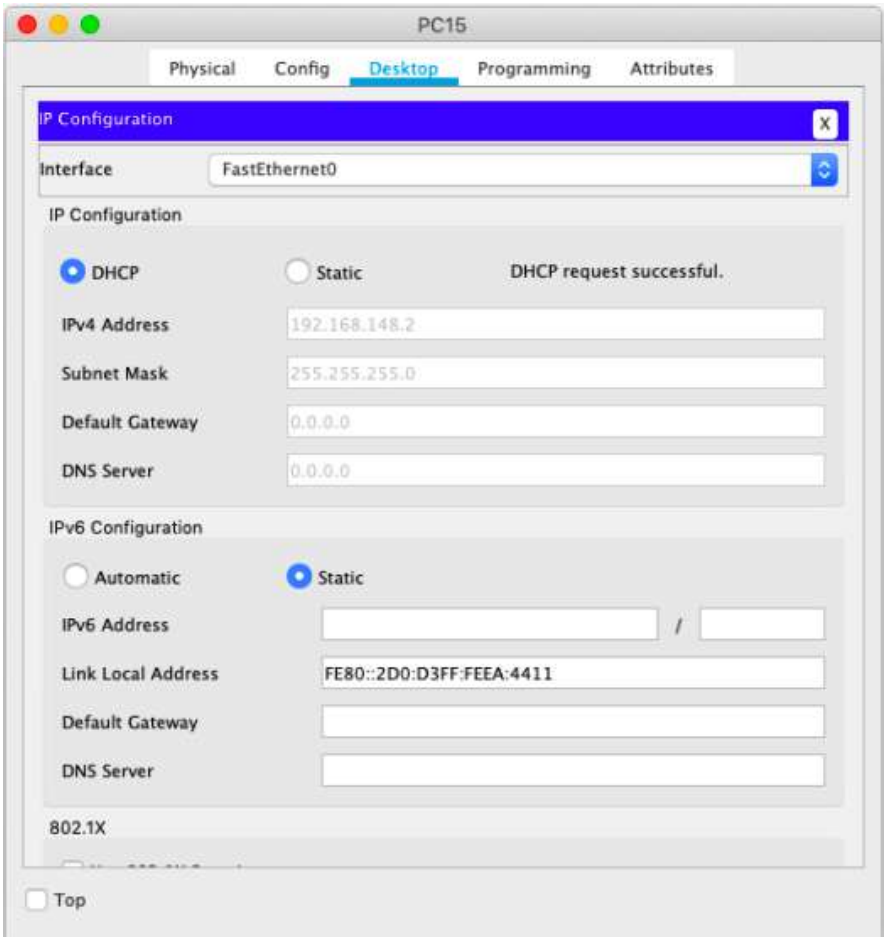
A host in JU-SL 192.168.149.3 pinging another host successfully in JU-MAIN with IP 192.168.148.3

Q6

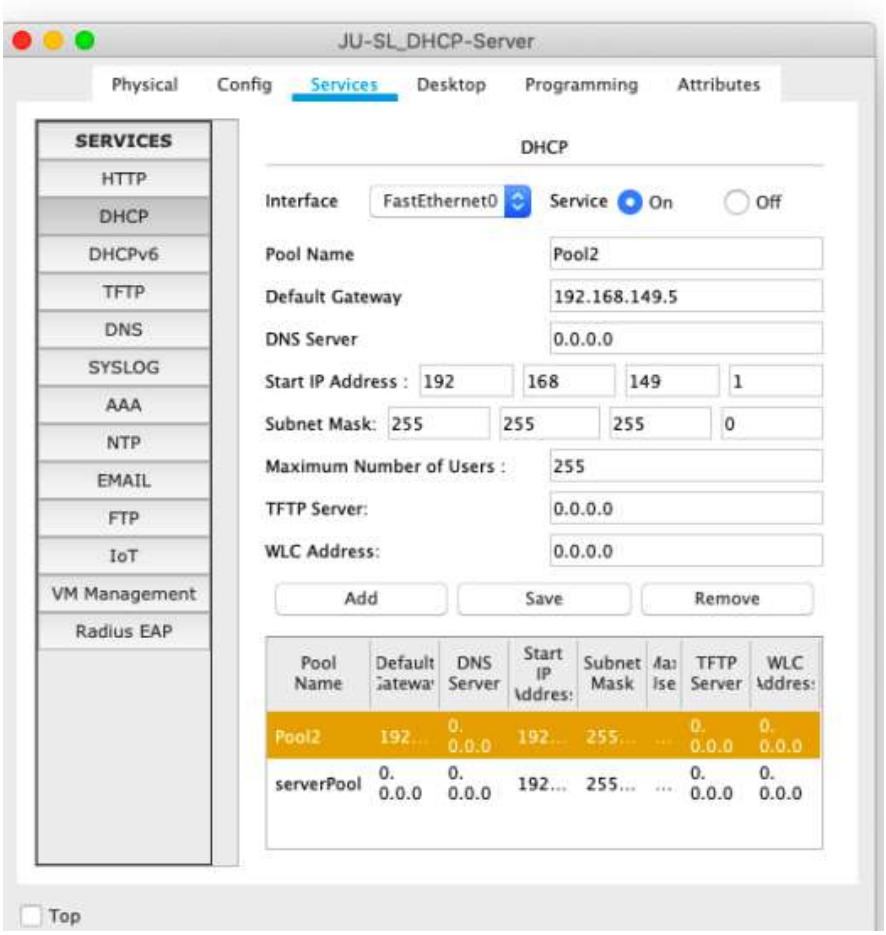
6. Add servers to the individual LANs (in problem 5) and configure them as a DHCP server. Configure the hosts in the individual LAN to obtain IP addresses and address of the default gateway via this DHCP server.



DHCP servers are added as an extension to network in 5, the host's IP configuration are modified to accommodate the dynamic change in IP.



