COMPUTER NETWORKS LAB

NAME: TANMOY SARKAR

ROLL NO: 002010501020

CLASS: BCSE-III First Semester

GROUP: A1

CONTENTS

Computer Networks Lab Report – Assignment 1

Name: Tanmoy Sarkar

Roll No: 002010501020

Class: BCSE 5th semester 3rd Year

Group: A1

Problem statement: Design and implement an error detection module

Design and implement an error detection module that 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, codewords will be prepared. The sender will send the codeword to the Receiver. The receiver will extract the data word from the codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for the following cases (not limited to).

A. Error is detected by all four schemes. Use a suitable CRC polynomial (the list is given on the 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: 07-08-2022

Deadline Date: 07-08-2022

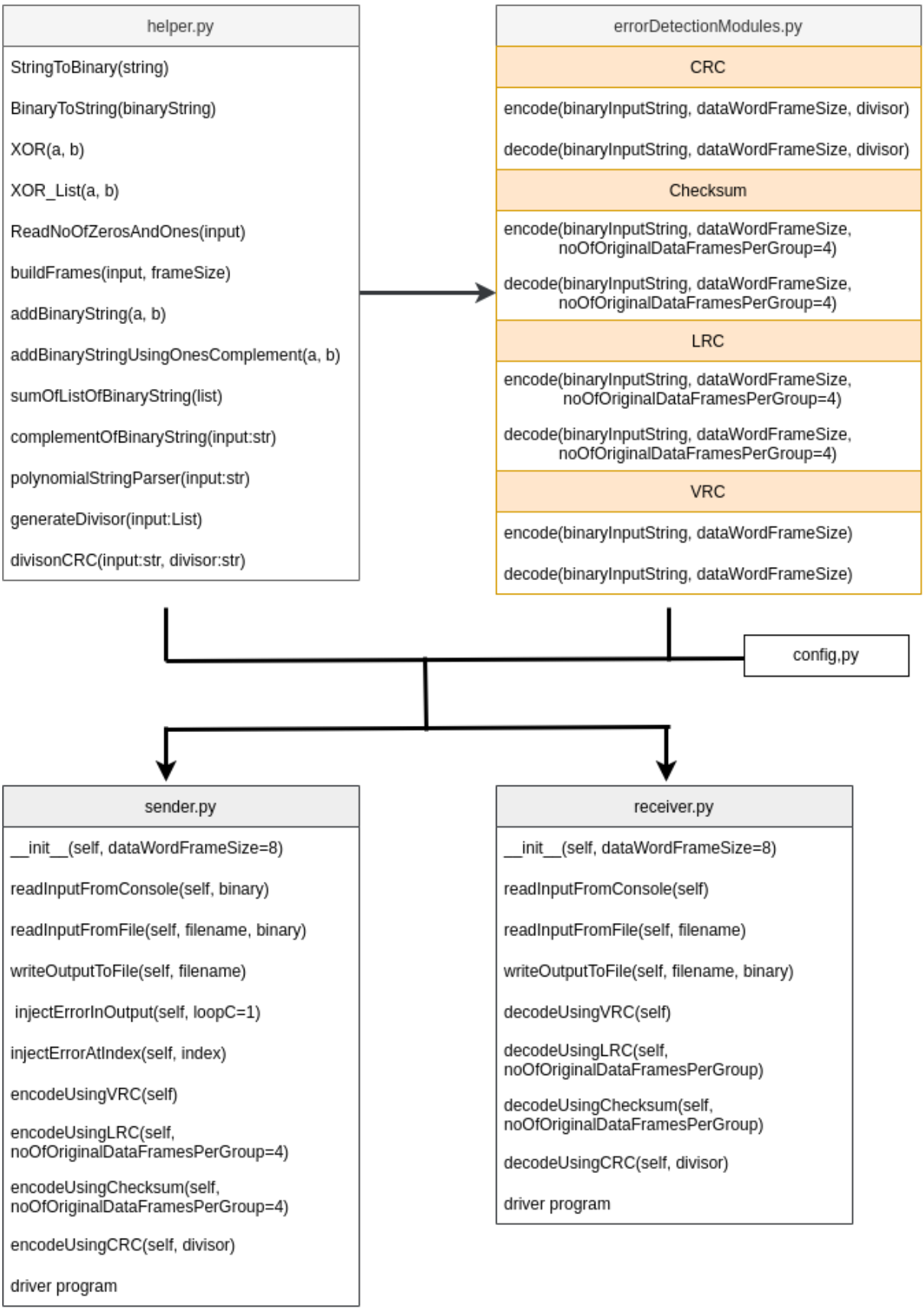
DESIGN

Purpose: Data is usually constructed by long strings of ‘1’s and ‘0’s. While traveling through a communication medium, due to electromagnetic interference, noise is introduced in data and the bits in data changed from ‘1’ to ‘0’ or vice-versa. So, on the receiver side, during the decoding of signals, the affected bits should be detected and corrected [if possible].

In this assignment, we will discuss the following error detection techniques.

1. VLC (Vertical Redundancy check)
2. LRC (Longitudinal Redundancy check)
3. Checksum (Checksum)
4. CRC (Cyclic Redundancy check)

Structure Diagram to Reflect The Procedural Organization of The Program



The error detection module is split into 5 files.

- helper.py (It has various methods, which is used by other programs at a different position)
- errorDetectionModules.py (It consists of decode and encode methods of VRC, LRC, Checksum, CRC)
- config.py (Dictionary of CRC Code and Polynomials)
- sender.py (Sender program, including a driver program to act as an interface to sender and receiver)
- receiver.py (Receiver program, included a driver program to act as an interface to sender and receiver)

Input and Output Format

1. By running the *sender.py* and following the program instructions on the screen, we can take input from the console or from the file and then select any encoding method out of VRC, CRC, LRC, and Checksum. We can opt for injecting errors at random indexes or selected indexes [manually]. In the end, the output will be stored in a file and the same will be displayed in the console.
2. By running the *receiver.py* and following the program instructions on the screen, we can take input from the console or read from the file and select the preferred decoding technique out of VRC, CRC, LRC, and Checksum. The program will decode the data. If any error occurred the frame will not be added to the output. The final data will be stored in a file and also shown to the console. We can see the error status on the console. If no error is found, we will get "*PASS*" and if the error is found, we will get "FAILED".

IMPLEMENTATION

Code snippets of *helper.py*

```
from functools import cmp_to_key
from typing import List, Tuple

def StringToBinary(string):
    return ''.join(format(ord(x), '08b') for x in string)

def BinaryToString(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0,
len(binary), 8))

def XOR(a, b):
    if a == b:
        return '0'
    return '1'

def XOR_List(a, b):
    return str(int(''.join([XOR(a[i], b[i]) for i in
range(len(a))])))

def ReadNoOfZerosAndOnes(input):
    noOfZeros = 0
    noOfOnes = 0

    for i in input:
        if i == '0':
            noOfZeros += 1
        elif i == '1':
            noOfOnes += 1

    return noOfZeros, noOfOnes

def buildFrames(input, frameSize):
    output = []
    for i in range(0, len(input), frameSize):
        output.append(input[i:i+frameSize])
    return output
```

```python
def addBinaryString(a, b):
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)
    result = ''
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = '1' + result

    return result.zfill(max_len)

def addBinaryStringUsingOnesComplement(a, b):
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)
    result = ''
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = addBinaryStringUsingOnesComplement(result, '1')
```

```
    return result.zfill(max_len)

def sumOfListOfBinaryString(l):
    result = l[0]
    for i in range(1, len(l)):
        result = addBinaryString(result, l[i])
    return result

def complementOfBinaryString(input:str):
    input = list(input)
    for i in range(len(input)):
        if input[i] == '0':
            input[i] = '1'
        else:
            input[i] = '0'
    return ''.join(input)

# Return sorted list of (power, coefficient)
# Valid polynomial format
# x^4 + x^2 + x^1 + 1
# return polynomials in form of list of (power, coefficient) in
descending order of power
def polynomialStringParser(input:str):
    # Raise exception for invalid input
    if input == '' :
        raise Exception("Invalid input")
    if input.find('x') != -1 and input.find('x^') == -1:
        raise Exception("Invalid input")

    # Main logic
    data = []

    # Iterate over list of (Splitted at the '+').removeSpaces
    for d in [j.strip() for j in input.split('+')]:
        if len(d) == 1:
            data.append((0, int(d)))
        else:
            tmp = d.split('x^')
            if len(tmp) == 1:
                data.append((int(tmp[0]), 1))
            elif len(tmp) == 2:
                if tmp[0] == '':
```

```
                    data.append((int(tmp[1]), 1))
                else:
                    data.append((int(tmp[1]), int(tmp[0])))

    # Sort the list by degree
    data = sorted(data, key=cmp_to_key(lambda item1, item2:
item2[0] - item1[0]))

    # Fill blank degress with 0 coefficients
    final_data = []

    i=0
    while i < len(data)-1:
        final_data.append(data[i])
        a = data[i][0]
        b = data[i+1][0]

        if a-b > 1:
            for j in range(b+1, a)[::-1]:
                final_data.append((j, 0))
        i+=1
    final_data.append(data[i])
    return final_data

# Accept polynomials in form of list of (power, coefficient) in
descending order of power
def generateDivisor(input:List):
    divisor = ''
    for i in input:
        divisor += ("1" if i[1] != 0 else "0")
    return divisor

# XOR List
def xor_list(a, b):

   # initialize result
   result = []

   # Traverse all bits, if bits are
   # same, then XOR is 0, else 1
   for i in range(1, len(a)):
       if a[i] == b[i]:
```

```
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)

# Divide function for CRC
def divisonCRC(input:str, divisor:str):
    input = str(int(input))
    divisor = str(int(divisor))

    # If input is longer than divisor, return input
    if len(input) < len(divisor):
        return input

    pick = len(divisor)
    tmp = input[0 : pick]

    while pick < len(input):
        if tmp[0] == '1':
            tmp = xor_list(tmp, divisor) + input[pick]
        else:
            tmp = xor_list(tmp, '0'*pick) + input[pick]

        pick += 1

    if tmp[0] == '1':
        tmp = xor_list(tmp, divisor)
    else:
        tmp = xor_list(tmp, '0'*pick)

    return str(int(tmp))
```

Method descriptions of helper.py:

- StringToBinary(string) : This method is used to convert string to binary by traversing character-by-character using the built-in ord() and format() function of python.
- BinaryToString(binaryString) : This method is used to convert long binary string [e.g 11001001…..] to readable string format [e.g. Hi …]

- XOR(a,b): This method is used to XOR two bits
- XOR_List(a,b): This method is used to XOR two strings of bits by iterating over the strings and applying XOR(a,b) function. Example: XOR_List(‘110010’, ‘001011’) will return ‘111001’
- ReadNoOfZerosAndOnes(input): This method will count no of zeros and no of ones in binary string and will return a tuple of (no of zeros, no of ones). Example : ReadNoOfZerosAndOnes(‘110010’) will return (3,3)
- buildFrames(input, frameSize): This method accept a string of ‘1’ and ‘0’ and frameSize. The method will split the string of ‘1’ and ‘0’ in frames of the length of `frameSize` & will return a list of frames. It will also add padding of ‘0’s if there is not enough length to generate the frames. Example: buildFrames(‘10111110’,4) will return [“1011”, “1110”]
- addBinaryString(a,b): This method is used to add two binary strings. Example: addBinaryString(“10101010”, “11001100”) will return “101110110”
- addBinaryStringUsingOnesComplement(a, b): This method is used to add two binary strings using 1s complement.
- sumOfListOfBinaryString(list): This method accepts a list of binary strings and returns the sum of binary strings
- complementOfBinaryString(input:str): This method takes a binary string and returns the complement of binary. complementOfBinaryString(“110010”) will return “1101”
- polynomialStringParser(input:str): This method accepts an polynomial string [like x^4 + x^2 + x^1 + 2] and return a list of tuples of (power, coefficient) in an sorted order . [(4,1),(3,0),(2,1),(1,1),(0,2)]
- generateDivisor(input:List): This method accepts list of tuples of (power, coefficient) of a polynomial string and generate the divisor. For an example, generateDivisor([(4,1),(3,0),(2,1),(1,1),(0,2)]) will return 10111
- divisonCRC(input:str, divisor:str): This method takes two arguments input and divisor. After dividing, the input [dividend] by divisor by

XORing, it will return the remainder. It is a helper function for CRC encoding and decoding.

Code Snippets of errorDetectionModules.py

```
from math import remainder
from helper import ReadNoOfZerosAndOnes,
addBinaryStringUsingOnesComplement, buildFrames,
complementOfBinaryString, divisonCRC


class VRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int):
        # ? Even Parity VRC
        output = ""
        for i in buildFrames(binaryInputString, dataWordFrameSize):
            _ , noOfOnes = ReadNoOfZerosAndOnes(i)
            output += i + ("1" if noOfOnes % 2 == 1 else "0")
        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize):
        frames = buildFrames(binaryInputString, dataWordFrameSize+1)
        errorFound = False
        output = ""
        for i in frames:
            # ? Even Parity VRC
            _ , noOfOnes = ReadNoOfZerosAndOnes(i)
            if noOfOnes % 2 == 0:
                i = i[:-1]
                output += i
            else:
                errorFound = True

        return output, errorFound

class LRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        # Check whether we can split in to equal length of frames
        noOfFramesRequiredToBeAdded = noOfOriginalDataFramesPerGroup -
```

```
len(frames)%noOfOriginalDataFramesPerGroup
        if noOfFramesRequiredToBeAdded > 0 and
noOfFramesRequiredToBeAdded != noOfOriginalDataFramesPerGroup:
            for _ in range(noOfFramesRequiredToBeAdded):
                frames.append("0"*dataWordFrameSize)

        # Iterate over the frames to calculate the parity
        for i in range(0, len(frames), noOfOriginalDataFramesPerGroup):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup]
            parity = ""
            for index in range(dataWordFrameSize)[::-1]:
                noOfOnes = 0
                # Count no of ones in frame
                for frame in tmpFrames:
                    if frame[index] == "1":
                        noOfOnes += 1
                # Add parity bit
                parity = ("1" if noOfOnes % 2 != 0 else "0") + parity


            # Append to output
            output = output + ''.join(tmpFrames) + parity

        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        # Split frames from data
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        errorFound = False
        output = ""
        # Iterate over the frames
        for i in range(0, len(frames),
noOfOriginalDataFramesPerGroup+1):
            tmp_frames = frames[i:i+noOfOriginalDataFramesPerGroup+1]
            parity_frame = tmp_frames[-1]
            frameErrorFound = False
            # Verify parity
            for index in range(dataWordFrameSize)[::-1]:
                noOfOnes = 0
                for frame in tmp_frames[:-1]:
                    noOfOnes = noOfOnes + (1 if frame[index] == '1' else
0)
                noOfOnes = noOfOnes + (1 if parity_frame[index] == '1'
else 0)
```

```
                if noOfOnes % 2 != 0:
                    frameErrorFound = True
                    break
            if frameErrorFound:
                errorFound = True
            else:
                output += ''.join(tmp_frames[:-1])

        return output, errorFound

class CheckSum:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        # Check whether we can split in to equal length of frames
        noOfFramesRequiredToBeAdded = noOfOriginalDataFramesPerGroup -
len(frames)%noOfOriginalDataFramesPerGroup
        if noOfFramesRequiredToBeAdded > 0 and
noOfFramesRequiredToBeAdded != noOfOriginalDataFramesPerGroup:
            for _ in range(noOfFramesRequiredToBeAdded):
                frames.append("0"*dataWordFrameSize)

        # Iterate over the frames to calculate the checksum
        for i in range(0, len(frames), noOfOriginalDataFramesPerGroup):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup]
            tmp = tmpFrames[0]
            frameString = ""
            for frame in tmpFrames[1:]:
                tmp = addBinaryStringUsingOnesComplement(tmp, frame)
                frameString += frame

            # Complement and get checksum
            checksum = complementOfBinaryString(tmp)
            output = output + tmpFrames[0] + frameString + checksum

        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        errorFound = False
```

```
        # Iterate over the frames to calculate the checksum and verify
        for i in range(0, len(frames),
noOfOriginalDataFramesPerGroup+1):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup+1]
            tmp = tmpFrames[0]
            frameString = ""
            for frame in tmpFrames[1:]:
                tmp = addBinaryStringUsingOnesComplement(tmp, frame)
                frameString += frame

            # Complement and get checksum
            checksum = complementOfBinaryString(tmp)
            if checksum == "0"*dataWordFrameSize:
                output += (tmpFrames[0] +
frameString[:-dataWordFrameSize])
            else:
                errorFound = True

        return output, errorFound

class CRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int,
divisor:str):
        output = ""
        crcSize = len(divisor)-1
        for i in buildFrames(binaryInputString, dataWordFrameSize):
            tmp = i+crcSize*"0" # [max degree of polynomial] times 0
            crc = divisonCRC(tmp, divisor)[:crcSize]
            output += i+crc.zfill(crcSize)
        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int,
divisor:str):
        output = ""
        errorFound = False
        crcSize = len(divisor)-1
        for i in buildFrames(binaryInputString,
dataWordFrameSize+crcSize):
            remainder = divisonCRC(i, divisor)
            if remainder == 0 or remainder == '0':
                messageData = i[:-crcSize]
                # print(messageData)
                output += messageData
            else:
```

```
                errorFound = True

        return output, errorFound
```

Classes of “errorDetectionModules.py”

1. VRC

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize: int): This method first builds data words from binaryInputString and then calculates the parity bits, and generates the codeword.
   b. decode(binaryInputString:str, dataWordFrameSize): This method splits the binaryInputString in codewords and checks the parity bits of each codeword and checks for any error.

2. LRC

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method first builds data words from binaryInputString and then calculates the parity bits and generates the frame with parity bits. Then construct the output to be sent to the receiver.
   b. decode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString into frames and checks the frame with parity bits to check for any error and then deletes the frame with parity bits, constructs the output and return.

3. Checksum

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString in frames. Then Iterate over the frames to calculate the sum of frames by one’s complement method. After

that, the sum is updated with its complement and constructs the final output.

b. decode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString in frames.Then Iterate over the frames to calculate the sum of frames by one's complement method. After that, the sum is updated with its complement. If the result is 0, then the data has no error else there may be some sort of error.

4. CRC

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize:int, divisor:str) : This function split binaryInputString in datawords. Then append no of zeros based on the CRC polynomial at the end of dataword. The using `divisonCRC` method calculates the CRC and builds the codeword.

   b. decode(binaryInputString:str, dataWordFrameSize:int, divisor:str): First it splits binaryInputString in codewords. For each codeword, it divide the codeword with the divider using `divisonCRC` method, if the remainder comes 0 , then the codeword has no error else it has some sort of error.

Code Snippets of "config.py"

```
availableCRCPolynomials = {
    "CRC_1": "x+1",
    "CRC_4_ITU"   : "x^4 + x^1 + 1",
    "CRC_5_ITU"   : "x^5 + x^4 + x^2 + 1",
    "CRC_5_USB"   : "x^5 + x^2 + 1",
    "CRC_6_ITU"   : "x^6 + x^1 + 1",
    "CRC_7"       : "x^7 + x^3 + 1",
    "CRC_8_ATM"   : "x^8 + x^2 + x^1 + 1",
    "CRC_8_CCITT" : "x^8 + x^7 + x^3 + x^2 + 1",
    "CRC_8_MAXIM" : "x^8 + x^5 + x^4 + 1",
```

```
    "CRC_8"        : "x^8 + x^7 + x^6 + x^4 + X^2 +1",
    "CRC_8_SAE"    : "x^8 + x^4 + x^3 + X^2 +1",
    "CRC_10"       : "x^10 + x^9 + x^5 + x^4 + x^1 + 1",
    "CRC_12"       : "x^12 + x^11 + x^3 + x^2 + x + 1"
}
```

config.py has not any method. It has a dictionary with the CRC code and its polynomials.