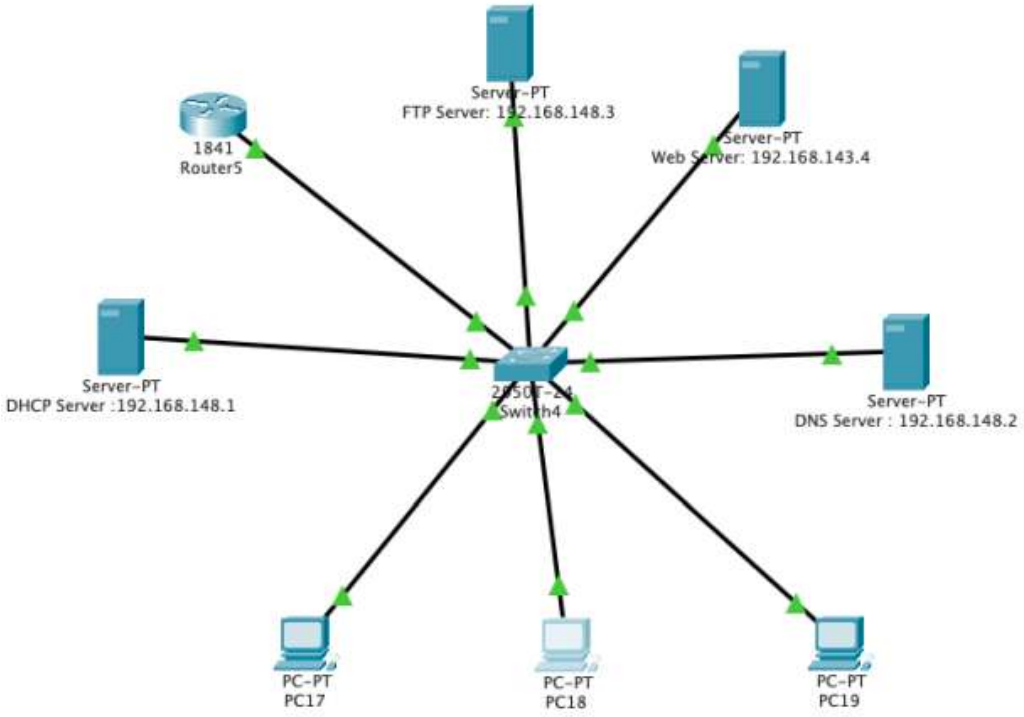
An example of host IP configuration in JU-Main getting its IP address dynamically as a result of DHCP.



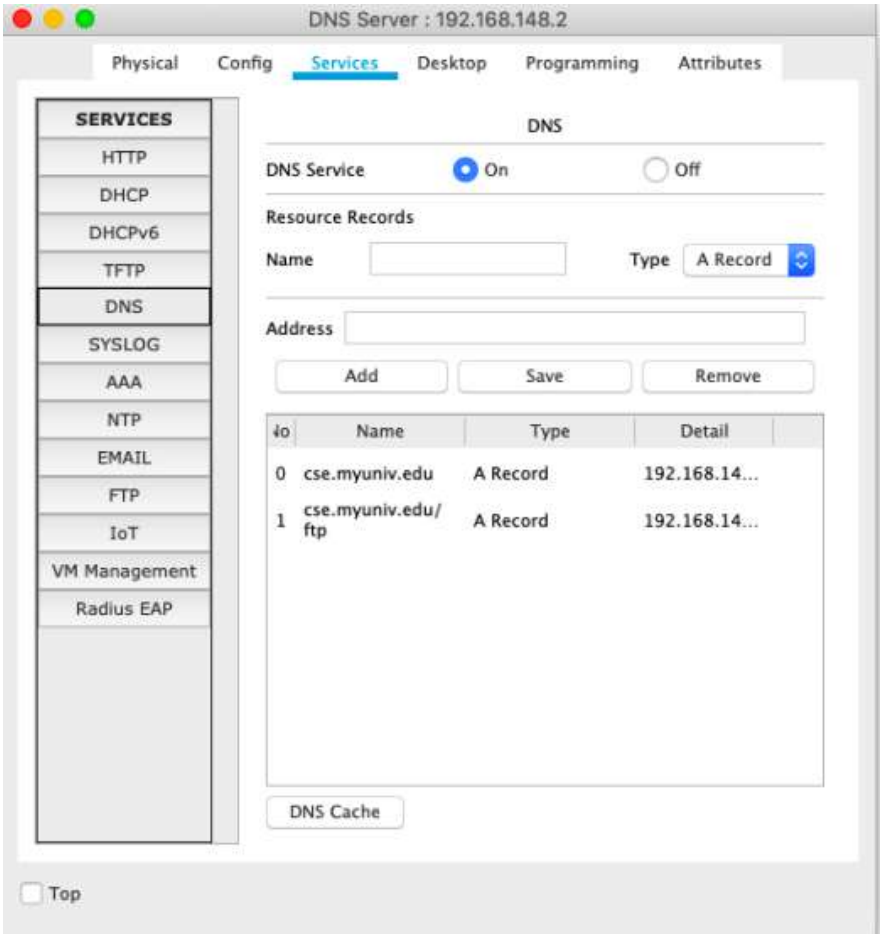
An example DHCP configuration in JUSL network.

Q7

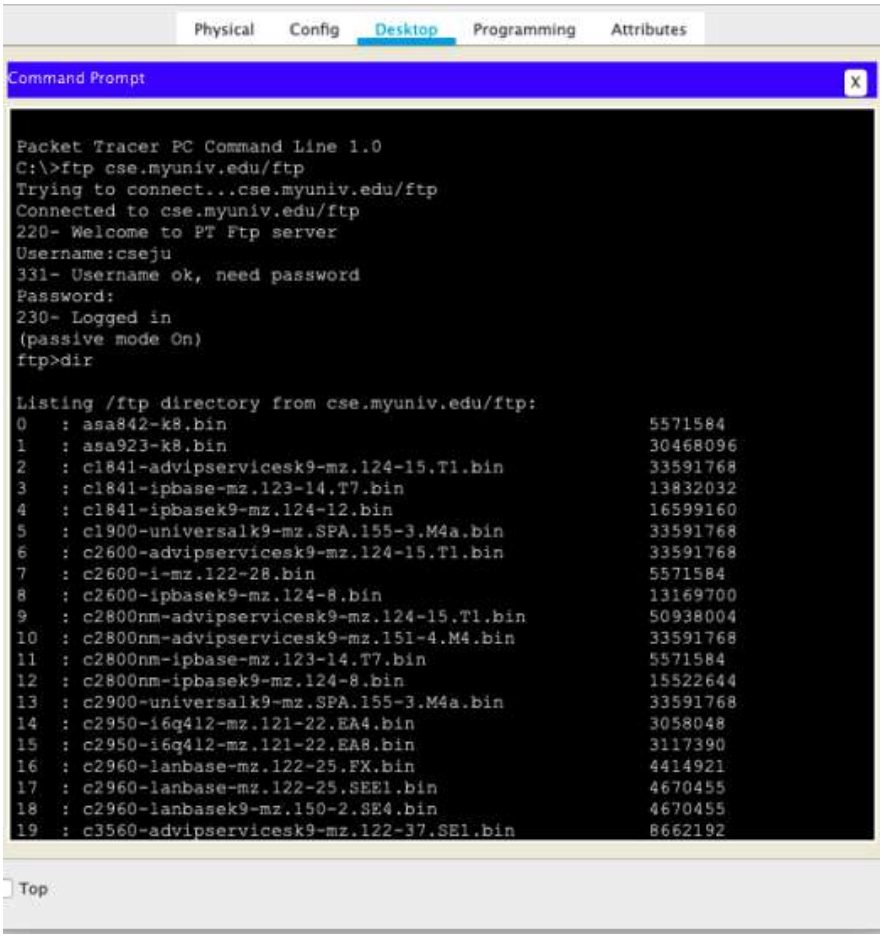
7. Create a LAN (CSE) with three hosts connected via a layer-2 switch (Cisco 2950 switch CSE-Switch). Also add a web server and a ftp server to this LAN. The hosts dynamically get their IP addresses from a local DHCP server. Servers are assigned fixed IP addresses. Configure the individual hosts to use the local DNS server for name resolution. Add a Domain Name Server (DNS) to this LAN. Create appropriate records in the DNS server for the individual servers in the LAN. The domain name of the LAN is cse.myuniv.edu. Configure the individual hosts to use the local DNS server for name resolution.



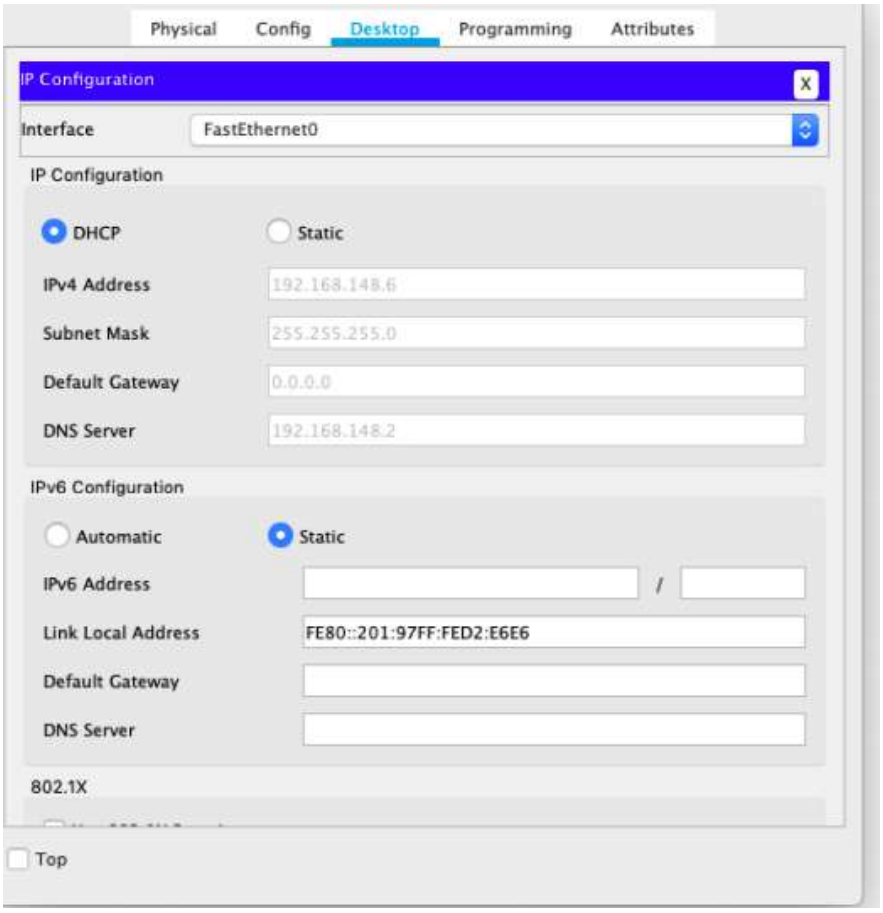
LAN(CSE) created with a switch connecting three switches and a FTP server and a Web server.



Shows the DNS records in the DNS server



Shows the working of a successfully FTP login and working in one of the hosts in the network.



Shows the Dynamically configured IP of one of the hosts through DHCP.

Computer Networks Lab Report - Assignment 7

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 7

Problem Statement:

Assignment 7: Network and Transport layer protocols
Submission due: 1st - 4th November 2022

Implement any two protocols using TCP/UDP Socket as suitable.

1. ARP
2. BOOTP
3. DHCP

Submission Date: **21 November, 2022**

Deadline: 4th November, 2022

Introduction

When a hosts send data to another host in a network, it generally gives destination ip address. Host to host delivery can only be acheived using MAC Addresses. Destination MAC addresses can be known using ARP Protocol which maps the mac address to its ip address.

DHCP is used to assign IP Addresses in a network based on availability of IP.

Design

ARP

Design of the Server:

The server is designed to be a simple ARP server. It will receive ARP packets from the clients and will send them to the destination client. The server will also send the ARP reply to the client who requested the ARP reply.

Design of the Client:

The client is designed to be a simple ARP client. It will send ARP packets to the server and will receive the ARP reply from the server. The client will also send the ARP reply to the destination client.

DHCP

Design of the Server:

The server will be designed in such a way that it will be able to handle multiple clients at the same time.

Initially, the server will be started and it will be waiting for the clients to connect to it. It will wait for the clients to send the request for the IP address.

The server will then check if the IP address is available or not. If the IP address is available, then the server will send the IP address to the client.

If the IP address is not available, then the server will send a message to the client that the IP address is not available.