Code Snippets of “sender.py”

```
import random

from helper import StringToBinary, generateDivisor,
polynomialStringParser
from errorDetectionModules import CRC, VRC, LRC, CheckSum
from config import availableCRCPolynomials

# ? Sender Class
class Sender:
    dataWordFrameSize = 0
    input = ""
    rawInput = "" # Holds input whether it was binary or string
    output = ""
    outputWithoutAnyError = ""

    def __init__(self, dataWordFrameSize=8):
        self.dataWordFrameSize = dataWordFrameSize

    # IO Related functions
    def readInputFromConsole(self, binary):
        tmp = input("Enter the input: ")
        self.rawInput = tmp
        self.input = StringToBinary(tmp) if not binary else tmp

    def readInputFromFile(self, filename, binary):
        t = ""
        with open(filename, 'r') as f:
            for i in f.readlines():
                t += i
        self.rawInput = t.replace('\n', '')
        self.input = StringToBinary(t.replace('\n', '')) if not binary
else t.replace('\n', '')
```

```
    def writeOutputToFile(self, filename):
        if self.output == "": raise Exception("Output is empty ! Nothing
to save")
        with open(filename, 'w') as f:
            f.write(self.output)
            f.close()

    # Error Injection Function
    def injectErrorInOutput(self, loopC=1):
        for i in range(loopC):
            random_bit_location = random.randint(0, len(self.output)-1)
            self.output = self.output[:random_bit_location] +  ('0' if
self.output[random_bit_location] == '1' else '1') +
self.output[random_bit_location+1:]

    # Inject error at specific index
    def injectErrorAtIndex(self, index):
        self.output = self.output[:index] + ('0' if self.output[index]
== '1' else '1') + self.output[index+1:]

    # Encode Related Wrapper unctions
    # VRC Encoding
    def encodeUsingVRC(self):
        self.output = VRC.encode(self.input, self.dataWordFrameSize)
        self.outputWithoutAnyError = self.output

    # LRC Encoding
    def encodeUsingLRC(self, noOfOriginalDataFramesPerGroup=4):
        self.output = LRC.encode(self.input, self.dataWordFrameSize,
noOfOriginalDataFramesPerGroup)
        self.outputWithoutAnyError = self.output

    # Checksum Encoding
    def encodeUsingChecksum(self, noOfOriginalDataFramesPerGroup=4):
        self.output = CheckSum.encode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.outputWithoutAnyError = self.output

    # CRC Encoding
    def encodeUsingCRC(self, divisor):
        self.output = CRC.encode(self.input, self.dataWordFrameSize,
divisor)
        self.outputWithoutAnyError = self.output
```

```
# ? ################## DRIVER CODE ############
if __name__ == "__main__":
    # ! Data word frame size
    dataWordFrameSize = input("Enter no of bits in each data frame of
dataword [default : 8]: ")
    dataWordFrameSize =  8 if dataWordFrameSize == '' else
int(dataWordFrameSize)

    # ! Sender object
    sender = Sender(dataWordFrameSize=dataWordFrameSize)

    # ! Take input
    inputSourceAsTerminal = input("Do you want to use terminal as input
source [y/n] : ")
    inputIsBinary = input("Do you want to input binary data [y/n] : ")
    if inputSourceAsTerminal.lower() == "y":
        sender.readInputFromConsole( binary= (inputIsBinary.lower() ==
"y"))
    else:
        filename = input("Enter the filename [default :
assets/sender_input.txt ]: ")
        filename = "assets/sender_input.txt" if filename == '' else
filename
        sender.readInputFromFile(filename, binary=
(inputIsBinary.lower() == "y"))

    # ! Choose encoding method
    encodingMethod = input("Enter the encoding method [VRC, LRC, CRC,
CHECKSUM] : ")
    if encodingMethod not in ["VRC", "LRC", "CRC", "CHECKSUM"] : raise
Exception("Invalid encoding method") # Check for invalid input
    selectedPolynomial = ""

    if encodingMethod == "VRC":
        sender.encodeUsingVRC()
    elif encodingMethod == "LRC":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

sender.encodeUsingLRC(noOfOriginalDataFramesPerGroup=noOfOriginalDataFra
mesPerGroup)
    elif encodingMethod == "CHECKSUM":
```

```
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

sender.encodeUsingChecksum(noOfOriginalDataFramesPerGroup=noOfOriginalDa
taFramesPerGroup)
    elif encodingMethod == "CRC":
        print("=== Available polynomials ===")
        for i in availableCRCPolynomials:
            print(i, end=", ")
        print(end="\n")
        selectedPolynomial = input("Enter the polynomial [default :
CRC_4_ITU]: ")
        selectedPolynomial = "CRC_4_ITU" if selectedPolynomial == ''
else selectedPolynomial
        parsedPolynomial =
polynomialStringParser(input=availableCRCPolynomials[selectedPolynomial]
)
        divisor = generateDivisor(parsedPolynomial)
        sender.encodeUsingCRC(divisor=divisor)

    # ! Inject error
    if input("Do you want to inject error in output [y/n] : ").lower()
== "y":
        if input("Mnaully inject error [y/n] : ").lower() == "y":
            specificBitsToInjectError = input("Enter the specific bits
to inject error [seperate by commas] : ")
            specificBitsToInjectError = [int(i.strip()) for i in
specificBitsToInjectError.split(",")]
            for i in specificBitsToInjectError:
                sender.injectErrorAtIndex(i)
        else:
            loopC = input("Enter the no of times you want to inject
error [default : 1]: ")
            loopC = 1 if loopC == '' else int(loopC)
            sender.injectErrorInOutput(loopC=loopC)

    # ! File name to store output
    outputFileName = input("Enter the filename to store output [default
: assets/sender_output.txt ]: ")
    outputFileName = "assets/sender_output.txt" if outputFileName == ''
else outputFileName
    sender.writeOutputToFile(outputFileName)
```

```
    # ! Print data

print("========================================================================
========================")
    print("Raw Input               : ", sender.rawInput)
    print("Final Input             : ", sender.input)
    print("Encoding technique      : ", encodingMethod+"
"+selectedPolynomial if encodingMethod == "CRC" else encodingMethod)
    print("Output [Without error] : ", sender.outputWithoutAnyError),
    print("Output [May have error]: ", sender.output)
    print("Written output in file : ", outputFileName)

print("========================================================================
========================")
```

Method Descriptions of "sender.py"

- readInputFromConsole(self, binary) : Read the input from user input. If binary is true, it will accept binary data [100110101….], else it will accept any string and later will convert it to binary.
- readInputFromFile(self, filename, binary) : Read the input from file. If binary is true, it will accept binary data [100110101….], else it will accept any string and later will convert it to binary.
- writeOutputToFile(self, filename): This method will write the output to a file.
- injectErrorInOutput(self, loopC=1) : This method will inject error at random position of data. The loopC parameter signifies no fo times this process will repeat.
- injectErrorAtIndex(self, index) : By this method , we can inject an error in the data at a selected index.
- encodeUsingVRC(self) : This is a wrapper method around the encode method of VRC.
- encodeUsingLRC(self, noOfOriginalDataFramesPerGroup=4) : This is a wrapper method around the encode method of LRC.

- encodeUsingChecksum(self, noOfOriginalDataFramesPerGroup=4) : This is a wrapper method around the encode method of Checksum.
- encodeUsingCRC(self, divisor) : This is a wrapper method around the encode method of CRC.
- Driver Program : This program is responsible to show the desired options , instructions and results to the user and to test the Sender class.

Code Snippets of “receiver.py”

```
from helper import BinaryToString, ReadNoOfZerosAndOnes, buildFrames,
generateDivisor, polynomialStringParser
from errorDetectionModules import CRC, VRC, LRC, CheckSum
from config import availableCRCPolynomials

class Receiver:
    dataWordFrameSize=0
    input = ""
    output = ""
    outputData = ""
    errorFound = False

    def __init__(self, dataWordFrameSize=8):
        self.dataWordFrameSize = dataWordFrameSize

    # IO Related functions
    def readInputFromConsole(self):
        self.input = input("Enter the input in binary format: ")

    def readInputFromFile(self, filename):
        t = ""
        with open(filename, 'r') as f:
            for i in f.readlines():
                t += i
        self.input = t.replace('\n', '')

    def writeOutputToFile(self, filename, binary):
        tmp = BinaryToString(self.output) if not binary else self.output
        with open(filename, 'w') as f:
            f.write(tmp)
            f.close()
```

```
    # Decode Wrapper Methods
    # VRC Decoding
    def decodeUsingVRC(self):
        output, errorFound = VRC.decode(self.input,
self.dataWordFrameSize)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # LRC Decoding
    def decodeUsingLRC(self, noOfOriginalDataFramesPerGroup):
        output, errorFound = LRC.decode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # Checksum Decoding
    def decodeUsingChecksum(self, noOfOriginalDataFramesPerGroup):
        output, errorFound = CheckSum.decode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # CRC Decoding
    def decodeUsingCRC(self, divisor):
        output, errorFound = CRC.decode(self.input,
self.dataWordFrameSize, divisor)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound


if __name__ == "__main__":
    # ! Data word frame size
    dataWordFrameSize = input("Enter no of bits in each data frame of
dataword [default : 8]: ")
    dataWordFrameSize =  8 if dataWordFrameSize == '' else
int(dataWordFrameSize)

    # ! Receiver object
    receiver = Receiver(dataWordFrameSize=dataWordFrameSize)

    # ! Input for receiver
    if input("Do you want to read input from console? (y/n): ").lower()
```

```
== "y":
        receiver.readInputFromConsole()
    else:
        inputfilename = input("Enter the filename [default :
assets/sender_output.txt]: ")
        inputfilename = "assets/sender_output.txt" if inputfilename ==
'' else inputfilename
        receiver.readInputFromFile(inputfilename)


    # ! Choose decoding method
    decodingMethod = input("Enter the decoding method [VRC, LRC, CRC,
CHECKSUM] : ")
    if decodingMethod not in ["VRC", "LRC", "CRC", "CHECKSUM"] : raise
Exception("Invalid encoding method") # Check for invalid input
    selectedPolynomial = ""

    # ! Decode
    if decodingMethod == "VRC":
        receiver.decodeUsingVRC()
    elif decodingMethod == "LRC":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

receiver.decodeUsingLRC(noOfOriginalDataFramesPerGroup=noOfOriginalDataF
ramesPerGroup)
    elif decodingMethod == "CHECKSUM":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

receiver.decodeUsingChecksum(noOfOriginalDataFramesPerGroup=noOfOriginal
DataFramesPerGroup)
    elif decodingMethod == "CRC":
        print("=== Available polynomials ===")
        for i in availableCRCPolynomials:
            print(i, end=", ")
        print(end="\n")
        selectedPolynomial = input("Enter the polynomial [default :
CRC_4_ITU]: ")
        selectedPolynomial = "CRC_4_ITU" if selectedPolynomial == ''
```

```
else selectedPolynomial
        parsedPolynomial =
polynomialStringParser(input=availableCRCPolynomials[selectedPolynomial]
)
        divisor = generateDivisor(parsedPolynomial)
        receiver.decodeUsingCRC(divisor=divisor)

    # ! Write output to file
    outputFilename = input("Enter the filename [default :
assets/receiver_output.txt]: ")
    outputFilename = "assets/receiver_output.txt" if outputFilename ==
'' else outputFilename
    tmpFilename = outputFilename.split(".")[0]

    receiver.writeOutputToFile(filename=tmpFilename+"_binary.txt",
binary=True)
    receiver.writeOutputToFile(filename=tmpFilename+"_data.txt",
binary=False)

    # ! Print data

print("================================================================
========================")
    print("Input                  : ", receiver.input)
    print("Decoding technique     : ", decodingMethod+"
"+selectedPolynomial if decodingMethod == "CRC" else decodingMethod)
    print("Output Binary          : ", receiver.output)
    print("Output Data            : ", receiver.outputData)
    print("Status                 : ", "FAILED" if receiver.errorFound
else "PASS")
    print("Written binary output in file : ", tmpFilename+"_binary.txt")
    print("Written binary output in file : ", tmpFilename+"_data.txt")

print("================================================================
========================")
```

Method description of “receiver.py”

- readInputFromConsole(self) : Read the input from user input.
- readInputFromFile(self, filename) : Read the input from file.
- writeOutputToFile(self, filename, binary): This method will write the output to a file. If the binary is true , then it will store the binary data in the file, otherwise, it will convert the binary to string and then store in file

- decodeUsingVRC(self) : This is a wrapper method around the decode method of VRC.
- decodeUsingLRC(self, noOfOriginalDataFramesPerGroup) : This is a wrapper method around the decode method of LRC.
- decodeUsingChecksum(self, noOfOriginalDataFramesPerGroup) : This is a wrapper method around the decode method of Checksum.
- decodeUsingCRC(self, divisor) : This is a wrapper method around the decode method of CRC.
- Driver Program: This program is responsible to show the desired options, instructions and results to the user and testing the Receiver class.

RESULTS

=> *Here are some screenshots of tests on VRC*

Testing with no error

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011101111001001000000101001101100001011100100110101101100001011100 10
Encoding technique     :  VRC
Output [Without error] :  01010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Output [May have error]:  01010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Decoding technique     :  VRC
Output Binary          :  01010100011000010110111001101101011011101111001001000000101001101100001011100100110101101100001011100 10
Output Data            :  Tanmoy Sarkar
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Testing with error [manually injected]

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 1
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011101111001001000000101001101100001011100100110101101100001011100 10
Encoding technique     :  VRC
Output [Without error] :  01010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Output [May have error]:  00010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  00010100101100001101101110101101101101110011110011001000001010100110011000011011100100011010111011000011011100100
Decoding technique     :  VRC
Output Binary          :  01100001011011100110110101101111011111001001000000101001101100001011100100110101101100001011100 10
Output Data            :  anmoy Sarkar
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

=> Here are some screenshots of tests on LRC

Testing with no error

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101101111011110010010000001010011011000010111001001101011011000010111010
Encoding technique     :  LRC
Output [Without error] :  01010100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Output [May have error]:  01010100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01010100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Decoding technique     :  LRC
Output Binary          :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010000000000000000000000000
Output Data            :  Tanmoy Sarkar
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Testing with error [manually injected]

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 2
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101101111011110010010000001010011011000010111001001101011011000010111010
Encoding technique     :  LRC
Output [Without error] :  01010100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Output [May have error]:  01110100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01110100011000010110111001101101001101100110111101111001001000000101001101100101011000010111001001101011011000010001100101110010
00000000000000000000000000001110010
Decoding technique     :  LRC
Output Binary          :  01101111011111001001000000101001101100001011100100110101101101100001011100100000000000000000000000000
Output Data            :  oy Sarkar
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

=> Here are some screenshots of tests on Checksum

Testing with no error

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011110111001001000000101001101100001011100100110101101100000101110010
Encoding technique     :  CHECKSUM
Output [Without error] :  01010100011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Output [May have error]:  01010100011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01010100011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Decoding technique     :  CHECKSUM
Output Binary          :  01010100011000010110111001101101011011110111001001000000101001101100001011100100110101101100000101110010000000000000000000000000
Output Data            :  Tanmoy Sarkar
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Testing with error [manually injected]

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 5
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011110111001001000000101001101100001011100100110101101100000101110010
Encoding technique     :  CHECKSUM
Output [Without error] :  01010100011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Output [May have error]:  01010000011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01010000011000010110111001101101011011100110111101111001001000000101001110100011011000010111001001101011011000010101111101110010
00000000000000000000000000010001101
Decoding technique     :  CHECKSUM
Output Binary          :  01101111011111001001000000101001101100001011100100110101101100000101110010000000000000000000000000
Output Data            :  oy Sarkar
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

=> Here are some screenshots of tests on CRC

Testing with no error

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Hi bro
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Hi bro
Final Input            :  01001000011010010010000001100010011100100l101111
Encoding technique     :  CRC CRC_4_ITU
Output [Without error] :  01001000010101110100101010010000000110110001010110111001001110110111111111
Output [May have error]:  01001000010101110100101010010000000110110001010110111001001110110111111111
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  01001000010101110100101010010000000110110001010110111001001110110111111111
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  01001000011010010010000001100010011100100l101111
Output Data            :  Hi bro
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Testing with error [manually injected]

Encode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Hi bro
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 1,3
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  Hi bro
Final Input            :  01001000011010010010000001100010011100100l101111
Encoding technique     :  CRC CRC_4_ITU
Output [Without error] :  01001000010101110100101010010000000110110001010110111001001110110111111111
Output [May have error]:  00011000010101110100101010010000000110110001010110111001001110110111111111
Written output in file :  assets/sender_output.txt
================================================================================
```

Decode

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  00011000010101110100101010010000000110110001010110111001001110110111111111
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  01101001001000000110001001110010011011111
Output Data            :  i bro
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

TEST CASES

a) Error is detected by checksum but not by CRC

Dataword -> 10 10 00 00

Insert error at index -> 3, 6, 7

Module used: CRC 4 ITU

Sender Side

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input             :  10100000
Final Input           :  10100000
Encoding technique    :  CRC CRC_4_ITU
Output [Without error] :  101000000100
Output [May have error]:  101100110100
Written output in file :  assets/sender_output.txt
================================================================================
```

Receiver Side

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                 :  101100110100
Decoding technique    :  CRC CRC_4_ITU
Output Binary         :  10110011
Output Data           :  ³
Status                :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Note : Error Injected but Receiver side can’t detect the error by CRC 4 ITU

Module used: Checksum

Sender Side

```
Enter no of bits in each data frame of dataword [default : 8]: 2
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input             :  10100000
Final Input           :  10100000
Encoding technique    :  CHECKSUM
Output [Without error] :  1010000010
Output [May have error]:  1011001110
Written output in file :  assets/sender_output.txt
================================================================================
```

Receiver Side:

```
Enter no of bits in each data frame of dataword [default : 8]: 2
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  1011001110
Decoding technique     :  CHECKSUM
Output Binary          :
Output Data            :
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Note : Checksum has detected the error in data

b) <u>Error is detected by VRC but not by CRC</u>

Dataword -> 10 10 00 00

Insert error at index -> 3, 6, 7

Module used: CRC 4 ITU

Sender Side

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  10100000
Final Input            :  10100000
Encoding technique     :  CRC CRC_4_ITU
Output [Without error] :  101000000100
Output [May have error]:  101100110100
Written output in file :  assets/sender_output.txt
================================================================================
```

Receiver Side

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                  :  101100110100
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  10110011
Output Data            :  ³
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Note : Error Injected but Receiver side can’t detect the error by CRC 4 ITU

Module used: VRC

Sender Side

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
================================================================================
Raw Input              :  10100000
Final Input            :  10100000
Encoding technique     :  VRC
Output [Without error] :  101000000
Output [May have error]:  101100110
Written output in file :  assets/sender_output.txt
================================================================================
```

Receiver Side:

```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
================================================================================
Input                :  101100110
Decoding technique   :  VRC
Output Binary        :
Output Data          :
Status               :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
================================================================================
```

Note : VRC has detected the error in data

ANALYSIS

Vertical Redundancy Check (VRC):

1. This scheme detects all single bit errors. Further, it detects all multiple errors as long as the number of bits corrupted is odd (referred to as odd bit errors). Suppose the message to be transmitted is

   10111 10001 10010

   and the message received at the receiver is

   10011 10001 10010

2. 0 represents that this bit is in error. For the above example, when the receiver performs the parity check, it detects that there was an error in the first nibble during transmission as there is a mismatch between the parity bit and the data in the nibble. However, the receiver does not know which bit in that nibble is in error. Similarly, the following error is also detected by the receiver.

   The message transmitted is

   10111 10001 10010

   and the message received at the receiver is

01111 10011 10010

Three bits are corrupted by the transmission medium and this is detected by the receiver as there is a mismatch between the 2nd nibble and the 2nd parity bit. It is important to note that the error in the first nibble is unnoticed as there is no mismatch between the data and the parity bit.

3. The above example also suggests that not all even bit errors (multiple bit errors with the number of bits corrupted even) are detected by this scheme. If an even bit error is such that the even number of bits are corrupted in each nibble, then such error is unnoticed by the receiver. However, if an even bit is such that at least one of the nibbles has an odd number of bits in error, then a such error is detected by VRC.

Longitudinal Redundancy Check (LRC):

1. Similar to VRC, LRC detects all single bit and odd bit errors. Some even bit errors are detected and the rest is unnoticed by the receiver.
2. The following error is detected by LRC but not by VRC. For the message 1011 1000 1001, supoose the received message is

```
1101
1000
1001
————
1010
```

3. In the above example, there is a mismatch between the number of 1's and the parity bit in Columns 2 and 3.
4. The following error is detected by VRC but not by LRC. For the message 1011 1000 1001, suppose the received message is

```
1100
1001
1001
————-
1010
```

The above error is unnoticed by the receiver if we follow VRC scheme whereas it is detected by LRC as illustrated below.

10101 10011 10010

5. Here again, not all even bit errors are detected by this scheme. If the error is such that each column has even number of bits in error, then such error is undetected. However, if the distribution is such that at least one column contains an odd number of bits in error, then such errors are always detected at the receiver.

Checksum:

1. If multiple bit error is such that in each column, a bit '0' is flipped to bit '1', then such an error is undetected by this scheme. Essentially, the message received at the receiver has lost the value 11111 with respect to the sum. Although, it loses this value, this error is unnoticed at the receiver.
2. Also, multiple bit error is such that the difference between the sum of the sender's data and the sum ofthe receiver's data is 1111, then this error is unnoticed by the receiver.
3. The above two errors are undetected by the receiver due to the following interesting observations.
4. 4. For any binary data a such that a != 0000, the value of a + (1111) in 1's complement arithmetic is a. On the similar line, if a1,...,an are nibbles, then a1+...+ak+(1111) = a1+...+ak This is true because, a + 1 1 1 1 gives a – 1 with a carry '1' and inturn this '1' is added with a – 1 as part of 1's complement addition, yielding a.
5. Consider the scenario in which the sender transmits a1,...,ak along with the checksum and the transmission line corrupts multiple bits due to which we lose 1 1 1 1 on the sum. Although, the receiver received a1+...+ak-(1 1 1 1), it follows from the above observation that a1+...+ak-(1 1 1 1) is still a1+...+ak, thus the error is undetected.
6. Similar to other error detection schemes, checksum detects all odd bit errors and most of even bit errors.

Cyclic Redundancy Check (CRC):

1. The answer to whether CRC detects all errors depends on the divisor polynomial used as a part of CRC computation. Consider the divisor polynomial x^2 which corresponds to 100 and the message to be transmitted is 101010. After appending CRC bits (in this case 0 0) we get 10101000. Assuming during the data transmission, the 3rd bit is in error and the message received at the receiver is 10101100. On performing CRC check on 10101100, we see that the remainder is zero and the receiver wrongly concludes that there is no error in transmission. The reason this error is undetected by the receiver is that the message received is perfectly divisible by the divisor 1 0 0. More appropriately, if we visualize the received message as the xor of the original message and the error polynomial, then we see that the error polynomial is divisible by 100.
2. Message received = 10101100

   Message received = message transmitted + error polynomial, i.e.,

   10101100 = 10101000 + 00000100

   Clearly both are divisible by the divisor 100. In general, if the error polynomial is divisible by the divisor, then such errors are undetected by the receiver. The receiver wrongly concludes that there is no error in transmission. For example, if the error polynomial is 00001100 (3^{rd} and 4^{th} bits in the message in error), then the receiver is unaware of this error. However, if the divisor is 101, then both errors are detected by the receiver. Thus, choosing an appropriate divisor is crucial in detecting errors at the receiver if any during the transmission.

COMMENTS

The assignment has helped me to understand more about the error detection methods in-depth and gave me an in-hand experience and idea about real-world usage in networking by implementing those in python. I am looking

forward to implement socket based communication between the sender and receiver

Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 2
Subject: Computer Network
Group: A1

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

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

Hints: Some points you may consider in your design.

Following functions may be required in Sender.

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

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

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

PRE-REQUISITES

As this protocol required communication between the client and server and also required a channel to simulate injecting error like real world, we need to build a program based on socket to make that easy to implement the protocols given in the assignment.

SocketServer - *Capable of handling multiple clients and support messages passing to one client to another by maintaining a connection pool*

```
import socket
from _thread import *

# Purpose
# 1. Manage connection pool
# 2. Manage client requests
# 3. Bi-directional communication
# -> Sender can send data to Receiver
# -> Receiver can send data to Sender

class SocketServer:

    clientConnectionInstances = {}
    senderReceiver = {}
    # map of senderIp:senderPort to receiverIp:receiverPort
    # map of receiverIp:receiverPort to senderIp:senderPort

    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.socket = socket.socket()

    def __repr__(self) -> str:
        return f"SocketServer({self.host}, {self.port})"

    def start(self):
        try:
            self.socket.bind((self.host, self.port))
        except socket.error as e:
            print(str(e))
        print('Server is online ...')
        self.socket.listen(5)

    def startAcceptConnections(self):
        while True:
```

```
            client, address = self.socket.accept()
            # Store client connection instance in dictionary
            SocketServer.clientConnectionInstances[f"{address[0]}:{address[1]}"] =
client
            print('Connected to: ' + address[0] + ':' + str(address[1]))
            # Start a new thread to handle client requests
            start_new_thread(SocketServer.handleClient, (address[0],
str(address[1])))

    @staticmethod
    def handleClient(clientIp, clientPort):
        address = f"{clientIp}:{clientPort}"
        # Get client connection instance from dictionary
        client = SocketServer.clientConnectionInstances[address]

        # Start listening for client requests
        while True:
            try:
                data = client.recv(1024)
                # Check whether client sent data
                if not data:
                    break
                data = data.decode('utf-8')
                # Check for special command to connect to receiver
                if str(data).startswith("connect:"):
                    splittedData = data.split(":")
                    if(len(splittedData) == 3):
                        receiverAddress = f"{splittedData[1]}:{splittedData[2]}"
                        # Check whether receiver address found
                        if receiverAddress in SocketServer.clientConnectionInstances:
                            # Check if receiver address is found in senderReceiver
map
                            if receiverAddress in SocketServer.senderReceiver or
address in SocketServer.senderReceiver:
                                print("Already connected to receiver")
                            else:
                                # Store in sender receiver map
                                SocketServer.senderReceiver[address] =
receiverAddress
                                SocketServer.senderReceiver[receiverAddress] =
address
                                print(f"Setup done ! Sender {address} is connected to
receiver {receiverAddress}")
                        else:
                            print(f"Receiver {receiverAddress} not found")
                    else:
                        continue
                # Else consider as normal data
                else:
```

```
                    # Get receiver address from senderReceiver map
                    receiverAddress = None
                    if address in SocketServer.senderReceiver:
                        receiverAddress = SocketServer.senderReceiver[address]
                    else:
                        print(f"No receiver found for sender {address}")
                        continue
                    # Check whether receiver instance found
                    receiverInstance =
SocketServer.clientConnectionInstances[receiverAddress]
                    if receiverInstance is None:
                        print(f"Receiver {receiverAddress} not found")
                        continue
                    # Send data to receiver

receiverInstance.sendall(str.encode(SocketServer.modifyData(data)))
                    print(f"{address} ---> {receiverAddress} --> {data}")

            except (BrokenPipeError, ConnectionResetError, OSError):
                    print(f"Receiver {receiverAddress} is offline")
                    break
            except Exception as e:
                print(str(e))
                break

        client.close()

        if address in SocketServer.clientConnectionInstances:
            del SocketServer.clientConnectionInstances[address]

        if address in SocketServer.senderReceiver:
            # Get receiver address from senderReceiver map
            receiverAddress = SocketServer.senderReceiver[address]
            # Get receiver instance from clientConnectionInstances map
            if receiverAddress in  SocketServer.clientConnectionInstances:
                receiverInstance =
SocketServer.clientConnectionInstances[receiverAddress]
                receiverInstance.send(str.encode(f"disconnect:"))
                # Close receiver instance
                receiverInstance.close()
                del SocketServer.clientConnectionInstances[receiverAddress]
                del SocketServer.senderReceiver[receiverAddress]
            # Delete own address from senderReceiver map
            del SocketServer.senderReceiver[address]

        print(f"Client {address} has disconnected")
```

```
    @staticmethod
    def modifyData(data):
        # An method to be overridden by child class
        return data

    def closeAllConnections(self):
        self.socket.close()
        print("Server is offline")

# Example Code
# tmp = SocketServer(host='127.0.0.1', port=8081)
# tmp.start()
# tmp.startAcceptConnections()
```

Client - Base class to implement a client that can communicate with the server easily and send and receive messages to/from another client. It has some abstract methods which we need to implement that's different for each protocol.

```
from abc import abstractmethod
import socket
import threading
import time


class Client:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.killed = False
        self.condition = threading.Condition()

    def connectToServer(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.sock.connect((self.host, self.port))
            print('Connected to server')
        except socket.error as e:
            print(str(e))

    def startProcess(self):
        self.setupBeforeProcess()
        self.senderThread = threading.Thread(target=self.sendData)
        self.receiverThread = threading.Thread(target=self.receiveData)
```

```
        self.receiverThread.start()
        self.senderThread.start()

    @abstractmethod
    def sendData(self):
        raise NotImplementedError

    @abstractmethod
    def receiveData(self):
        raise NotImplementedError

    @abstractmethod
    def setupBeforeProcess(self):
        raise NotImplementedError

    def closeConnection(self):
        if self.killed : return
        self.killed = True
        print('Closing connection')
        self.condition.acquire()
        self.condition.notifyAll()
        self.condition.release()

        try:
            self.sock.close()
        except socket.error as e:
            print(str(e))
        print('Connection closed')


# client = Client('127.0.0.1',8081)
# client.connectToServer()
# client.closeConnection()
```

Channel - *it inherited the socket server class and added the error injecting and random delay logics in it.*

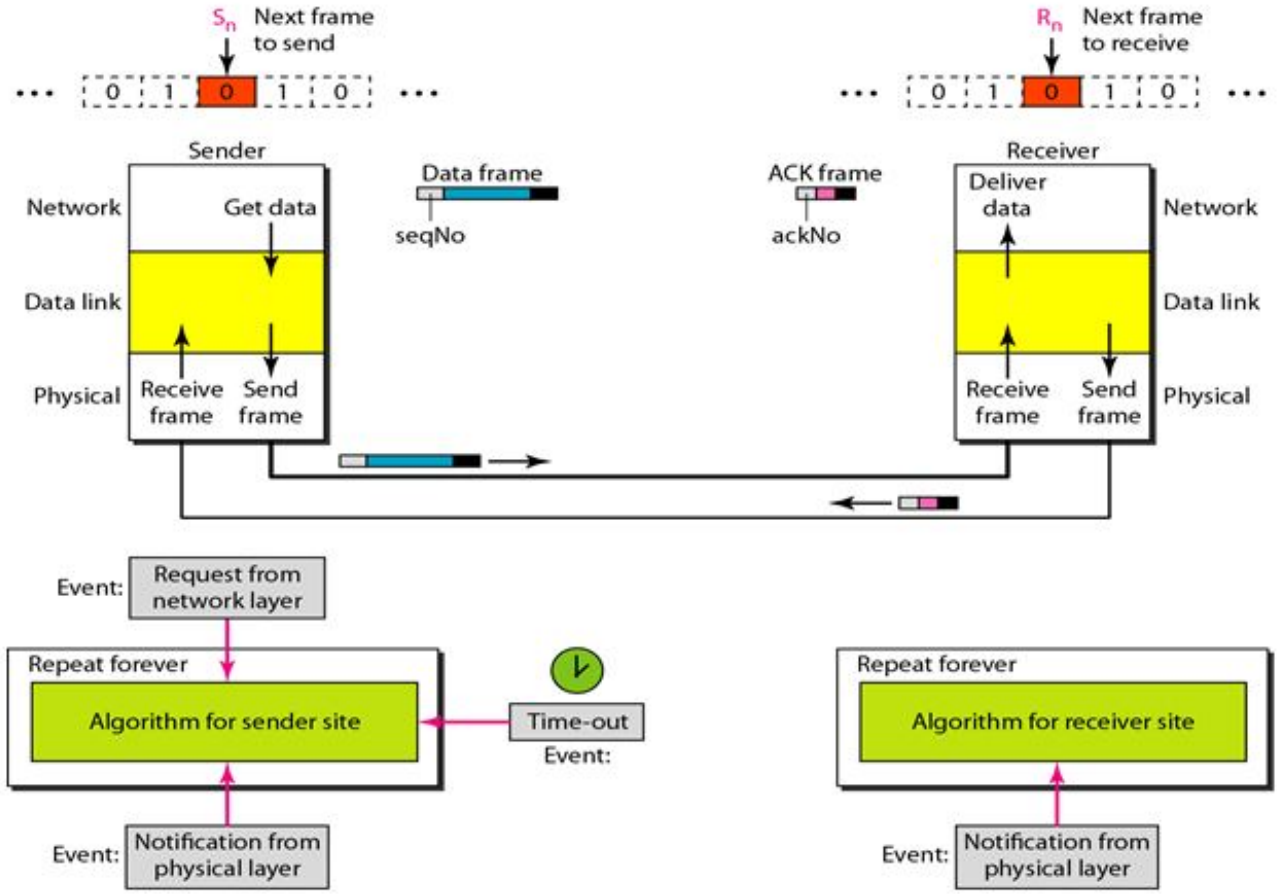
```
import time
from clientServer.socketServer import SocketServer
import random

class Channel(SocketServer):
```

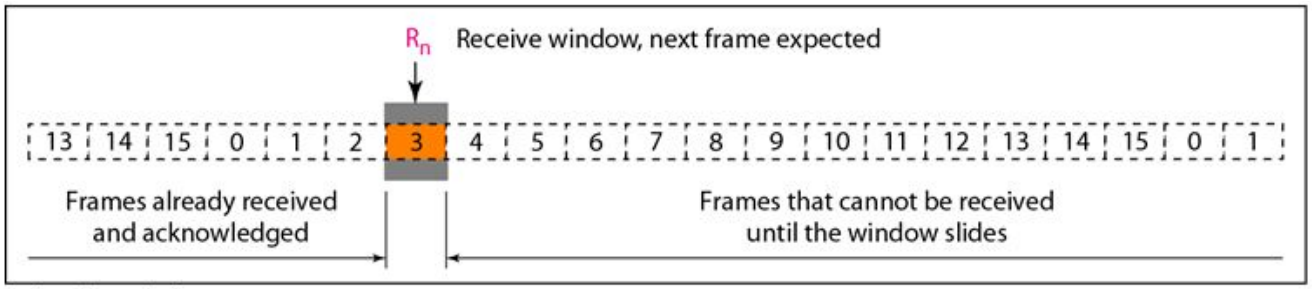
```
    errorCount = 0
    def __init__(self, host, port):
        super().__init__(host, port)
        # Update modifyData static method of  SocketServer with  Channel's
        SocketServer.modifyData = Channel.modifyData

    @staticmethod
    def modifyData(data):
        # return data
        # return data
        if data in ["disconnect:"]:
            return data
        return Channel.injectErrorInData(data, loopC=1)

    @staticmethod
    def injectErrorInData(data, loopC=5):
        # time.sleep(random.random())
        olddata = data
        for _ in range(loopC):
            random_bit_location = random.randint(0, len(data)-1)
            should_inject_error = Channel.errorCount % 17 == 0
            Channel.errorCount += 1
            if should_inject_error:
                data = data[:random_bit_location] +  ("0" if
data[random_bit_location] == "1" else "1") + data[random_bit_location+1:]
        if olddata != data:
            print("Injected Error : "+olddata+" ---> "+data)
        return data




if __name__ == "__main__":
    server = Channel(host='127.0.0.1', port=8081)
    try:
        print("Socket Server[Channel] is starting....")
        server.start()
        server.startAcceptConnections()
    except KeyboardInterrupt:
        server.closeAllConnections()
        exit()
    except Exception as e:
        print(str(e))
        server.closeAllConnections()
```

```
exit()
```

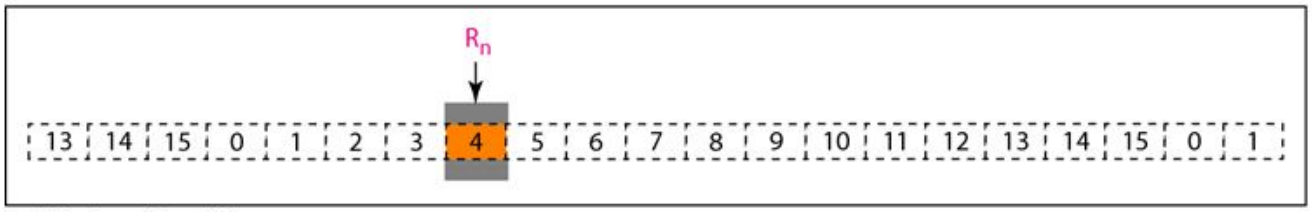
Error Detection Modules: We are using the error modules developed in the previous assignment for this assignment, for that reason we are going into detail about those.

DESIGN OF STOP AND WAIT PROTOCOL





a. Receive window



b. Window after sliding

Code -

Sender side

```
import socket
import threading
```

```
from time import sleep

from helpers import buildFrames, decodeData, encodeData


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [sn][parity]
# [ 4 ][  4   ]

class Sender:
    def __init__(self, host, port, timeout):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.ackReceivedEvent = threading.Event()
        self.ackReceivedEvent.clear()

        self.data = []
        self.frameIndex = 0
        self.sn = 0
        self.snmax = 1

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if len(receiver_ip.strip()) == 0:
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")

self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()
```

```
    def send(self):
        while True:
            # Check frames
            if self.frameIndex >= len(self.data):
                print("No more frames to send")
                break
            # Prepare fram e-- by taking the sn and the element of window
            frame =
encodeData(str(bin(self.sn)[2:]).zfill(2)+self.data[self.frameIndex])
            print("[SEND] new frame ", frame, " with sn ", self.sn)
            # Increase seq no
            self.increaseSn()
            # Icrease frame index
            self.frameIndex += 1
            # Send a frame
            self.sock.sendall(str.encode(frame))
            # Wait for ack with a timeout
            self.ackReceivedEvent.clear()
            isNotified = self.ackReceivedEvent.wait(timeout=self.timeout)
            if not isNotified:
                print("[RESEND] resending frame ", frame, " with sn ",
self.sn)
                self.frameIndex -= 1
                self.decreaseSn()

            # If timeout, resend the frame -- run the loop again


    def recvAck(self):
        while True:
            # Listen for data
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
                seqNo = int(decodedData[1], 2)
                # If ack sn == current sn, just emit ackreceived event
                if seqNo != self.sn:
                    # Else ack sn != current sn, set sn to ack sn , emit
ackreceived event
                    self.sn = seqNo
                    # Set frame index to previous frame index
                    self.frameIndex = max(self.frameIndex-1, 0)
```

```
                    print("[ACK] ack received with sn ", seqNo)
                else:
                    print("[ACK] ack discarded as rn not macthed")
                # Emit ackreceived event
                self.ackReceivedEvent.set()
            else:
                print("[DISCARD] discarding ACK due to error")


    def increaseSn(self):
        self.sn += 1
        if self.sn > self.snmax:
            self.sn = 0

    def decreaseSn(self):
        self.sn -= 1
        self.sn = max(self.sn, 0)