The server will also be able to release the IP address if the client is not using it anymore.

Design of the Client:

The client will be designed in such a way that it will be able to send the request for the IP address to the server.

The client will also be able to release the IP address if it is not using it anymore.

The client will also be able to send the request for the IP address to the server if it has released the IP address.

Implementation

ARP:

Server

```
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 9999
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("[ SERVER ] Starting Server...")
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((IP, PORT))
print("[ SERVER ] Server binded to IP: " + IP + " and Port: " + str(PORT))
# This makes server listen to new connections
server_socket.listen()
# List of sockets for select.select()
sockets_list = [server_socket]
# List of connected clients - socket as a key, user header and name as data
clients = {}
# for storing the ARP requesting clients
ARP_request = {}
print(f'[ SERVER ] Listening for Connections on IP : {IP} PORT : {PORT}...')


# Handles message receiving
def receive_message(client_socket):
    try:
        # Receive our "header" containing message length, it's size is defined and con
stant
        message_header = client_socket.recv(HEADER_LENGTH)
        if not len(message_header):
            return False
        # Convert header to int value
```

```
        message_length = int(message_header.decode('utf-8').strip())
        # Return an object of message header and message data
        return {'header': message_header, 'data': client_socket.recv(message_length)}

    except:
        return False


while True:

    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)
    # Iterate over notified sockets
    for notified_socket in read_sockets:
        # If notified socket is a server socket - new connection, accept it
        if notified_socket == server_socket:
            # Accept new connection
            # That gives us new socket - client socket, connected to this given client
only, it's unique for that client
            # The other returned object is ip/port set
            client_socket, client_address = server_socket.accept()
            # Client should send his name right away, receive it
            user = receive_message(client_socket)

            # If False - client disconnected before he sent his name
            if user is False:
                continue
                # Add accepted socket to select.select() list
            sockets_list.append(client_socket)
            # Also save username and username header
            clients[client_socket] = user

            print('[ SERVER ] Accepted new Connection from username IP: {}'.format(use
r['data'].decode('utf-8')))

            # Else existing socket is sending a message
        else:
            # Receive message
            message = receive_message(notified_socket)

            # If False, client disconnected, cleanup
            if message is False:
                print('[ SERVER ] Closed connection from: {}'.format(clients[notified_
socket]['data'].decode('utf-8')))

                # Remove from list for socket.socket()
                sockets_list.remove(notified_socket)

                # Remove from our list of users
                del clients[notified_socket]

                continue
                # Get user by notified socket, so we will know who sent the message
            user = clients[notified_socket]

            print(f'[ SERVER ] Received message from IP {user["data"].decode("utf-
8")}:')
```

```
            # Iterate over connected clients and broadcast message
            # Splitting the incoming packet
            ARP_packet = message["data"].decode("utf-8").split()
            print("------------------SENDING PACKET ----------------------")
            print(f"[ SERVER ] SENDER IP: {ARP_packet[0]}")
            print(f"[ SERVER ] SENDER MAC: {ARP_packet[1]}")
            print("------------------------------------------------------")
            print(f"[ SERVER ] RECEIVER IP: {ARP_packet[2]}")
            print(f"[ SERVER ] RECEIVER MAC: {ARP_packet[3]}")
            print("------------------------------------------------------")

            # this is ARP request
            if ARP_packet[3] == "FF.FF.FF.FF.FF.FF":
                ARP_request[ARP_packet[0]] = notified_socket
                for client_socket in clients:

                    # But don't send it to sender
                    if client_socket != notified_socket:
                        # Send user and message (both with their headers) We are reusi
ng here message header sent by
                        # sender, and saved username header send by user when he conne
cted
                        client_socket.send(user['header'] + user['data'] + message['he
ader'] + message['data'])
            # else it is an ARP reply therefore it must be uni-cast.
            else:
                if ARP_packet[2] in ARP_request:
                    ARP_request[ARP_packet[2]].send(user['header'] + user['data'] + me
ssage['header'] + message['data'])

        # It's not really necessary to have this, but will handle some socket exceptio
ns just in case
    for notified_socket in exception_sockets:
        # Remove from list for socket.socket()
        sockets_list.remove(notified_socket)

        # Remove from our list of users
        del clients[notified_socket]
```

Client

```
import socket
import select
import errno
import sys

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 9999
my_IP = input("[ CLIENT ] Enter the IP for this Client: ")
my_Mac = input("[ CLIENT ] Enter the MAC for this Client: ")
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP, PORT))
```

```
# Set connection to non-blocking state, so .recv() call won;t block, just return some
 exception we'll handle
client_socket.setblocking(False)
username = my_IP.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-8')
client_socket.send(username_header + username)
while True:

    ip_receiver = input("[ CLIENT ] Enter the IP of Receiver: ")


    # this is for ARP request

    if ip_receiver:
        message = my_IP + " " + my_Mac + " " + ip_receiver + " " + "FF.FF.FF.FF.FF.FF"
        # Encode message to bytes, prepare header and convert to bytes, like for usern
ame above, then send
        message = message.encode('utf-8')
        message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
        client_socket.send(message_header + message)
    try:
        # Now we want to loop over received messages (there might be more than one) an
d print them
        while True:
            # Receive our "header" containing username length, it's size is defined an
d constant
            username_header = client_socket.recv(HEADER_LENGTH)
            # If we received no data, server gracefully closed a connection, for examp
le using socket.close() or
            # socket.shutdown(socket.SHUT_RDWR)
            if not len(username_header):
                print("[ CLIENT ] Connection closed by the Server...")
                sys.exit()

            # Convert header to int value
            username_length = int(username_header.decode('utf-8').strip())

            # Receive and decode username
            username = client_socket.recv(username_length).decode('utf-8')
            # Now do the same for message (as we received username, we received whole
 message, there's no need to
            # check if it has any length)
            message_header = client_socket.recv(HEADER_LENGTH)
            message_length = int(message_header.decode('utf-8').strip())
            message = client_socket.recv(message_length).decode('utf-8')
            ARP_request = message.split()
            if ARP_request[2] == my_IP:
                ARP_reply = my_IP + " " + my_Mac + " " + ARP_request[0] + " " + ARP_re
quest[1]
                ARP_reply = ARP_reply.encode('utf-8')
                ARQ_header = f"{len(ARP_reply):<{HEADER_LENGTH}}".encode('utf-8')
                client_socket.send(ARQ_header + ARP_reply)
            else:
                print("----------------------------------------")
                print("[ CLIENT ] Discarded the Request...")
                print("----------------------------------------")
            # Print message
            print(f'{username} > {message}')
```

```
    except IOError as e:
        # This is normal on non blocking connections - when there are no incoming data
error is going to be raised
        # Some operating systems will indicate that using AGAIN, and some using WOULDB
LOCK error code
        # We are going to check for both - if one of them - that's expected, means no
 incoming data, continue as normal
        # If we got different error code - something happened
        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
            print('[ CLIENT ] Reading Error: {}'.format(str(e)))
            sys.exit()

        # We just did not receive anything
        continue

    except Exception as e:
        # Any other exception - something happened, exit
        print(' [ CLIENT ] Reading Error: '.format(str(e)))
        sys.exit()
```