    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.append(i)



sender = Sender('127.0.0.1', 8081, 4)
sender.pushData("01101000011001010110110101101100011011110111010101111001011100011")

sender.startProcess()
```

Receiver Side-

```
import socket
import threading

from helpers import decodeData, generateACK


class Receiver:
    def __init__(self, host, port):
        self.host = host
        self.port = port
```

```
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.frameReceived = threading.Event()
        self.frameReceived.clear()

        self.data = []
        self.rn = 0
        self.rnmax = 1


    def startProcess(self):
        self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve data
            data = self.sock.recv(1024)
            if data == "disconnect:" : break
            data = data.decode()
            # Decode data
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # If seq no == rn, save data and send ack for next frame
                if seqNo == self.rn:
                    self.data.append(frame)
                    self.increaseRn()
                    self.frameReceived.set()
                    self.printData()
                    print("[ACCEPT] Frame received ")
                # If seq no != rn, discard and send ack fagain
                else:
                    print("[DISCARD] seqNo not matched to rn")
                    self.frameReceived.set()
            else:
                print("[DISCARD] frame due to error")
```

```
    def sendACK(self):
        while True:
            # Wait for frame received event
            self.frameReceived.wait()
            print("[ACK] Sending ACK for frame ", self.rn)
            # Send ack for the frame
            self.sock.sendall(str.encode(generateACK(self.rn,
with_parity=True)))
            # Clear frame received event
            self.frameReceived.clear()

    def increaseRn(self):
        self.rn += 1
        if self.rn > self.rnmax:
            self.rn = 0

    def printData(self):
        print("Data : ", end="", flush=True)
        for i in self.data:
            print(i, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

Output -

Channel -

```
Server is online ...
Connected to: 127.0.0.1:34648
Connected to: 127.0.0.1:47108
Connected to: 127.0.0.1:39992
Setup done ! Sender 127.0.0.1:39992 is connected to receiver 127.0.0.1:34648
Injected Error : 00011010000110 ---> 00011011000110
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011010000110
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011010000110
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011001011101
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011011011001
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011011000101
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011011111111
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011101011000
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 00011110010000
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00010011
127.0.0.1:39992 ---> 127.0.0.1:34648 --> 01011100110010
127.0.0.1:34648 ---> 127.0.0.1:39992 --> 00000000
```

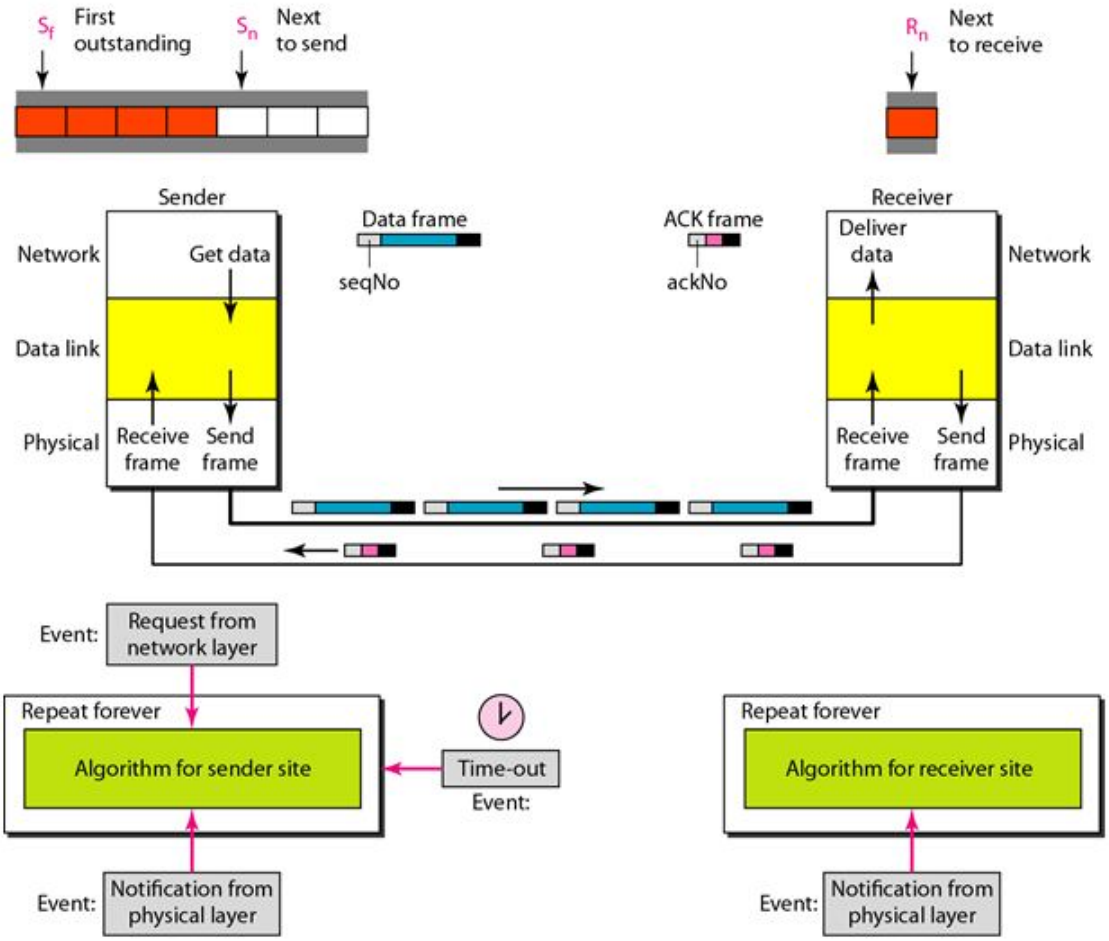
Sender -

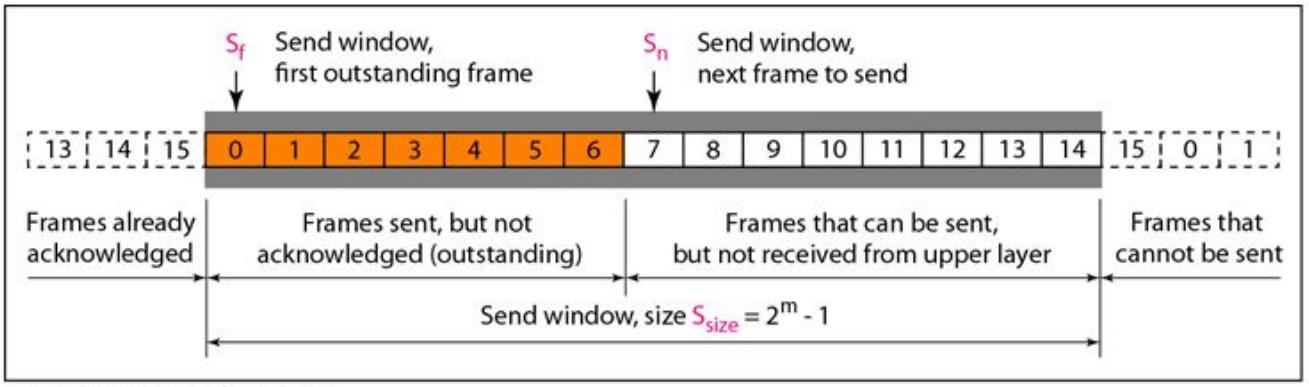
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sw-sender.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 34648
[SEND] new frame  00011010000110  with sn  0
[RESEND] resending frame  00011010000110  with sn  1
[SEND] new frame  00011010000110  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011001011101  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011011011001  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011011000101  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011011111111  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011101011000  with sn  1
[ACK] ack discarded as rn not macthed
[SEND] new frame  00011110010000  with sn  0
[ACK] ack discarded as rn not macthed
[SEND] new frame  01011100110010  with sn  1
[ACK] ack discarded as rn not macthed
No more frames to send
```

Receiver -

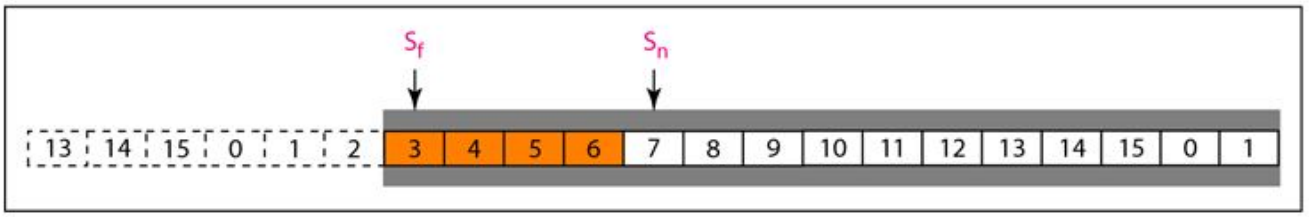
```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sw-receiver.py
[DISCARD] frame due to error
Data : 01101000
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : [ACK] Sending ACK for frame  0
0110100001100101
res 00000000
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  1
01101000res 00010011
0110010101101101
[ACCEPT] Frame received
Data : 01101000[ACK] Sending ACK for frame  0
011001010110110101101100
[ACCEPT] Frame received
res 00000000
Data : [ACK] Sending ACK for frame  1
0110100001100101res 00010011
011011010110110001101111
[ACCEPT] Frame received
Data : 011010000110010101101101011011000[ACK] Sending ACK for frame  0
0110111101110101
[ACCEPT] Frame received
res 00000000
Data : 011010000110010101101101011011000110111101110101010111001
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : 011010000110010101101101011011000110111101110101010111001[ACK] Sending ACK for frame  0
01110011
[ACCEPT] Frame received
res 00000000
```

DESIGN OF GO-BACK-N PROTOCOL





a. Send window before sliding



b. Send window after sliding

Code -

Sender Code -

```
from cmath import sqrt
import socket
import threading
from time import sleep

from helpers import buildFrames, decodeData, encodeData


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [sn][parity]
# [ 4 ][  4   ]

class Sender:
    def __init__(self, host, port, timeout, N=4):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.ackReceivedEvent = threading.Event()
        self.ackReceivedEvent.clear()

        self.N = N

        self.data = []
        self.frameIndex = 0
        self.sn = 0

        tmp = int(sqrt(N).real)
        self.snmax = int('1'*tmp, 2)

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if len(receiver_ip.strip()) == 0:
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")
```

```
self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()

    def send(self):
        while True:
            # Check frames
            if self.frameIndex >= len(self.data):
                print("No more frames to send")
                break
            # Prepare fram e-- by taking the sn and the element of window
            for index in range(self.frameIndex, self.frameIndex+self.N):
                if index >= len(self.data):
                    break
                frame =
encodeData(str(bin(self.sn)[2:]).zfill(2)+self.data[index])
                # Send frame
                print("[SEND] new frame ", frame, " with sn ", self.sn)
                self.sock.sendall(str.encode(frame))
                # Wait before sending next frame
                sleep(0.05)

            # Wait for ack with a timeout
            self.ackReceivedEvent.clear()
            isNotified = self.ackReceivedEvent.wait(timeout=self.timeout)
            if not isNotified:
                print("[RESEND] resending frame ", frame, " with sn ",
self.sn)
            # If timeout, resend the frame -- run the loop again


    def recvAck(self):
        while True:
            # Listen for data
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
```

```
                seqNo = int(decodedData[1], 2)
                # If ack sn == current sn, just emit ackreceived event
                if seqNo == self.sn:
                    self.increaseSn()
                    self.frameIndex += 1
                    self.ackReceivedEvent.set()
                    print("[ACK] ack received with sn ", seqNo)
                else:
                    print("[ACK] ack discarded as rn not macthed")
            else:
                print("[DISCARD] discarding ACK due to error")


    def increaseSn(self):
        self.sn += 1
        if self.sn > self.snmax:
            self.sn = 0

    def decreaseSn(self):
        self.sn -= 1
        self.sn = max(self.sn, 0)

    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.append(i)



sender = Sender('127.0.0.1', 8081, 4)

sender.pushData("0110100001100101011011010110110001101111011101010111100101110011")
sender.startProcess()
```

Receiver Code -

```
from cmath import sqrt
import socket
import threading

from helpers import decodeData, generateACK
```

```
class Receiver:
    def __init__(self, host, port, N=4):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))
        self.frameReceived = threading.Event()
        self.frameReceived.clear()
        self.N = N

        self.data = []
        self.rn = 0
        tmp = int(sqrt(N).real)
        self.rnmax = int('1'*tmp, 2)

    def startProcess(self):
        self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve data
            data = self.sock.recv(1024)
            if data == "disconnect:" : break
            data = data.decode()
            # Decode data
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # If seq no == rn, save data and send ack for next frame
                if seqNo == self.rn:
                    self.data.append(frame)
                    self.frameReceived.set()
                    self.printData()
                    print("[ACCEPT] Frame received ")
                # If seq no != rn, discard and send ack fagain
```

```
                else:
                    print("[DISCARD] seqNo not matched to rn")
                    self.frameReceived.set()
            else:
                print("[DISCARD] frame due to error")

    def sendACK(self):
        while True:
            # Wait for frame received event
            self.frameReceived.wait()
            print("[ACK] Sending ACK for frame ", self.rn)
            # Send ack for the frame
            self.sock.sendall(str.encode(generateACK(self.rn,
with_parity=True)))
            # Clear frame received event
            self.increaseRn()
            self.frameReceived.clear()

    def increaseRn(self):
        self.rn += 1
        if self.rn > self.rnmax:
            self.rn = 0

    def printData(self):
        print("Data : ", end="", flush=True)
        for i in self.data:
            print(i, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

Output -

Channel -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python channel.py
Socket Server[Channel] is starting....
Server is online ...
Connected to: 127.0.0.1:37620
Connected to: 127.0.0.1:42470
Setup done ! Sender 127.0.0.1:42470 is connected to receiver 127.0.0.1:37620
Injected Error : 00011010000110 ---> 00011010000111
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011010000110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011001010010
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00000000
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 01011011010110
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00010011
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 10011011000111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00100110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 11011011001000
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00110101
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 00011011111111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00000000
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 01011101011000
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00010011
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 10011110011101
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00100110
127.0.0.1:42470 ---> 127.0.0.1:37620 --> 11011100111111
127.0.0.1:37620 ---> 127.0.0.1:42470 --> 00110101
```

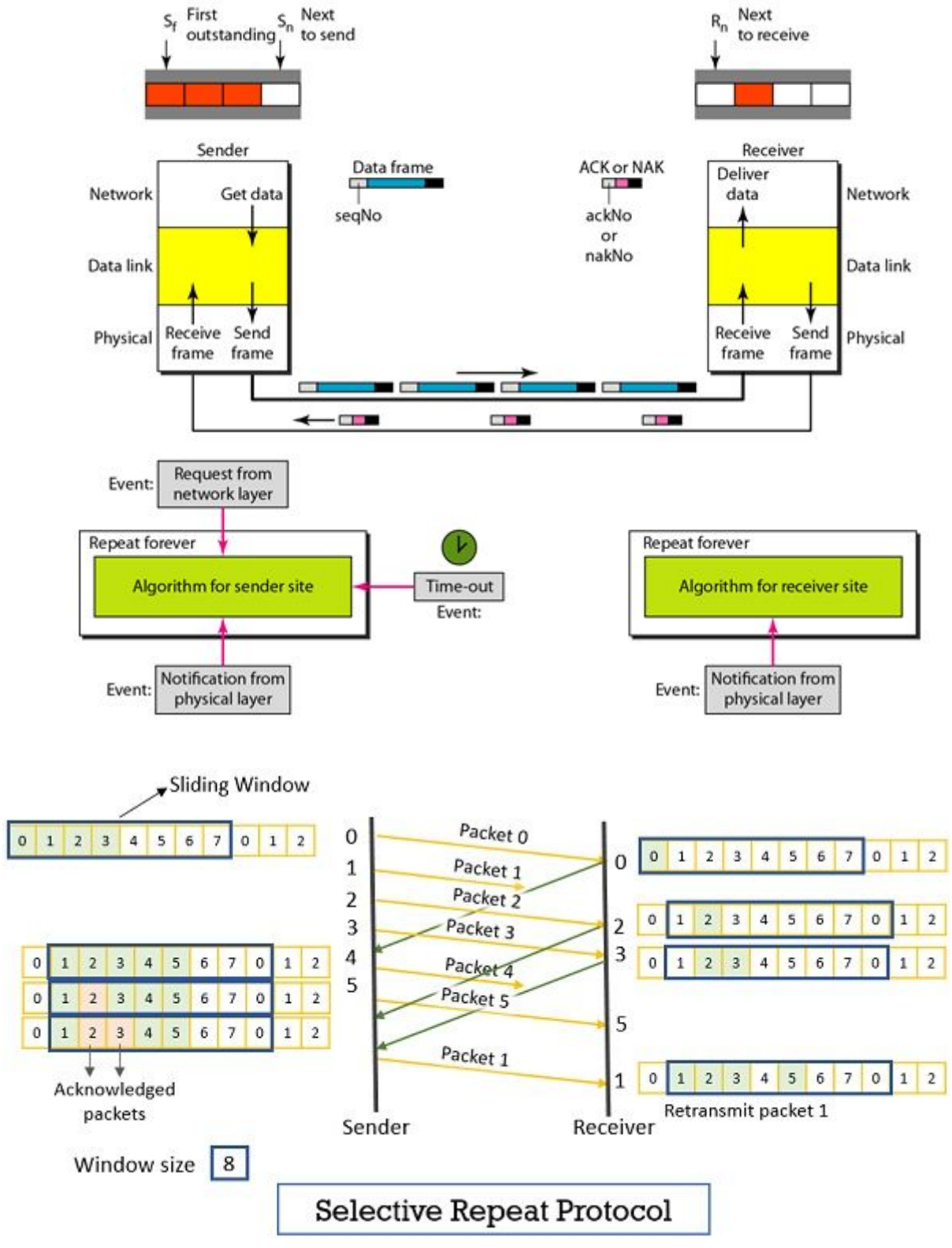
Sender -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python gbn-sender.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 37620
[SEND] new frame  00011010000110  with sn  0
[SEND] new frame  00011001010010  with sn  0
[ACK] ack received with sn  0
[SEND] new frame  01011011010110  with sn  1
[ACK] ack received with sn  1
[SEND] new frame  10011011000111  with sn  2
[ACK] ack received with sn  2
[RESEND] resending frame  10011011000111  with sn  3
[SEND] new frame  11011011001000  with sn  3
[ACK] ack received with sn  3
[SEND] new frame  00011011111111  with sn  0
[ACK] ack received with sn  0
[SEND] new frame  01011101011000  with sn  1
[ACK] ack received with sn  1
[SEND] new frame  10011110011101  with sn  2
[ACK] ack received with sn  2
[RESEND] resending frame  10011110011101  with sn  3
[SEND] new frame  11011100111111  with sn  3
[ACK] ack received with sn  3
[RESEND] resending frame  11011100111111  with sn  0
No more frames to send
```

Receiver -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python gbn-receiver.py
[DISCARD] frame due to error
Data : 01100101
[ACCEPT] Frame received
[ACK] Sending ACK for frame  0
res 00000000
Data : [ACK] Sending ACK for frame  1
0110010101101101
res 00010011
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  2
01100101res 00100110
0110110101101100
[ACCEPT] Frame received
Data : [ACK] Sending ACK for frame  3
01100101res 00110101
011011010110110001101100
[ACCEPT] Frame received
Data : 011001010110110101101100011011000110111101
[ACK] Sending ACK for frame  0
[ACCEPT] Frame received
res 00000000
Data : 0110010101101101011011000110110001101111011101101
[ACCEPT] Frame received
[ACK] Sending ACK for frame  1
res 00010011
Data : [ACK] Sending ACK for frame  2
0110010101101101res 00100110
0110110001101100011011110111010101111001
[ACCEPT] Frame received
Data : 011001010110110101101100011011000110111101110101[ACK] Sending ACK for frame  3
0111100101110011
res 00110101
[ACCEPT] Frame received
```

DESIGN OF SELECTIVE REPEAT PROTOCOL



Code -

Sender Code -

```
from cmath import sqrt
```

```python
from queue import Queue
import socket
import threading
from time import sleep

from helpers import buildFrames, decodeData, encodeData
from queuec import QueueC


# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [type][sn][parity]
# [  1 ][ 4 ][  4   ]
# Type -> ACK -> 1, NAK -> 0

class Sender:
    def __init__(self, host, port, timeout, N=4):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.host, self.port))

        self.data = Queue()
        self.sn = 0
        self.sf = 0


        # tmp = int(sqrt(N).real)
        self.sw = 2**(N-1)
        self.N = N

        self.oldFramesData = {}

        # Window frame status
        self.timers = [None]*(N)

        # Configure the receiver
        receiver_ip = input("Enter receiver ip: ")
        if receiver_ip == None or  len(receiver_ip.strip()) == 0:
```

```
            receiver_ip = "127.0.0.1"
        receiver_port = input("Enter receiver port: ")

self.sock.sendall(str.encode(f'connect:{receiver_ip}:{receiver_port}'))
        sleep(1)

    def startProcess(self):
        self.senderThread = threading.Thread(target=self.send)
        self.receiverACKThread = threading.Thread(target=self.recvAck)
        self.receiverACKThread.start()
        self.senderThread.start()

    def send(self):
        while True:
            if self.data.empty():
                print("end....")
                break

            if (self.sn - self.sf) < self.sw:
                # Prepare fram e-- by taking the sn and the element of window
                x = self.data.get()
                # Make frame
                frame =  encodeData(str(bin(self.sn)[2:]).zfill(2)+x)
                # Store frame
                self.oldFramesData[self.sn] = frame
                # Send frame
                print("Sending frame: ", frame)
                self.sock.sendall(str.encode(frame))
                # Start timer
                self.startTimer(self.sn)
                # Increment sn
                self.sn = (self.sn+1)%self.N

            sleep(0.2)

    def recvAck(self):
        while True:
            # Receive frae
            data = self.sock.recv(1024)
            data = data.decode()
            # If ack is valid
            decodedData = decodeData(data)
            if decodedData[0]:
                # print("Received ack: ", decodedData[1])
```

```
                seqNo = (int(decodedData[1][1:], 2)-1)%self.N
                if decodedData[1][0] == '1':
                    print("[ACK] for frame ", seqNo)
                    # ACK
                    if seqNo > self.sf and seqNo <= self.sn:
                        while self.sf <= seqNo:
                            # Start timer
                            if self.timers[seqNo]:
                                self.timers[seqNo].cancel()
                                self.timers[seqNo] = None
                            self.sf = (self.sf+1)%self.N
                else:
                    # NAK
                    print("[NAK] for frame ", seqNo)
                    if seqNo > self.sf and seqNo < self.sn:
                        self.resendFrameAndSetTimer(seqNo)
            else:
                print("[DISCARD] acknolwedgement de to error")
            # sleep(0.2)


    def startTimer(self, seqNo):
        if self.timers[seqNo]:
            self.timers[seqNo].cancel()
        self.timers[seqNo] = threading.Timer(self.timeout, self.resendFrame,
[seqNo])
        self.timers[seqNo].start()

    def resendFrame(self, seqNo):
        if seqNo not in self.oldFramesData:
            return
        frame =  self.oldFramesData[seqNo]
        print("Resending frame: ", frame)
        self.sock.sendall(str.encode(frame))
        sleep(0.2)

    def resendFrameAndSetTimer(self, seqNo):
        if seqNo not in self.oldFramesData:
            return
        frame =  self.oldFramesData[seqNo] # It wil have encoded frame
        if len(frame) == 14:
            self.sock.sendall(str.encode(frame))
            print("Resending frame: ", frame)
            self.startTimer(seqNo)
```

```
            sleep(0.2)


    def pushData(self, data):
        frames = buildFrames(data, frameSize=8)
        for i in frames:
            self.data.put(i)




sender = Sender('127.0.0.1', 8081, 4)

sender.pushData("0110100001100101011011010110110001101111011101010111100101110011011101110011")

sender.startProcess()
```

Receiver Code -

```
from cmath import sqrt
from queue import Queue
import socket
import threading
from time import sleep

from helpers import decodeData, generateACK

# FRAME FORMAT:
# For a frame with length of 8, the format is:
# [sn][data][parity]
# [2 ][  8 ][  4   ]

# Ack frame format:
# [type][sn][parity]
# [  1 ][ 4 ][  4   ]
# Type -> ACK -> 1, NAK -> 0


class Receiver:
    def __init__(self, host, port, N=4):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
        self.sock.connect((self.host, self.port))
        self.N = N
        self.rn = 0
        # tmp = int(sqrt(N).real)
        self.rnmax = 2**(N-1)

        self.data = []
        self.buffer = ["0"]*self.N

        self.nakSent = False
        self.ackNeeded = False

        self.marked = [False]*self.N

    def startProcess(self):
        # self.sendACKThread = threading.Thread(target=self.sendACK)
        self.receiverThread = threading.Thread(target=self.recv)
        self.receiverThread.start()
        # self.sendACKThread.start()


    def recv(self):
        while True:
            # Receieve frame
            data = self.sock.recv(1024)
            if not data:
                continue
            if data == "disconnect:" : break
            data = data.decode()
            print("data", data)

            # Decode frame
            decodedData = decodeData(data)
            # Check if valid
            if decodedData[0]:
                print("[ACCEPTED] Valid frame")
                # Extract frame and seq no
                data = decodedData[1]
                seqNo = int(data[:2], 2)
                frame = data[2:]
                # if seqno != rn
                if seqNo != self.rn and not self.nakSent:
                    self.sendNAK()
                    self.nakSent = True
```

```
                # If seq no == rn, save data and send ack for next frame
                if self.rn <= seqNo <= self.rnmax and not self.marked[seqNo]:
                    # Store frame
                    self.buffer[seqNo] = frame
                    # Mark Received
                    self.marked[seqNo] = True
                    while self.marked[self.rn]:
                        self.data.append(self.buffer[self.rn])
                        self.marked[self.rn] = False
                        self.rn = (self.rn+1)%self.N
                        self.ackNeeded = True

                    if self.ackNeeded:
                        self.sendACK()
                        self.ackNeeded = False
                        self.nakSent = False
            else:
                print("[DISCARD] Discarding frame due to error")

            self.printData()

    def sendACK(self):
        print("[ACK] Sending ACK for ", self.rn)
        data = str.encode(generateACK(self.rn, with_parity=True,
for_selective_repeat=True, isNak=False))
        self.sock.sendall(data)
        sleep(0.1)

    def sendNAK(self):
        print("[NAK] Sending NAK for ", self.rn)
        data = str.encode(generateACK(self.rn, with_parity=True,
for_selective_repeat=True, isNak=True))
        self.sock.sendall(data)
        sleep(0.1)

    def printData(self):
        print("Data ", end="")
        for frame in self.data:
            print(frame, end="", flush=True)
        print()

receiver = Receiver("127.0.0.1", 8081)
receiver.startProcess()
```

Output -

Channel -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python channel.py
Socket Server[Channel] is starting....
Server is online ...
Connected to: 127.0.0.1:51070
Connected to: 127.0.0.1:42420
Setup done ! Sender 127.0.0.1:42420 is connected to receiver 127.0.0.1:51070
Injected Error : 11011010000100 ---> 11011000000100
127.0.0.1:42420 ---> 127.0.0.1:51070 --> 11011010000100
Client 127.0.0.1:42420 has disconnected
Client 127.0.0.1:51070 has disconnected
Connected to: 127.0.0.1:58038
Connected to: 127.0.0.1:58042
Setup done ! Sender 127.0.0.1:58038 is connected to receiver 127.0.0.1:58042
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011010000110
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100010110
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011001011101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100100011
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011011010100
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100110000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011011001000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011011111111
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100010110
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011101011000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100100011
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011110011101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100110000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011100111111
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
Injected Error : 00011100111101 ---> 00011101111101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011100111101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 01011101011000
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 000000000
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 10011110011101
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 11011100111111
127.0.0.1:58038 ---> 127.0.0.1:58042 --> 00011100111101
127.0.0.1:58042 ---> 127.0.0.1:58038 --> 100000101
```

Sender -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sr-sender-v2.py
Enter receiver ip: 127.0.0.1
Enter receiver port: 58042
Sending frame:  00011010000110
[ACK] for frame  0
Sending frame:  01011001011101
[ACK] for frame  1
Sending frame:  10011011010100
[ACK] for frame  2
Sending frame:  11011011001000
[ACK] for frame  3
Sending frame:  00011011111111
[ACK] for frame  0
Sending frame:  01011101011000
[ACK] for frame  1
Sending frame:  10011110011101
[ACK] for frame  2
Sending frame:  11011100111111
[ACK] for frame  3
Sending frame:  00011100111101
end....
Resending frame:  01011101011000
[NAK] for frame  3
Resending frame:  10011110011101
Resending frame:  11011100111111
Resending frame:  00011100111101
[ACK] for frame  3
```

Receiver -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass2$ python sr-receiver.py
data 00011010000110
[ACCEPTED] Valid frame
[ACK] Sending ACK for  1
res 100010110
Data 01101000
data 01011001011101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  2
res 100100011
Data 0110100001100101
data 10011011010100
[ACCEPTED] Valid frame
[ACK] Sending ACK for  3
res 100110000
Data 0110100001100101011011101
data 11011011001000
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 011010000110010101101101011011100
data 00011011111111
[ACCEPTED] Valid frame
[ACK] Sending ACK for  1
res 100010110
Data 0110100001100101011011010110110001101111
data 01011101011000
[ACCEPTED] Valid frame
[ACK] Sending ACK for  2
res 100100011
Data 011010000110010101101101011011000110111101110101
data 10011110011101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  3
res 100110000
Data 0110100001100101011011010110110001101111011101010111001
data 11011100111111
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 011010000110010101101101011011000110111101110101011110010111011
data 00011101111101
[DISCARD] Discarding frame due to error
Data 011010000110010101101101011011000110111101110101011110010111011
data 01011101011000
[ACCEPTED] Valid frame
[NAK] Sending NAK for  0

res 000000000
Data 011010000110010101101101011011000110111101110101011110010111011
data 10011110011101
[ACCEPTED] Valid frame
Data 011010000110010101101101011011000110111101110101011110010111011
data 11011100111111
[ACCEPTED] Valid frame
Data 011010000110010101101101011011000110111101110101011110010111011
data 00011100111101
[ACCEPTED] Valid frame
[ACK] Sending ACK for  0
res 100000101
Data 011010000110010101101101011011000110111101110101011110010111001011100110111001101110101011110010111011
```

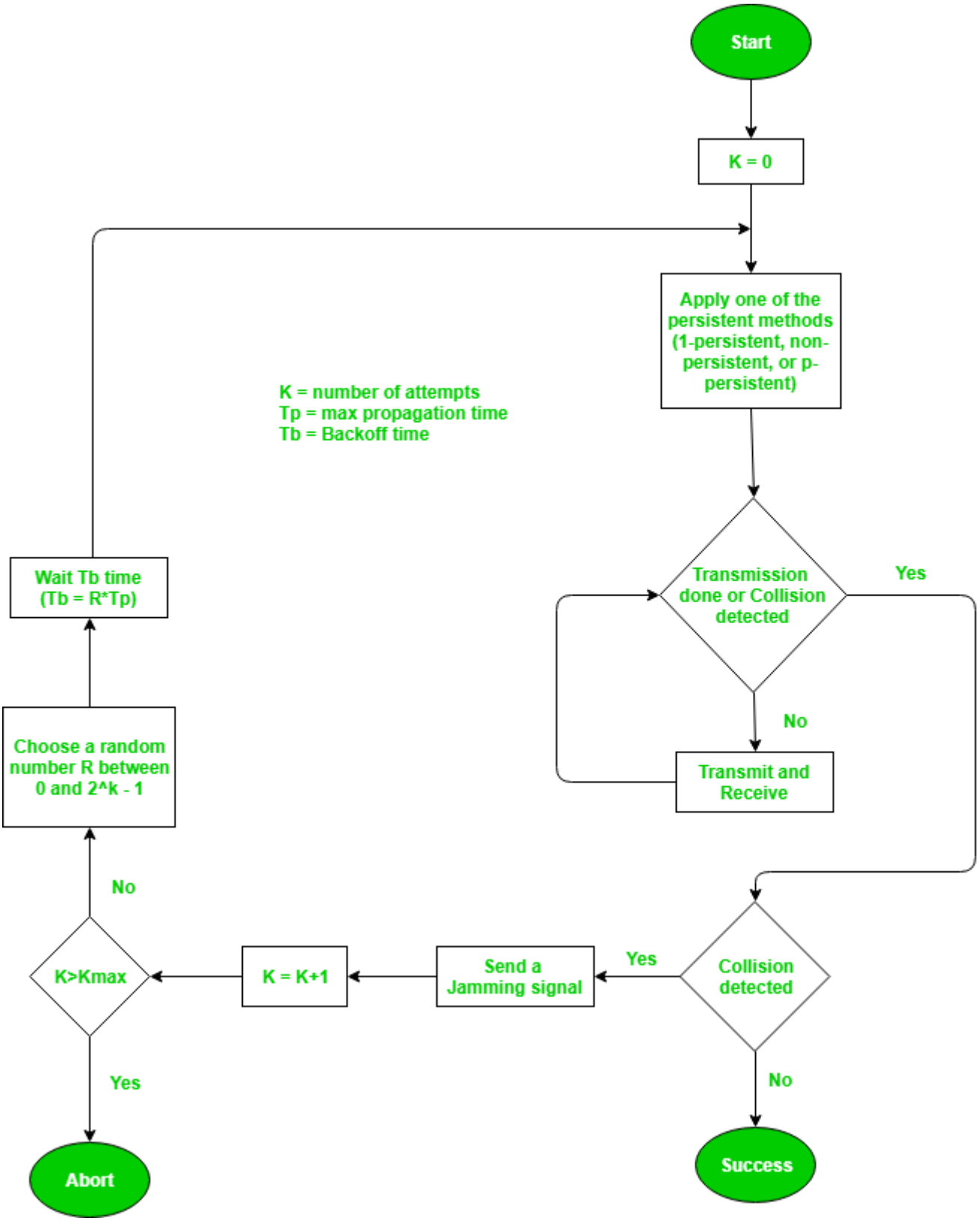
RESULTS AND ANALYSIS

All the protocol is working as expected. The error will not happen in a real-world scenario as we do in the programme. It can be more random. Also, ack the loss will be much in that case. So in the real world, the performance can be slightly different than expected.

Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 3
Subject: Computer Network
Group: A1

Problem Statement: Implement 1-persistent, non-persistent and p-persistent CSMA techniques.

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 the performance of different CSMA techniques.



For CSMA, In our implementation, we have created each thread for each node and to identify whether the channel is busy or idle we used a thread lock to simulate them. Instead of sending a frame from one client to another, we just wait for a static time that is required to send the frame from one client to another client.

IMPLEMENTATION

Code for 1-persistent CSMA

```
import threading
import time


frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0


def log(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

    def run(self):
        global numFrames
        global totalFrames
        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                pass

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the log
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
```

```
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create log thread
    logThread = threading.Thread(target=log, args=[lock, numberNodes*numFrames])
    logThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
        node.join()
    # Wait for log thread to finish
    logThread.join()
```

Code for non-persistent CSMA

```
import random
import threading
import time

frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0

def channel(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

```
        self.lock=lock
        self.index=index

    def run(self):
        global numFrames
        global totalFrames
        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                backOffTime=random.randint(2,5)
                print(f"[Waiting] Fram {count} | Node {self.index} | Backoff
Time {backOffTime}")
                time.sleep(backOffTime)

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the
channel
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create channel thread
    channelThread = threading.Thread(target=channel, args=[lock,
numberNodes*numFrames])
    channelThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
```

```
        node.join()
    # Wait for channel thread to finish
    channelThread.join()
```

Code for p-persistent CSMA

```
import random
import threading
import time

frameTime = 3
interFrameTime = 1

# Total frames currently sent
totalFrames = 0


def channel(lock, totalFramesWillBeSent):
    global totalFrames
    tmp = 0
    used = 0
    while totalFrames < totalFramesWillBeSent:
        if lock.locked():
            used += 1
        tmp += 1
    print(f"Channel utilization -> {used/tmp}")

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

    def run(self):
        global numFrames
        global totalFrames
        global backOffTime
        global probability

        count=1

        while count <= numFrames:
            print(f"[Attempting] Frame {count} | Node {self.index}")
            # If locked wait for milliseconds
            while self.lock.locked():
                pass
```

```
            decision = random.random()
            while decision>probability:
                print(f"[Waiting] Fram {count} | Node {self.index} | Backoff
Time {backOffTime} | Decision {decision}")
                time.sleep(backOffTime)
                while self.lock.locked():
                    pass
                decision=random.random()

            # Acquire lock
            self.lock.acquire()
            # Sleep for frame time -- simulation that the frame sending on the
channel
            time.sleep(frameTime)
            totalFrames += 1
            print(f"[Successful] Frame {count} | Node {self.index}")
            # Release lock
            self.lock.release()
            # Sleep for interframe time
            time.sleep(interFrameTime)
            count+=1

if __name__=='__main__':
    numberNodes = int(input("Enter number of nodes: "))
    numFrames = int(input("Enter number of frames will be sent by frame : "))
    backOffTime = int(input("Enter backoff time: "))
    probability = 1/numberNodes

    lock = threading.Lock() # Lock object
    nodes = [CSMA(lock,i+1) for i in range(0,numberNodes)] # List of nodes
    # Create channel thread
    channelThread = threading.Thread(target=channel, args=[lock,
numberNodes*numFrames])
    channelThread.start()
    # Start all threads
    for node in nodes:
        node.start()
    # Wait for all threads to finish
    for node in nodes:
        node.join()
    # Wait for channel thread to finish
    channelThread.join()
```

OUTPUT

Output for 1-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python 1_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
[Attempting] Frame 1 | Node 1
[Attempting] Frame 1 | Node 2
[Attempting] Frame 1 | Node 3
[Successful] Frame 1 | Node 1
[Attempting] Frame 2 | Node 1
[Successful] Frame 1 | Node 2
[Attempting] Frame 2 | Node 2
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Successful] Frame 2 | Node 2
[Attempting] Frame 3 | Node 2
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Successful] Frame 3 | Node 1
[Attempting] Frame 4 | Node 1
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Successful] Frame 4 | Node 1
[Attempting] Frame 5 | Node 1
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Successful] Frame 5 | Node 1
[Successful] Frame 5 | Node 2
[Successful] Frame 5 | Node 3
Channel utilization -> 0.9972703974209695
```