DHCP:

Server

```
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("[ Server ] Starting Server...")
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((IP, PORT))
# This makes server listen to new connections
server_socket.listen()
# List of sockets for select.select()
sockets_list = [server_socket]
# List of connected clients - socket as a key, user header and name as data
clients = {}
# for storing the ARP requesting clients
print("[ Server ] Waiting for connections...")
subnet_mask = "255.255.255.0"
network = "168.173.10."
available_ip = [True] * 254
available_ip[0] = False


# Handles message receiving
def receive_message(client_socket):
    try:
        # Receive our "header" containing message length, it's size is defined and con
stant
        message_header = client_socket.recv(HEADER_LENGTH)
```

```
        if not len(message_header):
            return False
        # Convert header to int value
        message_length = int(message_header.decode('utf-8').strip())
        # Return an object of message header and message data
        return {'header': message_header, 'data': client_socket.recv(message_length)}

    except:
        return False


while True:

    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)
    for notified_socket in read_sockets:
        if notified_socket == server_socket:
            client_socket, client_address = server_socket.accept()
            user = receive_message(client_socket)
            if user is False:
                continue
            sockets_list.append(client_socket)
            clients[client_socket] = user
            print('[ SERVER ] Accepted new connection from username IP: {}'.format(use
r['data'].decode('utf-8')))

            # Else existing socket is sending a message
        else:
            # Receive message
            message = receive_message(notified_socket)
            if message is False:
                print('[ SERVER ] Closed Connection from: {}'.format(clients[notified_
socket]['data'].decode('utf-8')))
                sockets_list.remove(notified_socket)
                del clients[notified_socket]
                continue
                # Get user by notified socket, so we will know who sent the message
            user = clients[notified_socket]

            print(f'[ SERVER ] Received DHCP request from Client {user["data"].decode
("utf-8")}:')
            ARP_packet = message["data"].decode("utf-8").split()
            if ARP_packet[0] == "release":
                print("[ SERVER ] Releasing...")
                index = ARP_packet[1][11:]
                index = int(index)
                available_ip[index] = True
                DHCP_reply = "0.0.0.0"
                DHCP_reply = DHCP_reply.encode('utf-8')
                DHCP_header = f"{len(DHCP_reply):<{HEADER_LENGTH}}".encode('utf-8')
                notified_socket.send(user['header'] + DHCP_header + DHCP_reply)
                print("[ SERVER ] DHCP Reply Sent...")

            if ARP_packet[0] == "request":
                print("[ SERVER ] Requesting...")
                for i in range(1, 254):
                    if available_ip[i]:
                        available_ip[i]=False
                        print("[ SERVER ] Available!")
```

```
                        DHCP_reply = network + str(i)
                        print(DHCP_reply)
                        DHCP_reply = DHCP_reply.encode('utf-8')
                        DHCP_header = f"{len(DHCP_reply):<{HEADER_LENGTH}}".encode('ut
f-8')
                        notified_socket.send(user['header'] + DHCP_header + DHCP_repl
y)
                        print("[ SERVER ] DHCP Reply Sent...")
                        break

        # It's not really necessary to have this, but will handle some socket exceptio
ns just in case
    for notified_socket in exception_sockets:
        # Remove from list for socket.socket()
        sockets_list.remove(notified_socket)

        # Remove from our list of users
        del clients[notified_socket]
```

Client

```
import socket
import select
import errno
import sys

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
my_IP = "0.0.0.0"
username=input("[ CLIENT ] Enter Client's name : ")
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP, PORT))
# Set connection to non-blocking state, so .recv() call won;t block, just return some
 exception we'll handle
client_socket.setblocking(False)
username = username.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-8')
client_socket.send(username_header + username)
while True:
    message = "ipconfig"
    while message == "ipconfig":
        message = input("[ CLIENT ] Enter message : ")
        print("_____________________________________")
        print(f"[ CLIENT ] IP :{my_IP}")
        print("_____________________________________")

    if message:
        message = message + " " + my_IP
        # Encode message to bytes, prepare header and convert to bytes, like for usern
ame above, then send
        message = message.encode('utf-8')
        message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
        client_socket.send(message_header + message)
```

```
    try:
        while True:
            username_header = client_socket.recv(HEADER_LENGTH)
            if not len(username_header):
                print('[ CLIENT ] : Connection closed by the Server...')
                sys.exit()
            message_header = client_socket.recv(HEADER_LENGTH)
            print(message_header.decode('utf-8'))
            message_length = int(message_header.decode('utf-8').strip())
            message = client_socket.recv(message_length).decode('utf-8')
            print(message)
            my_IP = message

    except IOError as e:
        # This is normal on non blocking connections - when there are no incoming data
error is going to be raised
        # Some operating systems will indicate that using AGAIN, and some using WOULDB
LOCK error code
        # We are going to check for both - if one of them - that's expected, means no
 incoming data, continue as normal
        # If we got different error code - something happened
        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
            print('[ CLIENT ] Reading error: {}'.format(str(e)))
            sys.exit()

        # We just did not receive anything
        continue

    except Exception as e:
        # Any other exception - something happened, exit
        print('[ CLIENT ] Reading error: '.format(str(e)))
        sys.exit()
```