Output for non-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python non_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
[Attempting] Frame 1 | Node 1
[Attempting] Frame 1 | Node 2
[Waiting] Fram 1 | Node 2 | Backoff Time 4
[Attempting] Frame 1 | Node 3
[Waiting] Fram 1 | Node 3 | Backoff Time 3
[Successful] Frame 1 | Node 1
[Attempting] Frame 2 | Node 1
[Waiting] Fram 2 | Node 1 | Backoff Time 4
[Waiting] Fram 1 | Node 2 | Backoff Time 5
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Waiting] Fram 2 | Node 1 | Backoff Time 2
[Waiting] Fram 1 | Node 2 | Backoff Time 2
[Waiting] Fram 2 | Node 1 | Backoff Time 2
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Waiting] Fram 2 | Node 1 | Backoff Time 3
[Successful] Frame 1 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Attempting] Frame 2 | Node 2
[Waiting] Fram 2 | Node 2 | Backoff Time 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Waiting] Fram 3 | Node 1 | Backoff Time 3
[Waiting] Fram 3 | Node 3 | Backoff Time 5
[Successful] Frame 2 | Node 2
[Attempting] Frame 3 | Node 2
[Waiting] Fram 3 | Node 2 | Backoff Time 3
[Waiting] Fram 3 | Node 3 | Backoff Time 2
[Successful] Frame 3 | Node 1
[Attempting] Frame 4 | Node 1
[Waiting] Fram 4 | Node 1 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 3
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Waiting] Fram 4 | Node 2 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 3
[Successful] Frame 4 | Node 1
[Attempting] Frame 5 | Node 1
[Waiting] Fram 5 | Node 1 | Backoff Time 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Waiting] Fram 5 | Node 1 | Backoff Time 4
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 5
[Waiting] Fram 5 | Node 1 | Backoff Time 2
[Successful] Frame 5 | Node 2
[Waiting] Fram 3 | Node 3 | Backoff Time 4
[Successful] Frame 5 | Node 1
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Successful] Frame 5 | Node 3
Channel utilization -> 0.761094890756491
```

Output for p-persistent -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass3$ python p_persistent.py
Enter number of nodes: 3
Enter number of frames will be sent by frame : 5
Enter backoff time: 1
[Attempting] Frame 1 | Node 1
[Waiting] Fram 1 | Node 1 | Backoff Time 1 | Decision 0.44661324889794696
[Attempting] Frame 1 | Node 2
[Attempting] Frame 1 | Node 3
[Successful] Frame 1 | Node 2
[Waiting] Fram 1 | Node 1 | Backoff Time 1 | Decision 0.8762436572912545
[Waiting] Fram 1 | Node 3 | Backoff Time 1 | Decision 0.6219134203679332
[Attempting] Frame 2 | Node 2
[Successful] Frame 1 | Node 3
[Attempting] Frame 2 | Node 3
[Successful] Frame 2 | Node 2
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.39491054673103576
[Attempting] Frame 3 | Node 2
[Successful] Frame 1 | Node 1
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.6877240626941749
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.804523328874772
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.7538629728803926
[Attempting] Frame 2 | Node 1
[Waiting] Fram 2 | Node 1 | Backoff Time 1 | Decision 0.3790190034986691
[Waiting] Fram 2 | Node 3 | Backoff Time 1 | Decision 0.4851412069250256
[Waiting] Fram 2 | Node 1 | Backoff Time 1 | Decision 0.6837603032060176
[Waiting] Fram 3 | Node 2 | Backoff Time 1 | Decision 0.9343598397899764
[Successful] Frame 2 | Node 3
[Attempting] Frame 3 | Node 3
[Successful] Frame 2 | Node 1
[Attempting] Frame 3 | Node 1
[Successful] Frame 3 | Node 2
[Attempting] Frame 4 | Node 2
[Successful] Frame 3 | Node 1
[Waiting] Fram 3 | Node 3 | Backoff Time 1 | Decision 0.555302821463085
[Waiting] Fram 4 | Node 2 | Backoff Time 1 | Decision 0.4205059484333239
[Attempting] Frame 4 | Node 1
[Successful] Frame 4 | Node 1
[Waiting] Fram 3 | Node 3 | Backoff Time 1 | Decision 0.6889952073915103
[Attempting] Frame 5 | Node 1
[Successful] Frame 4 | Node 2
[Attempting] Frame 5 | Node 2
[Successful] Frame 3 | Node 3
[Attempting] Frame 4 | Node 3
[Successful] Frame 5 | Node 2

[Waiting] Fram 4 | Node 3 | Backoff Time 1 | Decision 0.6235615738429497
[Waiting] Fram 5 | Node 1 | Backoff Time 1 | Decision 0.9021538466265436
[Waiting] Fram 4 | Node 3 | Backoff Time 1 | Decision 0.5907892551227651
[Successful] Frame 5 | Node 1
[Successful] Frame 4 | Node 3
[Attempting] Frame 5 | Node 3
[Waiting] Fram 5 | Node 3 | Backoff Time 1 | Decision 0.860453998302961
[Waiting] Fram 5 | Node 3 | Backoff Time 1 | Decision 0.5010107987806726
[Successful] Frame 5 | Node 3
Channel utilization -> 0.6788664549915409
```

RESULTS & ANALYSIS

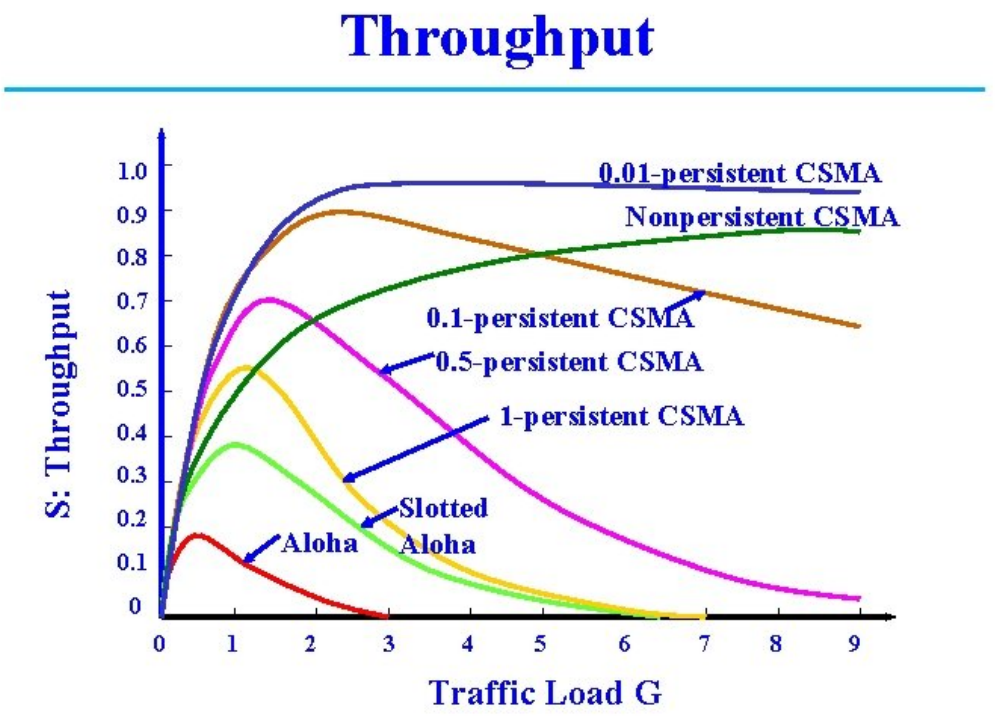
p-persistent CSMA:

- This method is used when the channel has time slots such that the time slot duration is equal to or greater than the maximum propagation delay time.
- Whenever a station becomes ready to send, it senses the channel.
- If the channel is busy, the station waits until the next slot.
- If the channel is idle, it transmits with a probability p.
- With the probability q=l-p, the station then waits for the beginning of the next time slot.
- If the next slot is also idle, it either transmits or waits again with probabilities p and q.
- This process is repeated till either frame has been transmitted or another station has begun transmitting.
- In case of the transmission by another station, the station acts as though a collision has occurred and it waits a random amount of time and starts again.

Advantages of p-persistent CSMA:

- It reduces the chance of collision and improves the efficiency of the network.

CSMA/CD :



Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 4
Subject: Computer Network
Group: A1

Assignment 4: Implement CDMA with Walsh code.

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.

DESIGN

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

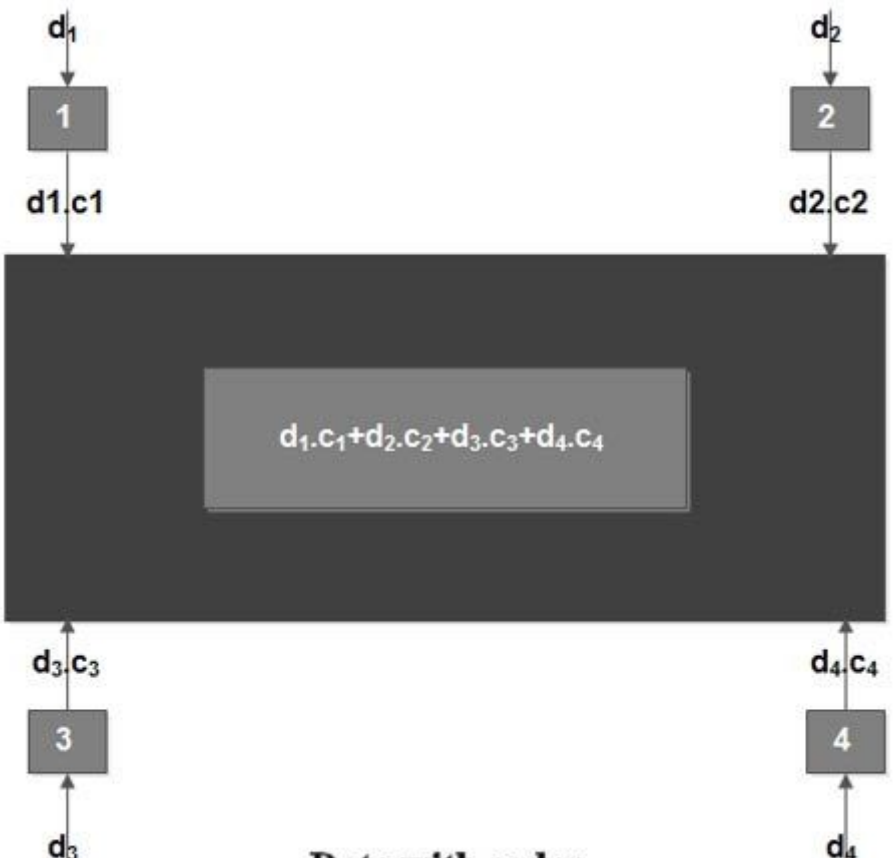
- channel.py (Program for channel)
- station.py (Program for a station process)

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

1. channel.py – The following are the tasks performed in this program :
    A. Asks for a number of stations and initiates all stations.
    B. Creates a Walsh Table for the given number of stations (highest power of 2).
    C. Receives data bits from each station.
    D. Multiplies data bits with the corresponding Walsh Sequence of each station, and sums them to get the final data.
    E. Asks for sender and receiver station number, from which sender to which receiver we want to send the data bit.
    F. Calculates the data bit by multiplying the final data with the Walsh sequence of the sender station, summing it up; and then dividing the result by the number of stations.

2. station.py – The following are the tasks performed in this program :
    A. Sends the stream of data bits to the channel process.
    B. Let's say the maximum length of data bits sent by a station is X. If a station sends a stream having a length less than X, then the rest of the bits are assumed to be silent.
    C. Receives a data bit from the channel.

IMPLEMENTATION



Code for the channel.py -

```
import json
import socket
import helpers


class Channel():

    def __init__(self, totalstations):
        self.totalstation = totalstations
        self.stationhost = '127.0.0.1'
        self.stationport = 8082
        self.stationconn = []
        self.walshtable = [[]]

    def initStations(self):
        stationSocket = socket.socket()
        stationSocket.bind((self.stationhost, self.stationport))
        stationSocket.listen(self.totalstation)
        for i in range(1, self.totalstation+1):
            conn = stationSocket.accept()
            self.stationconn.append(conn)
        print('Initiated all station connections')
```

```
def closeStations(self):
    for conn in self.stationconn:
        conn[0].close()
    print('Closed all station connections')

def generateWalshTable(self):
    print('Generating Walsh table ...')
    self.walshtable = helpers.generateWalshTable(self.totalstation)

def process(self):
    data = []
    for i in range(self.totalstation):
        conn = self.stationconn[i]
        d = conn[0].recv(1024).decode()
        data.append(d)

    for i in range(self.totalstation):
        print('Station',i+1,'will send data:',end=' ')
        print(data[i])
    maxlen = 0
    for i in data:
        maxlen = max(maxlen,len(i))
    datavalue = []
    for d in data:
        tup = []
        for j in range(maxlen):
            if j < len(d):
                if d[j] == '0':
                    tup.append(-1)
                elif d[j] == '1':
                    tup.append(1)
            else:
                tup.append(0)
        datavalue.append(tup)

    for i in range(maxlen):
        print('-----------------------------------------------')
        print('Sending bit',i+1,'of each station\'s data')
        finaldata = []
        d = []
        c = []
        n = len(self.walshtable)
        for j in range(n):
            if j < self.totalstation:
```

```
                    d.append(datavalue[j][i])
                else:
                    d.append(1)
                c.append(self.walshtable[j])
                finaldata.append(0)

            for j in range(n):
                temp = helpers.multiplyScalar(c[j], d[j])
                finaldata = helpers.addTuples(finaldata,temp)
            print('Bit',i+1,'of each station is:', end=' ')
            print(d)
            print('Final data is:'+json.dumps(finaldata))

            choice = input('Does any station want to receive data ? (y/n) ')
            while choice == 'y':
                # User input
                stationNo = int(input("Enter the station number: "))
                receiverNo = int(input("Enter the receiver station number:
"))
                if stationNo > self.totalstation or stationNo <= 0 or
receiverNo > self.totalstation or receiverNo <= 0:
                    # Invalid check
                    print('Invalid station number')
                else:
                    temp = helpers.multiplyTuples(finaldata, c[stationNo-1])
                    summ = sum(temp)
                    databit = str(summ//n)
                    conn = self.stationconn[receiverNo-1]
                    conn[0].sendto(databit.encode(), conn[1])
                    print('Multiplying final data with Code bits of sender
station',stationNo)
                    print('The result is : ',temp)
                    print('the sum of result is:',summ)
                    print('THE DATA BIT OF STATION',stationNo,'is:',databit)
                    print('Data bit sent to
receiver',receiverNo,'successfully!')

                    choice = input('Does any station want to receive data ?
(y/n) ')


if __name__ == '__main__':
    totalstations = int(input('Enter number of stations: '))

    ch = Channel(totalstations)
```

```
    ch.generateWalshTable()
    helpers.displayWalshTable(ch.walshtable)
    ch.initStations()
    ch.process()
    ch.closeStations()
```

Code for Station.py

```
import socket
import time
import random
import sys

class Station:
    def __init__(self, index, host, port):
        self.index = index
        self.host = host
        self.port = port

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

    def startSending(self):
        data = input("Enter data to send: ")
        self.socketNew.send(data.encode())

    def startReceiving(self):
        while True:
            data = self.socketNew.recv(1024).decode()
            if not data:
                break
            print("Received from channel: " + data)
            data = int(data)
            if data == -1:
                val = 0
            elif data == 1:
                val = 1
            else:
                val = "silent"
            print("Value of received bit is " + str(val))
        self.socketNew.close()

    def startProcess(self):
        self.startSending()
        self.startReceiving()
```

```
if __name__ == '__main__':
    station = Station(sys.argv[1], "127.0.0.1", 8082)
    station.startProcess()
```

Code for helpers.py

```
def multiplyTuples(t1, t2):
      tup = []
      for i in range(len(t1)):
            tup.append(t1[i] * t2[i])
      return tup


def multiplyScalar(t1, x):
      tup = []
      for i in range(len(t1)):
            tup.append(t1[i] * x)
      return tup


def addTuples(t1, t2):
      tup = []
      for i in range(len(t1)):
            tup.append(t1[i]+t2[i])
      return tup


def generateWalshTable(n):
    print('Generating Walsh table ...')
    p = 1
    prevtable = [[1]]
    while p < n:
        p *= 2
        curtable = []
        for i in range(p):
            tup = []
            for j in range(p):
                tup.append(0)
            curtable.append(tup)
        for i in range(0, p//2):
            for j in range(0, p//2):
                curtable[i][j] = prevtable[i][j]
                curtable[i+p//2][j] = prevtable[i][j]
                curtable[i][j+p//2] = prevtable[i][j]
                curtable[i+p//2][j+p//2] = -1*prevtable[i][j]
        prevtable = curtable
```

```
    return prevtable

def displayWalshTable(walshtable):
    p = len(walshtable)
    for i in range(p):
        for j in range(p):
            if walshtable[i][j] == 1:
                print(end=' ')
            print(walshtable[i][j],end=' ')
        print()
```

OUTPUT

Channel -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass4$ python channel.py
Enter number of stations: 4
Generating Walsh table ...
Generating Walsh table ...
 1  1  1  1
 1 -1  1 -1
 1  1 -1 -1
 1 -1 -1  1
Initiated all station connections
Station 1 will send data: 1
Station 2 will send data: 0
Station 3 will send data: 1
Station 4 will send data: 0
---------------------------------------------
Sending bit 1 of each station's data
Bit 1 of each station is: [1, -1, 1, -1]
Final data is:[0, 4, 0, 0]
Does any station want to receive data ? (y/n) y
Enter the station number: 1
Enter the receiver station number: 3
Multiplying final data with Code bits of sender station 1
The result is :  [0, 4, 0, 0]
the sum of result is: 4
THE DATA BIT OF STATION 1 is: 1
Data bit sent to receiver 3 successfully!
Does any station want to receive data ? (y/n) y
Enter the station number: 1
Enter the receiver station number: 2
Multiplying final data with Code bits of sender station 1
The result is :  [0, 4, 0, 0]
the sum of result is: 4
THE DATA BIT OF STATION 1 is: 1
Data bit sent to receiver 2 successfully!
Does any station want to receive data ? (y/n) y
Enter the station number: 2
Enter the receiver station number: 4
Multiplying final data with Code bits of sender station 2
The result is :  [0, -4, 0, 0]
the sum of result is: -4
THE DATA BIT OF STATION 2 is: -1
Data bit sent to receiver 4 successfully!
```

Station 1 -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass4$ python station.py 1
Enter data to send: 1
```

Station 2 -

```
p(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass4$ python station.py 2
Enter data to send: 0
Received from channel: 1
Value of received bit is 1
```

Station 3 -

```
(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass4$ python station.py 3
Enter data to send: 1
Received from channel: 1
Value of received bit is 1
```

Station 4 -

```
pytho(base) tanmoy@tanmoy-laptop:~/Desktop/Lab/Computer Network/Ass4$ python station.py 4
Enter data to send: 0
Received from channel: -1
Value of received bit is 0
```

ANALYSIS

Unlike TDMA, in CDMA all stations can transmit data simultaneously, there is no timesharing.
CDMA allows each station to transmit over the entire frequency spectrum all the time.
Multiple simultaneous transmissions are separated using coding theory.
In CDMA each user is given a unique code sequence.
The basic idea of CDMA is explained below:

1. Let us assume that we have four stations 1, 2, 3 and 4 that are connected to same channel. The data from station 1 are dl, from station 2 are d2 and so on.
2. The code assigned to first station is C1, to the second is C2 and so on.
3. These assigned codes have two properties:
    (a) If we multiply each code by another, we get O.
    (b) If we multiply each code by itself, we get 4. (No. of stations).
4. When these four stations are sending data on the same channel, station 1 multiplies its data by its code i.e. dl .c1}, station 2 multiplies its data by its code i.e. d2 .C2 and so on.
5. The data that go on channel are the sum of all these terms as shown in Fig.
6. Any station that wants to receive data from one of the other three stations multiplies the data on channel by the code of the sender. For example, suppose station 1 and 2 are talking to each other. Station 2 wants to hear what station 1 is saying. It multiples the data on the channel by CI (the code of station 1).
7. Because (C1. C1) is 4, but (C2. C1), (C3. C1), and (C4. C1) are all zeroes, station 2 divides the result by 4 to get the data from station 1.

data = (dl . C1+ d2 . C2+ d3. C3+ d4. C4)

C1 = dl. C1 . C1+ d2 . C2. C1+ d3 . C3. C1 + d4 . C4. C1= 4 x dl

- The code assigned to each station is a sequence of numbers called chips. These chips are called orthogonal sequences. This sequence has following properties:
  1. Each sequence is made of N elements, where N is the number of stations as shown
  2. If we multiple a sequence by a number, every element in the sequence is multiplied by that element. This is called multiplication of a sequence by a scalar.
     For example: [+1 +1-1 -1] = [+2 +2 -2 -2]
  3. If we multiply two equal sequences, element by element and add the results, we get N, where N is the number of elements in each sequence. This is called inner product of two equal sequences.
     For example: [+1 +1-1 -1] . [+1 +1-1-1] = 1+ 1+ 1+ 1 = 4

4. If we multiply two different sequences, element by element and add the results, we get 0. This is called inner product of two different sequences.