Test Cases and Results

ARP:

```
[ SERVER ] Server binded to IP: 127.0.0.1 and Port: 9999
[ SERVER ] Listening for Connections on IP : 127.0.0.1 PORT : 9999...
[ SERVER ] Accepted new Connection from username IP: 1
[ SERVER ] Accepted new Connection from username IP: 2
[ SERVER ] Received message from IP 1:
------------------SENDING PACKET ----------------------
[ SERVER ] SENDER IP: 1
[ SERVER ] SENDER MAC: A
-----------------------------------------------------------
[ SERVER ] RECEIVER IP: 2
[ SERVER ] RECEIVER MAC: FF.FF.FF.FF.FF.FF
-----------------------------------------------------------
[ SERVER ] Received message from IP 2:
------------------SENDING PACKET ----------------------
[ SERVER ] SENDER IP: 2
[ SERVER ] SENDER MAC: B
-----------------------------------------------------------
[ SERVER ] RECEIVER IP: 1
[ SERVER ] RECEIVER MAC: A
-----------------------------------------------------------
[ SERVER ] Received message from IP 1:
------------------SENDING PACKET ----------------------
[ SERVER ] SENDER IP: 1
[ SERVER ] SENDER MAC: A
-----------------------------------------------------------
[ SERVER ] RECEIVER IP: 2
[ SERVER ] RECEIVER MAC: B
-----------------------------------------------------------
```

```
thebikashsah@BIKASHs-MacBook-Air ARP % python3 client.py
[ CLIENT ] Enter the IP for this Client: 1
[ CLIENT ] Enter the MAC for this Client: A
[ CLIENT ] Enter the IP of Receiver: 2
[ CLIENT ] Enter the IP of Receiver:
2 > 2 B 1 A
[ CLIENT ] Enter the IP of Receiver:
```

```
thebikashsah@BIKASHs-MacBook-Air ARP % python3 client.py
[ CLIENT ] Enter the IP for this Client: 2
[ CLIENT ] Enter the MAC for this Client: B
[ CLIENT ] Enter the IP of Receiver:
1 > 1 A 2 FF.FF.FF.FF.FF.FF
[ CLIENT ] Enter the IP of Receiver:
```

DHCP:

```
thebikashsah@BIKASHs-MacBook-Air DHCP % python3 server.py
[ Server ] Starting Server...
[ Server ] Waiting for connections...
[ SERVER ] Accepted new connection from username IP: 1.1.1.1
[ SERVER ] Received DHCP request from Client 1.1.1.1:
[ SERVER ] Requesting...
[ SERVER ] Available!
168.173.10.1
[ SERVER ] DHCP Reply Sent...
```

```
thebikashsah@BIKASHs-MacBook-Air DHCP % python3 client.py
[ CLIENT ] Enter Client's name : 1.1.1.1
[ CLIENT ] Enter message : ipconfig
_____________________________
[ CLIENT ] IP :0.0.0.0
_____________________________
[ CLIENT ] Enter message : request
_____________________________
[ CLIENT ] IP :0.0.0.0
_____________________________
[ CLIENT ] Enter message :
_____________________________
[ CLIENT ] IP :0.0.0.0
_____________________________
12
168.173.10.1
[ CLIENT ] Enter message : ipconfig
_____________________________
[ CLIENT ] IP :168.173.10.1
_____________________________
[ CLIENT ] Enter message :
```

Comments

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot, I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.

Computer Networks Lab Report - Assignment 8

Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 8

Problem Statement:

Assignment 8: Application and Transport layer protocols
Submission due: 7th - 11th November 2022

Implement any two protocols using TCP/UDP Socket as suitable.

1. FTP
2. DNS
3. Telnet

Submission Date: **21 November, 2022**

Deadline: 11th November, 2022

Implementation

FTP:

Server

```
package FTP;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

 class FileServer {
    public static void main(String[] args) throws Exception {
        // Initialize Sockets
        ServerSocket ssock = new ServerSocket(5000);
        Socket socket = ssock.accept();
        // The InetAddress specification
        InetAddress IA = InetAddress.getByName("localhost");
        // Specify the file
        // Input the file location
        String file_location = "input.txt";
        File file = new File(file_location);
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);
        // Get socket's output stream
        OutputStream os = socket.getOutputStream();
        // Read File Contents into contents array
        byte[] contents;
        long fileLength = file.length();
        long current = 0;
        long start = System.nanoTime();
        while (current != fileLength) {
            int size = 10000;
            if (fileLength - current >= size)
                current += size;
            else {
                size = (int) (fileLength - current);
                current = fileLength;
            }
            contents = new byte[size];
            bis.read(contents, 0, size);
            os.write(contents);
            System.out.print("Sending file ... " + (current * 100) / fileLength + "% c
omplete!");
        }
        os.flush();
        // File transfer done. Close the socket connection!
        socket.close();
        ssock.close();
        System.out.println("File sent succesfully!");
    }
}
```

Client

```
package FTP;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

class FileClient {
    public static void main(String[] args) throws Exception {
        // Initialize socket
        Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
        byte[] contents = new byte[10000];
        // Initialize the FileOutputStream to the output file's full path.
        FileOutputStream fos = new FileOutputStream("output.txt");
        // FileOutputStream fos = new FileOutputStream("e:\\Bookmarks1.html");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        InputStream is = socket.getInputStream();
        // No of bytes read in one read() call
        int bytesRead = 0;
        while ((bytesRead = is.read(contents)) != -1)
            bos.write(contents, 0, bytesRead);
        bos.flush();
        socket.close();
        System.out.println("File saved successfully!");
    }
}
```

DNS:

Local Server

```
# Implementation of DNS Local Server

# The local server accepts connection from the client and sends the IP address of the
 domain name to the client.

# The local server will have its own cache in which it will store the < domain name, I
P address > pairs. This cache will be used to resolve the domain names.

# If the domain name is not present in the cache, then the local server will send a qu
ery to the root server to get the IP address of the domain name.

# The root server will send the IP Address of the .com server or the .in server to the
local server, according to the domain name.

# The local server will then send a query to the .com server or the .in server to get
 the IP address of the domain name.

# The .com server or the .in server will send the IP address of the domain name to the
```

```
local server.

# The local server will then send the IP address of the domain name to the client.

# The local server will store the < domain name, IP address > pair in its cache.


# Importing the socket library and the time library
import socket
import time

IP = "127.0.0.1"
PORT = 9999

# IP address of the root server and the port number which will be used to connect to t
he root server
ROOT_IP ="127.0.0.3"
ROOT_PORT=9998

# Create the cache dictionary
cache = {}

Found=False

# Dictionary of domain names with their IP addresses in format < domain name, IP addre
ss >
# google.com, 100.100.100.1
# facebook.com, 100.100.100.2
# youtube.com, 100.100.100.3

# instagram.in, 99.99.99.1
# twitter.in, 99.99.99.2

dict_domain={}
dict_domain["google.com"]="100.100.100.1"
dict_domain["facebook.com"]="100.100.100.2"
dict_domain["youtube.com"]="100.100.100.3"
dict_domain["instagram.in"]="99.99.99.1"
dict_domain["twitter.in"]="99.99.99.2"



# Create the socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# print ("Socket successfully created")
print("[ Local Server ]: Socket successfully created...")
# Bind the socket to the port
s.bind((IP, PORT))
# print ("socket binded to %s" %(PORT))
print(f"[ Local Server ]: Socket binded to {PORT}...")
# Put the socket into listening mode
s.listen(5)
# print ("socket is listening")
print("[ Local Server ]: Socket is listening...")

# Connect to the client
c, addr = s.accept()
# print ('Got connection from', addr)
```

```
print(f"[ Local Server ]: Got connection from {addr}...")
# Receive the domain name from the client
domain_name = c.recv(1024).decode()
# print("The domain name is: ",domain_name)
print(f"[ Local Server ]: The domain name is: {domain_name}...")

# Check if the domain name is present in the cache
for i in cache:
    if i==domain_name:
        Found=True
        # print("The IP address is: ",cache[i])
        print(f"[ Local Server ]: The IP address is: {cache[i]}...")
        # Send the IP address to the client
        c.send(cache[i].encode())
        # Close the connection
        c.close()
        # Close the socket
        s.close()
        break


# Function to send the query to the root server


def send_query(domain_name):
    isReal=False
    Inter_IP=""
    # The root server will send the IP Address along with a flag to the local server,
 the flag will be 1 if the domain name is present in the root server and 0 if the doma
in name is not present in the root server.
    # If the flag is 1, then the local server will send the IP address of the domain n
ame to the client.
    # If the flag is 0, then the local server will send a query to the Inter_IP that i
t will receive from the root server. The Inter_IP will be the IP address of the .com s
erver or the .in server.
    TEMP_IP ="localhost"
    TEMP_PORT=9998
    while isReal==True:
        # Connect to the root server
        s.connect((TEMP_IP,TEMP_PORT))
        # Send the domain name to the root server
        s.send(domain_name.encode())
        # Receive the IP address and the flag from the root server
        Inter_IP=s.recv(1024).decode()
        # Received format: < IP address, port , flag >

        # Split the received string to get the IP address and the flag

        # Get the IP address
        TEMP_IP=Inter_IP.split(",")[0]
        # Get the port number
        TEMP_PORT=Inter_IP.split(",")[1]
        # Get the flag

        Real=Inter_IP.split(",")[2]
        isReal=int(Real)
    # Search in the dictionary of domain names and IP addresses
    for i in dict_domain:
```

```
        if i==domain_name:
            # print("The IP address is: ",dict_domain[i])
            print(f"[ Local Server ]: The IP address is: {dict_domain[i]}...")
            # Send the IP address to the client
            TEMP_IP=dict_domain[i]
            print("[ Local Server ]: IP address sent to the client...")
            # Close the connection

            # Close the socket

            break
    # Now the IP address of the domain name is present in the ROOT_IP and the ROOT_POR
T
    # Now send the IP address of the domain name to the client
    # c.send(TEMP_IP.encode())

    # Check if the domain is .com or .in

    # If the domain is .com, then send the query to the .com server
    # if domain_name.split(".")[1]=="com":
    #     TEMP_IP="192.00.00.00"
    # Close the connection

    # Close the socket

    # Store the < domain name, IP address > pair in the cache
    cache[domain_name]=TEMP_IP

    return TEMP_IP




IP_Address=""

if Found==False:
    IP_Address=send_query(domain_name)
    if IP_Address!="":
        # Store the < domain name, IP address > pair in the cache
        cache[domain_name]=IP_Address
        # Send the IP address to the client
        c.send(IP_Address.encode())
        # Close the connection
        c.close()
        # Close the socket
        s.close()
```

Client:

```
# Implementation of DNS client

# Client will have its own local cache in which it will store the
# < domain name, IP address > pairs. This cache will be used to
# resolve the domain names.
```

```
# If the domain name is not present in the cache, then the client will send a query to
the local server.

# When the client receives the response from the server, it will store the < domain na
me, IP address > pair in its cache.


# The client will only contact the local server. It will not contact any other server.


# Importing the socket library and the time library
import socket
import time

# Store the IP address of the local server and the port number
IP="localhost"
PORT=9999

# Create the cache dictionary
cache={}
# Create the socket object
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Take the domain name as input from the user
domain_name=input("Enter the domain name: ")

# Check if the domain name is present in the cache

isFound=False
for i in cache:
    if i==domain_name:
        isFound=True
        print("The IP address is: ",cache[i])
        break

IP_ADDRESS=""

# Function to send the query to the server
def send_query(domain_name):
    # Connect to the server
    s.connect((IP,PORT))
    # Send the domain name to the server
    s.send(domain_name.encode())
    # Receive the IP address from the server
    IP_ADDRESS=s.recv(1024).decode()
    # Close the connection
    # s.close()
    # Return the IP address
    return IP_ADDRESS

if not isFound:
    # If the domain is not found call a function to send the query to the server
    IP_ADDRESS=send_query(domain_name)

    # If the IP address is not empty, then store the < domain name, IP address > pair
 in the cache

    if IP_ADDRESS!="":
```

```
        cache[domain_name]=IP_ADDRESS
        print("The IP address is::",IP_ADDRESS)
    else:
        print("The domain name is not present in the DNS server")
```