For example: [+1 +1-1-1]. [+1 +1 +1 +1] = 1+1-1 -1= 0

5. Adding two sequences means adding the corresponding elements. The result is another sequence.

For example: [+1 +1-1 -1] + [+1 +1+1 +1] = [+2 +2 0 0]

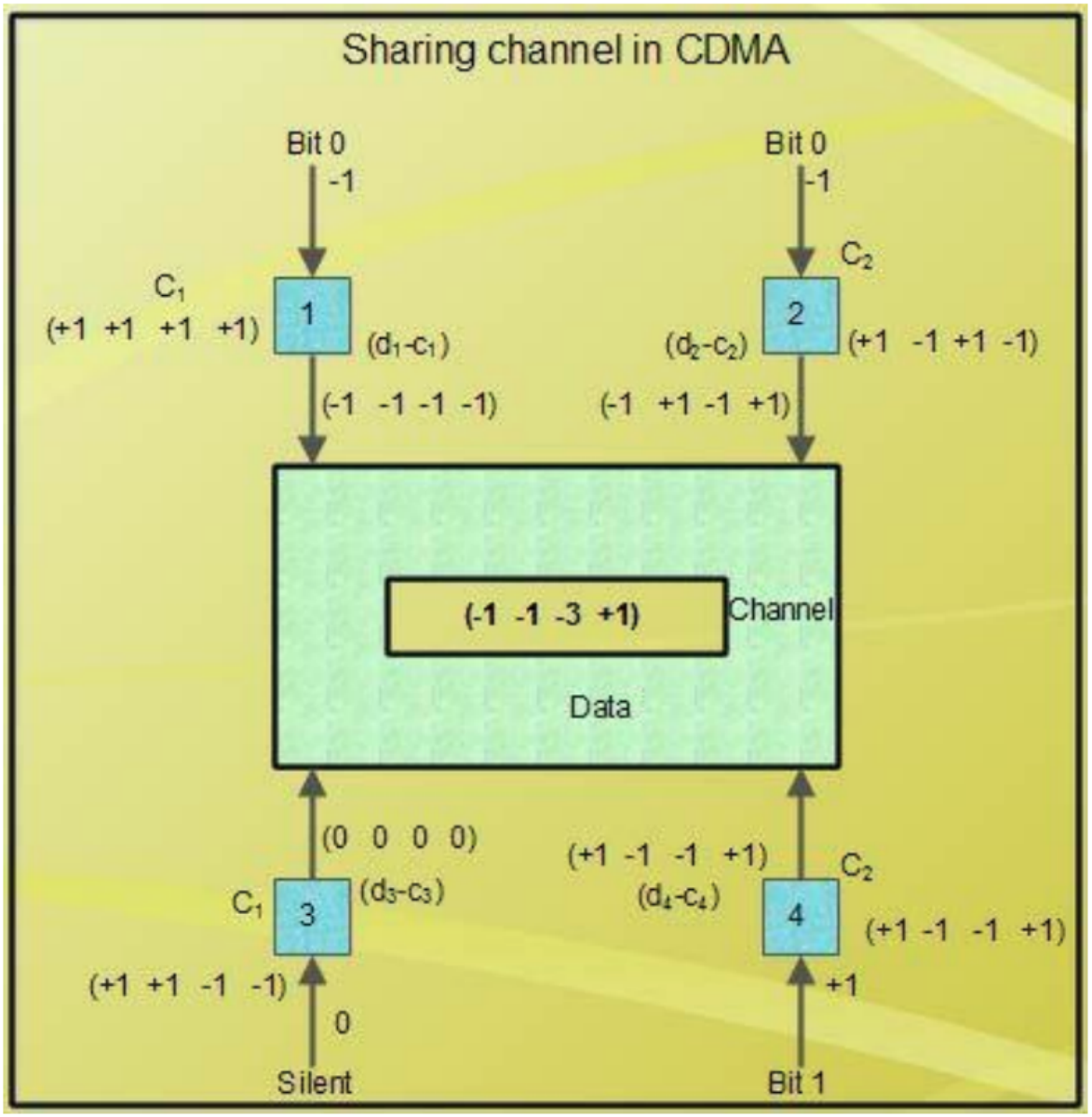
The data representation and encoding is done by different stations in following manner:

a. If a station needs to send a 0 bit, it encodes it as -1
b. If it needs to send a 1 bit, it encodes it as + 1.
c. When station is idle, it sends no signal, which is interpreted as a 0.

For example,

If station 1 and station 2 are sending a 0 bit, station 3 is silent and station 4 is sending a 1 bit; the data at sender site are represented as -1, - 1,0 and +1 respectively.

Each station multiplies the corresponding number by its chip, which is unique for each station. Each station send this sequence to the channel ; The sequence of channel is the sum of all four sequence as shown in fig.



If station 3, which was silent, is listening to station 2. Station 3 multiplies the total data on the channel by the code for station 2, which is [+ 1 -1 +1 -1], to get [-1 -1 -3+1] . [+1 -1 +1 -1]= -4/4 = -1 --> bit 0

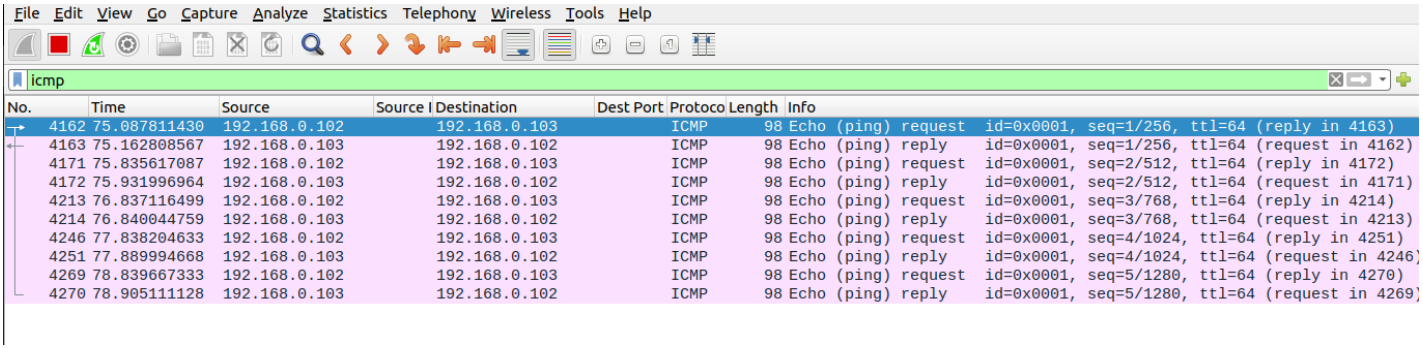
Name : Tanmoy Sarkar
Roll No : 002010501020
Class : BCSE III
Assignment No : 5
Subject : Computer Network
Group : A1

Overview

Wireshark is an open source cross-platform packet capture and analysis tool, with versions for Windows and Linux. The GUI window gives a detailed breakdown of the network protocol stack for each packet, colourizing packet details based on protocol, as well as having functionality to filter and search the traffic, and pick out TCP streams. Wireshark can also save packet data to files for offline analysis and export/import packet captures to/from other tools. Statistics can also be generated for packet capture files.

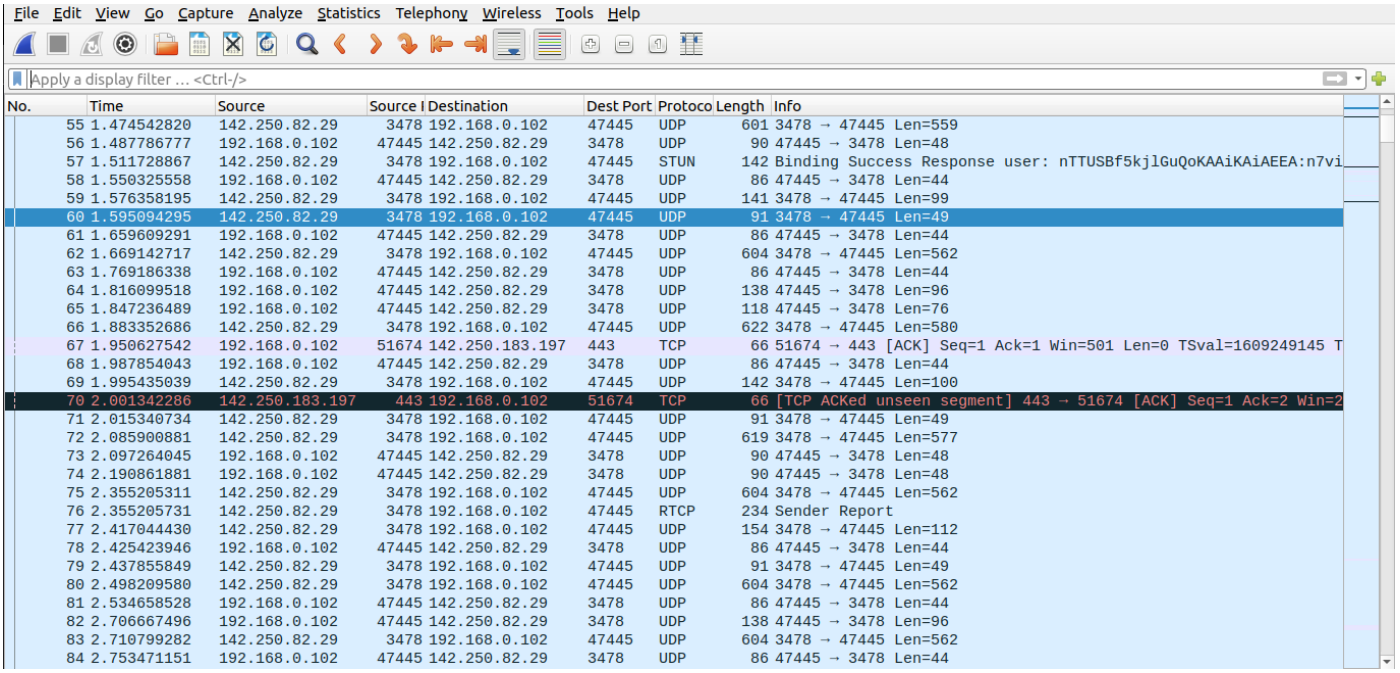
1. Generate some ICMP traffic by using the Ping command line tool to check the connectivity of a neighbouring machine (or router). Note the results in Wireshark. The initial ARP request broadcast from your PC determines the physical MAC address of the network IP Address, and the ARP reply from the neighbouring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

```
(base) tanmoy@tanmoy-laptop:~$ ping 192.168.0.103 -c 5
PING 192.168.0.103 (192.168.0.103) 56(84) bytes of data.
64 bytes from 192.168.0.103: icmp_seq=1 ttl=64 time=329 ms
64 bytes from 192.168.0.103: icmp_seq=2 ttl=64 time=96.4 ms
64 bytes from 192.168.0.103: icmp_seq=3 ttl=64 time=2.95 ms
64 bytes from 192.168.0.103: icmp_seq=4 ttl=64 time=51.8 ms
64 bytes from 192.168.0.103: icmp_seq=5 ttl=64 time=65.5 ms

--- 192.168.0.103 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 2.949/109.085/328.769/113.900 ms
```



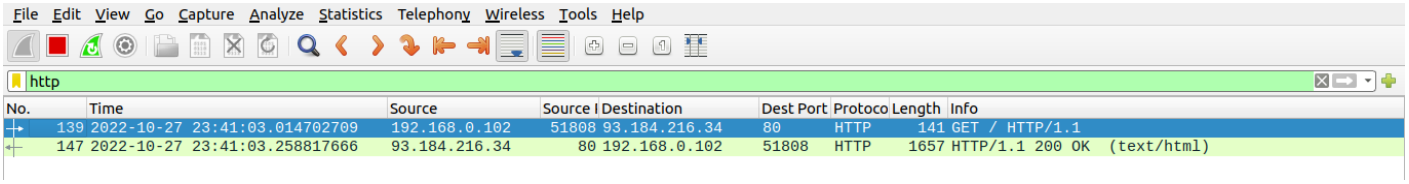
2. Generate some web traffic and

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



No.	Time	Source	Source I	Destination	Dest Port	Protoco	Length	Info
55	1.474542820	142.250.82.29	3478	192.168.0.102	47445	UDP	601	3478 → 47445 Len=559
56	1.487786777	192.168.0.102	47445	142.250.82.29	3478	UDP	90	47445 → 3478 Len=48
57	1.511728867	142.250.82.29	3478	192.168.0.102	47445	STUN	142	Binding Success Response user: nTTUSBf5kjlGuQoKAAiKAiAEEA:n7vi
58	1.550325558	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
59	1.576358195	142.250.82.29	3478	192.168.0.102	47445	UDP	141	3478 → 47445 Len=99
60	1.595094295	142.250.82.29	3478	192.168.0.102	47445	UDP	91	3478 → 47445 Len=49
61	1.659609291	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
62	1.669142717	142.250.82.29	3478	192.168.0.102	47445	UDP	604	3478 → 47445 Len=562
63	1.769186338	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
64	1.816099518	192.168.0.102	47445	142.250.82.29	3478	UDP	138	47445 → 3478 Len=96
65	1.847236489	192.168.0.102	47445	142.250.82.29	3478	UDP	118	47445 → 3478 Len=76
66	1.883352686	142.250.82.29	3478	192.168.0.102	47445	UDP	622	3478 → 47445 Len=580
67	1.950627542	192.168.0.102	51674	142.250.183.197	443	TCP	66	51674 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=1609249145 T
68	1.987854043	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
69	1.995435039	142.250.82.29	3478	192.168.0.102	47445	UDP	142	3478 → 47445 Len=100
70	2.001342286	142.250.183.197	443	192.168.0.102	51674	TCP	66	[TCP ACKed unseen segment] 443 → 51674 [ACK] Seq=1 Ack=2 Win=2
71	2.015340734	142.250.82.29	3478	192.168.0.102	47445	UDP	91	3478 → 47445 Len=49
72	2.085900881	142.250.82.29	3478	192.168.0.102	47445	UDP	619	3478 → 47445 Len=577
73	2.097264045	192.168.0.102	47445	142.250.82.29	3478	UDP	90	47445 → 3478 Len=48
74	2.190861881	192.168.0.102	47445	142.250.82.29	3478	UDP	90	47445 → 3478 Len=48
75	2.355205311	142.250.82.29	3478	192.168.0.102	47445	UDP	604	3478 → 47445 Len=562
76	2.355205731	142.250.82.29	3478	192.168.0.102	47445	RTCP	234	Sender Report
77	2.417044430	142.250.82.29	3478	192.168.0.102	47445	UDP	154	3478 → 47445 Len=112
78	2.425423946	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
79	2.437855849	142.250.82.29	3478	192.168.0.102	47445	UDP	91	3478 → 47445 Len=49
80	2.498209580	142.250.82.29	3478	192.168.0.102	47445	UDP	604	3478 → 47445 Len=562
81	2.534658528	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44
82	2.706667496	192.168.0.102	47445	142.250.82.29	3478	UDP	138	47445 → 3478 Len=96
83	2.710799282	142.250.82.29	3478	192.168.0.102	47445	UDP	604	3478 → 47445 Len=562
84	2.753471151	192.168.0.102	47445	142.250.82.29	3478	UDP	86	47445 → 3478 Len=44

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.)



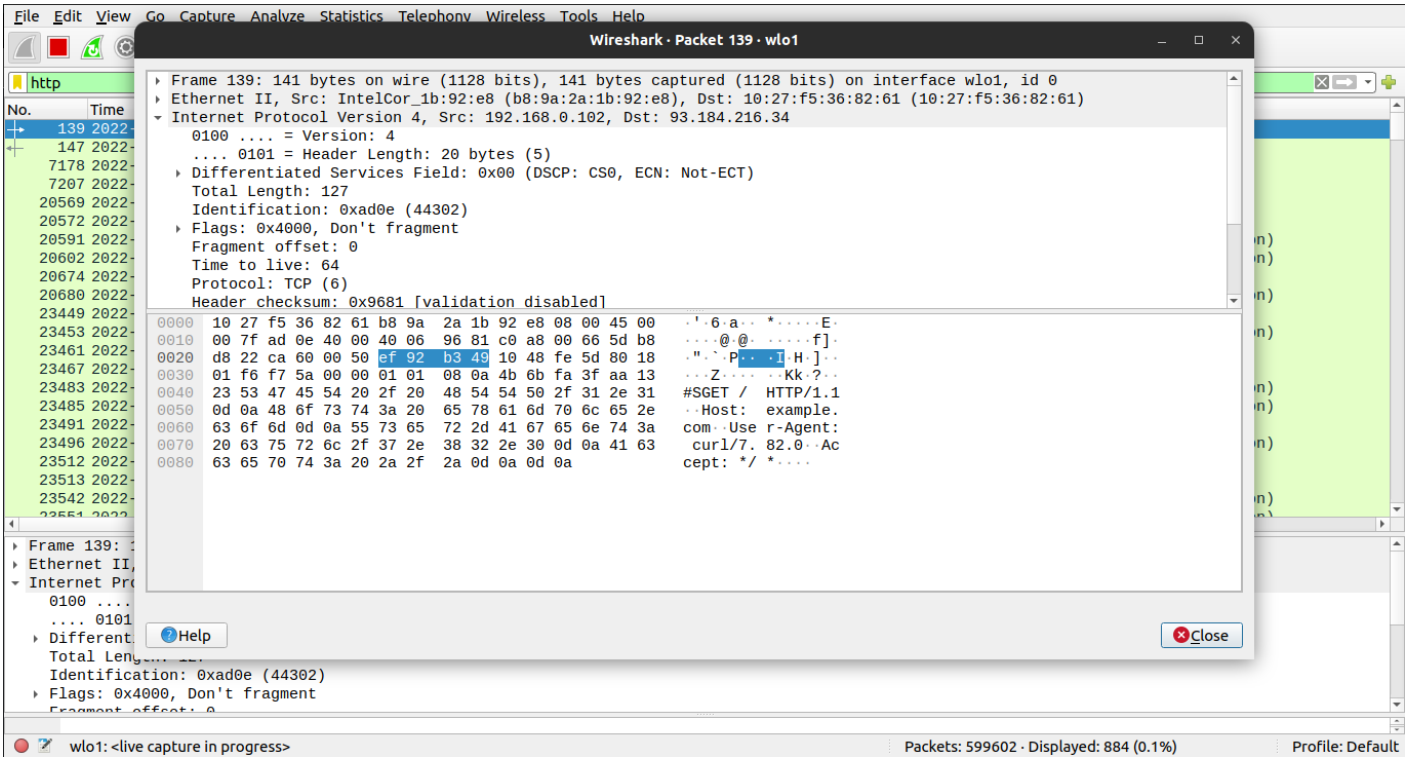
No.	Time	Source	Source I	Destination	Dest Port	Protoco	Length	Info
139	2022-10-27 23:41:03.014702709	192.168.0.102	51808	93.184.216.34	80	HTTP	141	GET / HTTP/1.1
147	2022-10-27 23:41:03.258817666	93.184.216.34	80	192.168.0.102	51808	HTTP	1657	HTTP/1.1 200 OK (text/html)

Request sent at 23:41:03.014702709 and received response at 23:41:03.258817666.
So, delay = 0.244114957 seconds

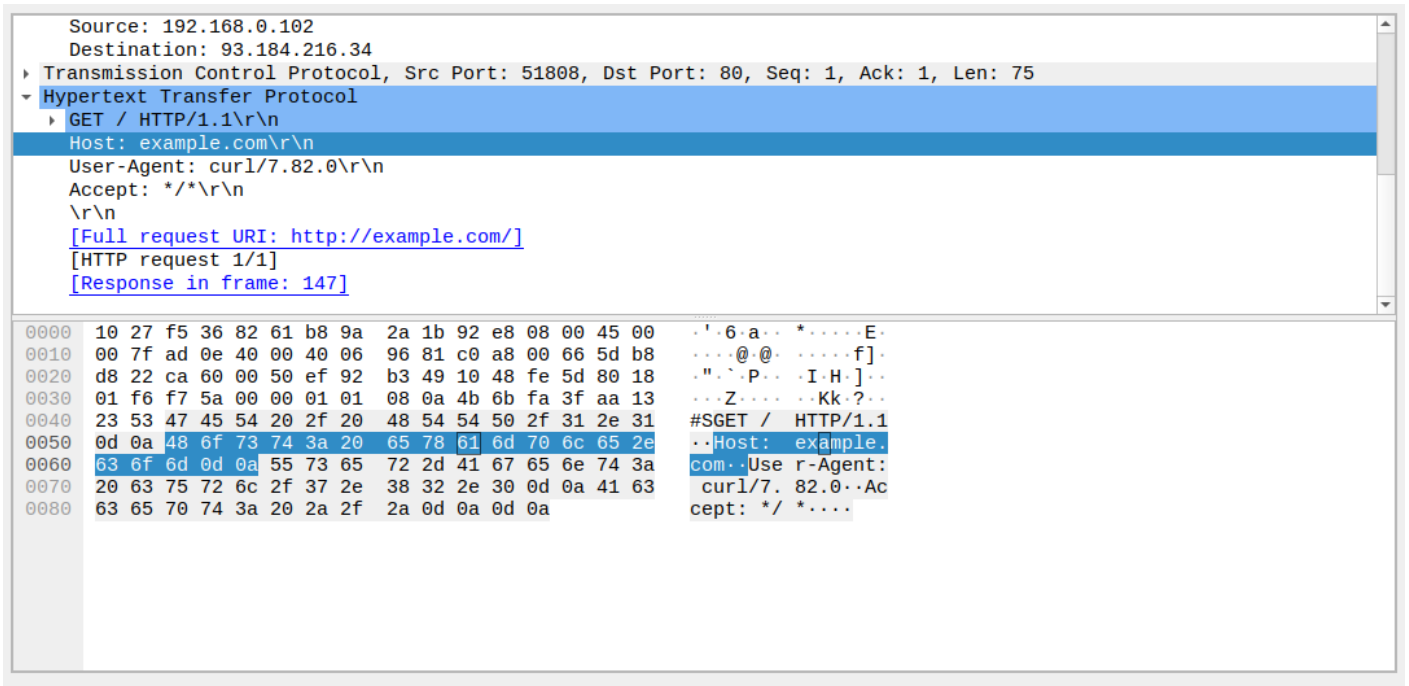
c. What is the Internet address of the website? What is the Internet address of your computer?