Root Server:

```
# Implementation of DNS Root Server

# The root server accepts connection from the local server and sends the IP address of
the .com server or the .in server to the local server, according to the domain name.

import socket

# IP address of the root server and the port number which will be used to connect to t
he root server

ROOT_IP ="127.0.0.1"
ROOT_PORT=9998


IP_COM="127.0.0.4"
PORT_COM=9997

IP_IN="10.10.10.4"
PORT_IN=9996



# It will receive the domain name from the local server and send the IP address of the
.com server or the .in server to the local server, according to the domain name.

# Socket object
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Bind the socket to the port
s.bind((ROOT_IP,ROOT_PORT))

# Put the socket into listening mode
s.listen(5)
print("[ Root Server ]: Socket is listening...")

# Connect to the local server
c,addr=s.accept()

# Receive the domain name from the local server
domain_name=c.recv(1024).decode()

print(f"[ Root Server ]: The domain name is: {domain_name}...")
# Extract the extension of the domain name
extension=domain_name.split(".")[-1]



# Send the IP address of the .com server or the .in server to the local server, accord
```

```
ing to the domain name.
# It will send in the format < IP address port number 0 >
if extension=="com":
    print(f"[ Root Server ]: Sending IP address of .com server to the local serve
r...")
    c.send((IP_COM+" "+str(PORT_COM)+" 0").encode())
elif extension=="in":
    print(f"[ Root Server ]: Sending IP address of .in server to the local server...")
    c.send((IP_IN+" "+str(PORT_IN)+" 0").encode())

# Close the connection
c.close()
# Close the socket
s.close()

# Write in the file pychache.txt the IP address of the .com server and the .in server
```

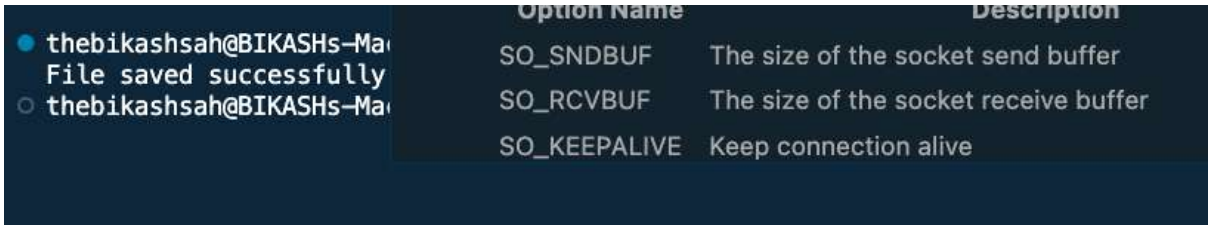
com server

```
# Implemenation of DNS .com Server
# It will store the < domain name, IP address > pairs in the cache.
# It will accept connection from the local server and send the IP address of the domai
n name to the local server.


import socket


IP_COM="127.0.0.1"
PORT_COM=9997

# Create the cache dictionary
cache={}
cache["google.com"]="199.99.99.99"
cache["facebook.com"]="200.200.200.200"

# Create the socket object
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Bind the socket to the port
s.bind((IP_COM,PORT_COM))

# Put the socket into listening mode
s.listen(5)
print("[ .com Server ]: Socket is listening...")

# Connect to the local server
c,addr=s.accept()
print(f"[ .com Server ]: Got connection from {addr}...")
# Receive the domain name from the local server
domain_name=c.recv(1024).decode()
print(f"[ .com Server ]: The domain name is: {domain_name}...")

# Send the IP address of the domain name to the local server
```

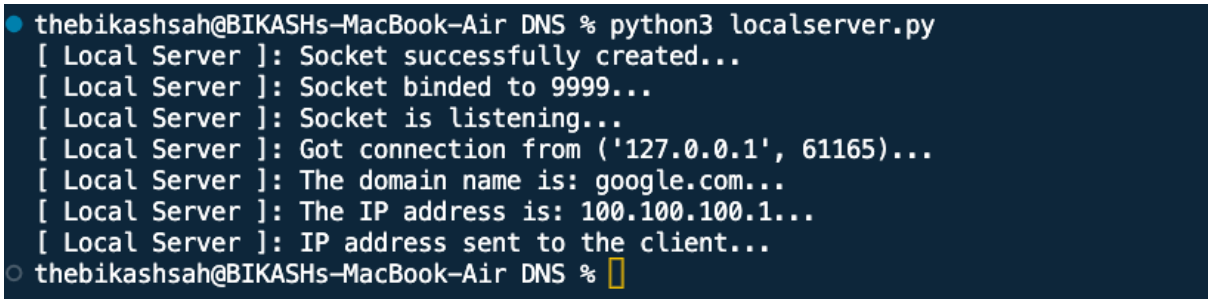
```
# Send in the format < IP address port number 1 >
c.send((cache[domain_name]+" "+str(PORT_COM)+" 1").encode())
print(f"[ .com Server ]: The IP address is: {cache[domain_name]}...")

# Close the connection
c.close()
# Close the socket
s.close()
```

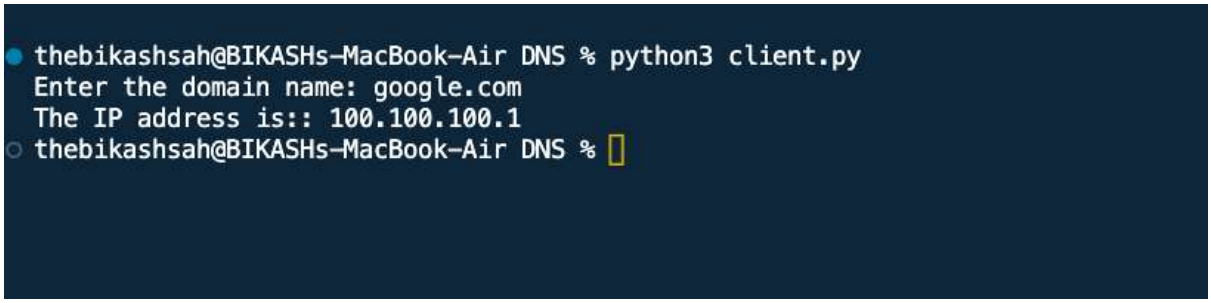
Test Cases and Results

FTP:



DNS:





Comments

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot, I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.