Internet address of website : 93.184.216.34
Internet address of computer : 192.168.0.102

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.

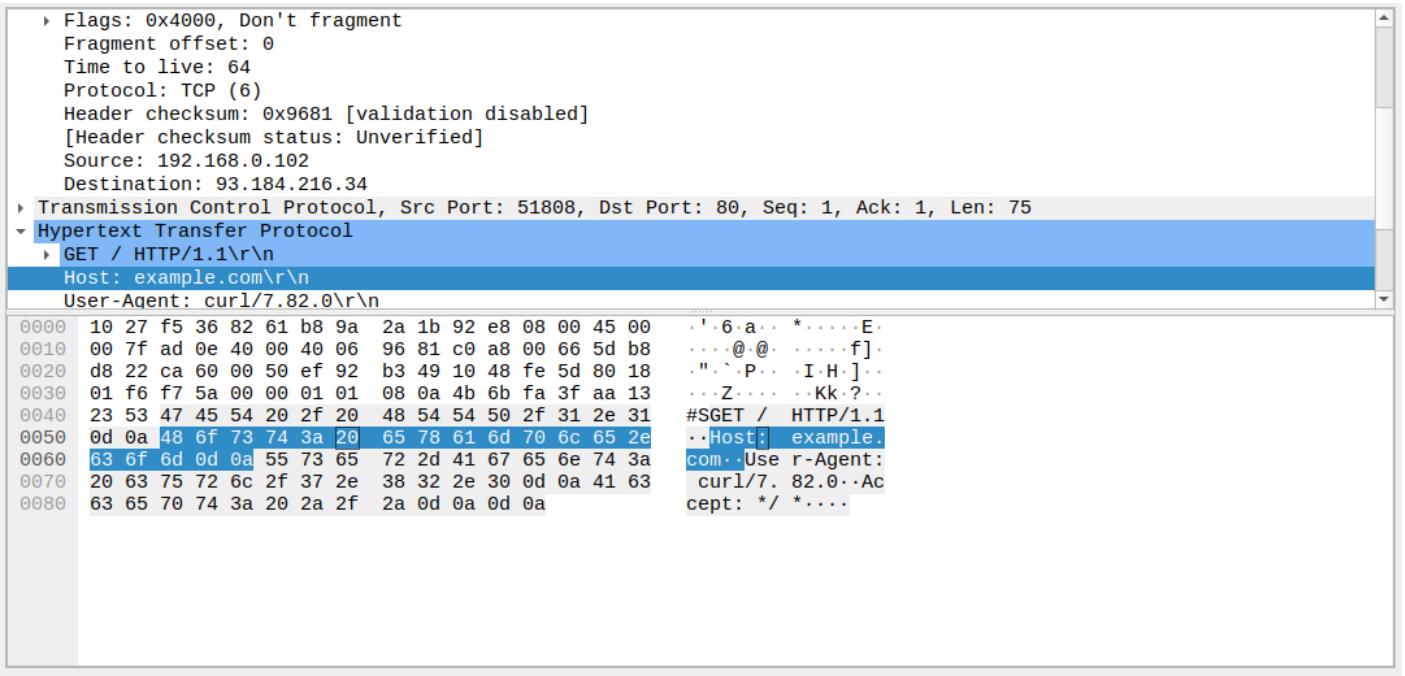


e. Find out the value of the Host from the Packet Details Panel, within the GET command.



Host : example.com

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

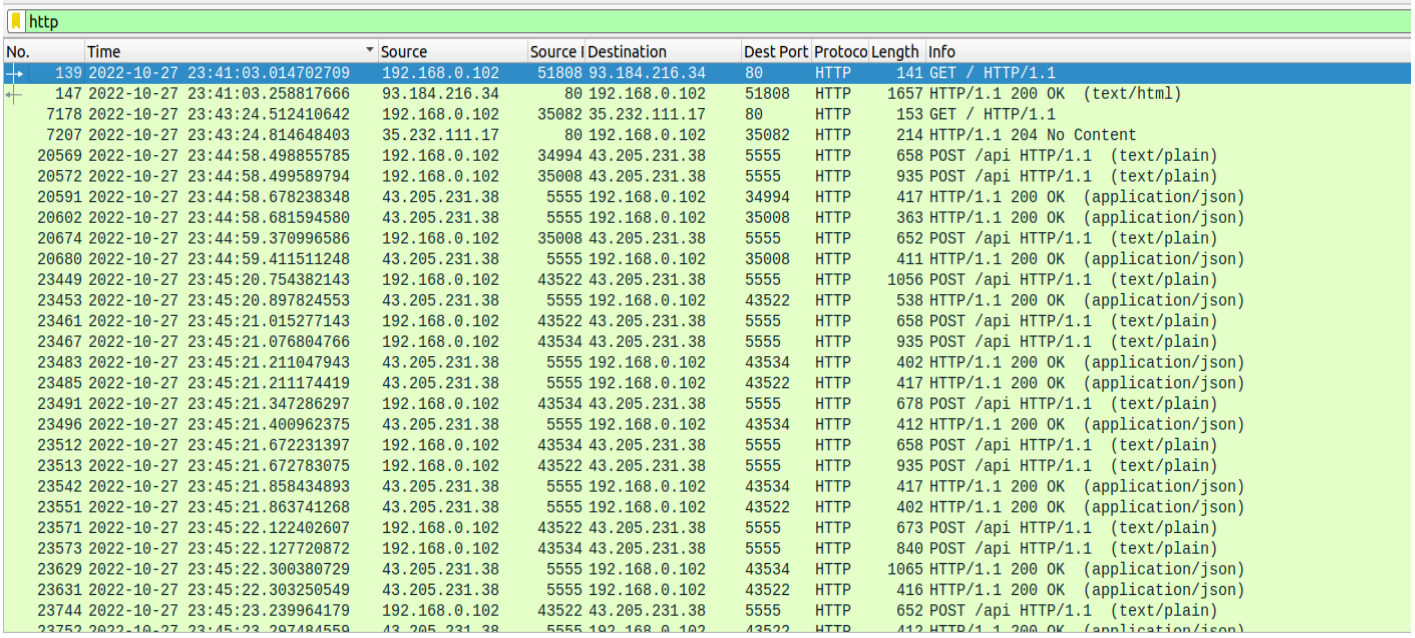


4. Find out the first 4 bytes of the Hex value of the Host parameter from the Packet Bytes Panel.

First 4 bytes are - 48 6f 73 74

5. Filter packets with http, TCP, DNS and other protocols. a. Find out what 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.

HTTP :



```
GET / HTTP/1.1
Host: example.com
User-Agent: curl/7.82.0
Accept: */*

HTTP/1.1 200 OK
Age: 202148
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Thu, 27 Oct 2022 18:11:03 GMT
Etag: "3147526947+ident"
Expires: Thu, 03 Nov 2022 18:11:03 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7F18)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-
serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
```

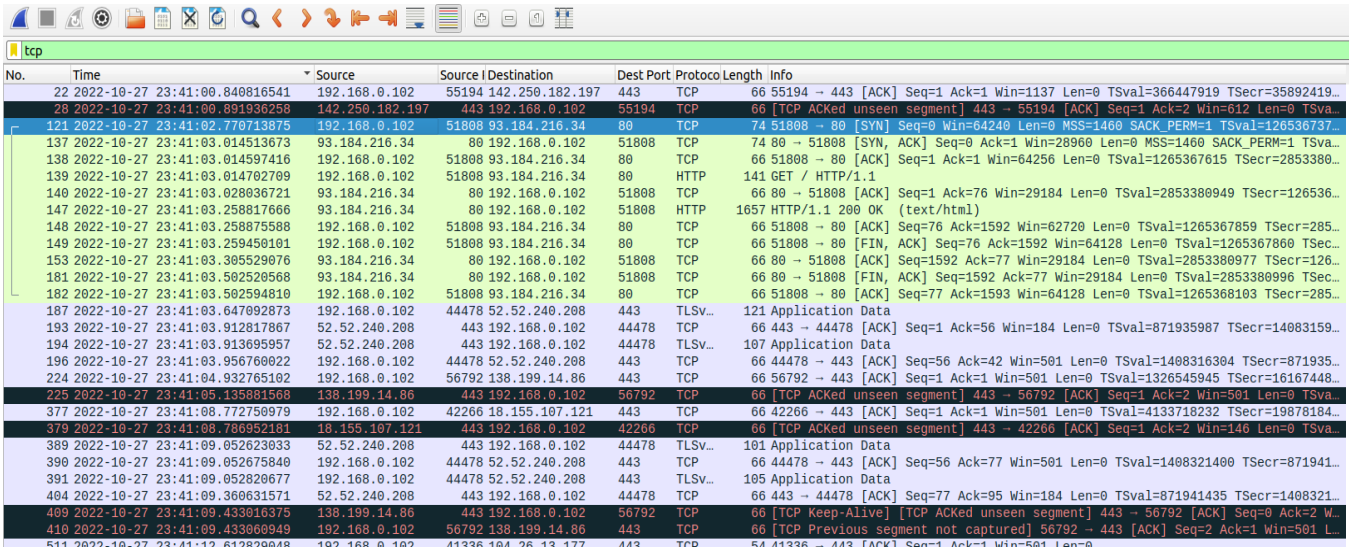
Packet 147. 1 client pkt, 1 server pkt, 1 turn. Click to select.

Entire conversation (1,666 bytes) | Show and save data as ASCII

Find: | Find Next

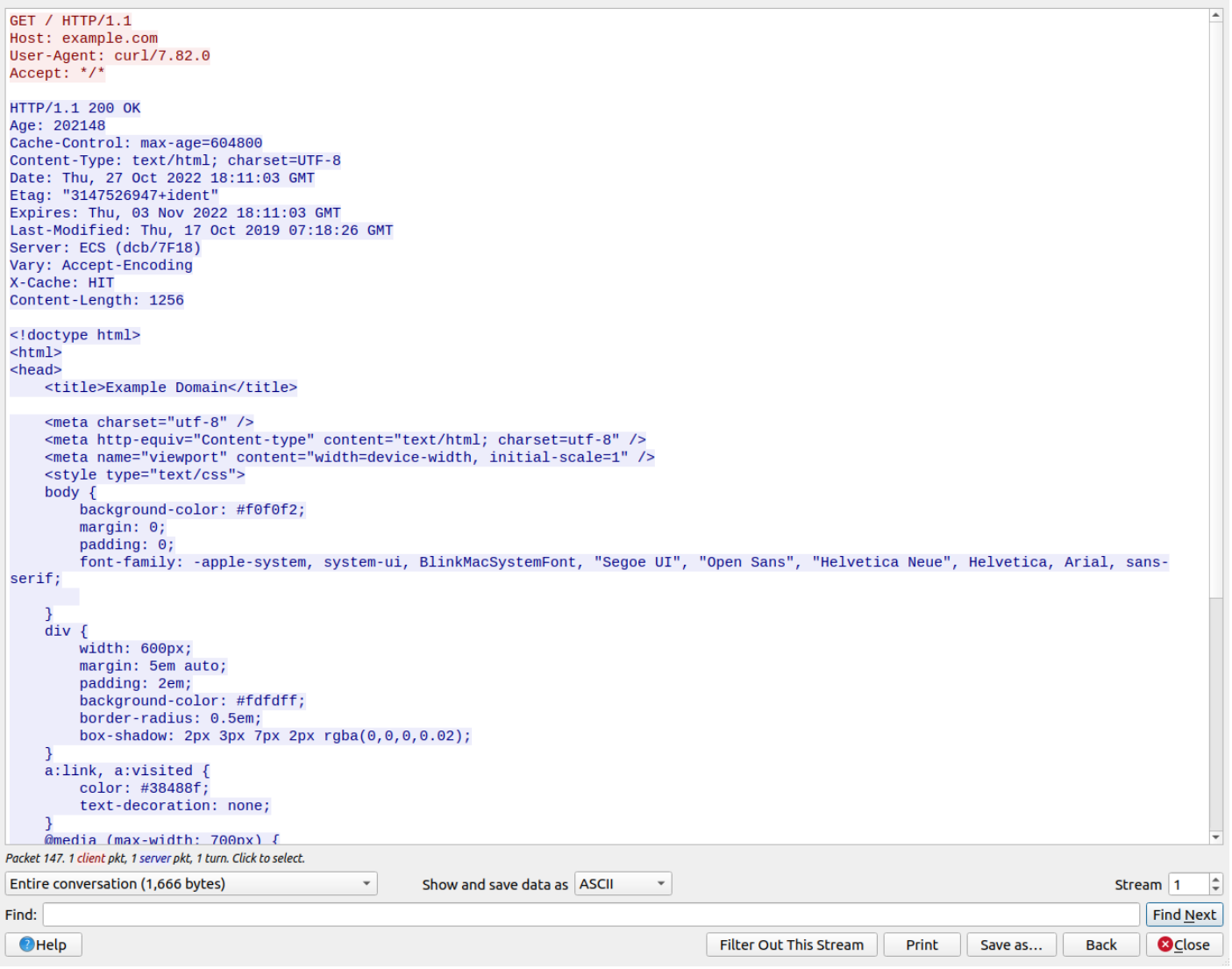
Help | Filter Out This Stream | Print | Save as... | Back | Close

TCP :



tcp

No.	Time	Source	Source I	Destination	Dest Port	Protoco	Length	Info
22	2022-10-27 23:41:00.840816541	192.168.0.102	55194	142.250.182.197	443	TCP	66	55194 → 443 [ACK] Seq=1 Ack=1 Win=1137 Len=0 TSval=366447919 TSecr=35892419…
28	2022-10-27 23:41:00.891936258	142.250.182.197	443	192.168.0.102	55194	TCP	66	[TCP ACKed unseen segment] 443 → 55194 [ACK] Seq=1 Ack=2 Win=612 Len=0 TSva…
121	2022-10-27 23:41:02.770713875	192.168.0.102	51808	93.184.216.34	80	TCP	74	51808 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=126536737…
137	2022-10-27 23:41:03.014513673	93.184.216.34	80	192.168.0.102	51808	TCP	74	80 → 51808 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSva…
138	2022-10-27 23:41:03.014597416	192.168.0.102	51808	93.184.216.34	80	TCP	66	51808 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1265367615 TSecr=2853380…
139	2022-10-27 23:41:03.014702709	192.168.0.102	51808	93.184.216.34	80	HTTP	141	GET / HTTP/1.1
140	2022-10-27 23:41:03.028036721	93.184.216.34	80	192.168.0.102	51808	TCP	66	80 → 51808 [ACK] Seq=1 Ack=76 Win=29184 Len=0 TSval=2853380949 TSecr=126536…
147	2022-10-27 23:41:03.258817666	93.184.216.34	80	192.168.0.102	51808	HTTP	1657	HTTP/1.1 200 OK (text/html)
148	2022-10-27 23:41:03.258875588	192.168.0.102	51808	93.184.216.34	80	TCP	66	51808 → 80 [ACK] Seq=76 Ack=1592 Win=62720 Len=0 TSval=1265367859 TSecr=285…
149	2022-10-27 23:41:03.259450101	192.168.0.102	51808	93.184.216.34	80	TCP	66	51808 → 80 [FIN, ACK] Seq=76 Ack=1592 Win=64128 Len=0 TSval=1265367860 TSec…
153	2022-10-27 23:41:03.305529076	93.184.216.34	80	192.168.0.102	51808	TCP	66	80 → 51808 [ACK] Seq=1592 Ack=77 Win=29184 Len=0 TSval=2853380977 TSecr=126…
181	2022-10-27 23:41:03.502520568	93.184.216.34	80	192.168.0.102	51808	TCP	66	80 → 51808 [FIN, ACK] Seq=1592 Ack=77 Win=29184 Len=0 TSval=2853380996 TSec…
182	2022-10-27 23:41:03.502594810	192.168.0.102	51808	93.184.216.34	80	TCP	66	51808 → 80 [ACK] Seq=77 Ack=1593 Win=64128 Len=0 TSval=1265368103 TSecr=285…
187	2022-10-27 23:41:03.647092873	192.168.0.102	44478	52.52.240.208	443	TLSv…	121	Application Data
193	2022-10-27 23:41:03.912817867	52.52.240.208	443	192.168.0.102	44478	TCP	66	443 → 44478 [ACK] Seq=1 Ack=56 Win=184 Len=0 TSval=871935987 TSecr=14083159…
194	2022-10-27 23:41:03.913695957	52.52.240.208	443	192.168.0.102	44478	TLSv…	107	Application Data
196	2022-10-27 23:41:03.956760022	192.168.0.102	44478	52.52.240.208	443	TCP	66	44478 → 443 [ACK] Seq=56 Ack=42 Win=501 Len=0 TSval=1408316304 TSecr=871935…
224	2022-10-27 23:41:04.932765102	192.168.0.102	56792	138.199.14.86	443	TCP	66	56792 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=1326545945 TSecr=16167448…
225	2022-10-27 23:41:05.135881568	138.199.14.86	443	192.168.0.102	56792	TCP	66	[TCP ACKed unseen segment] 443 → 56792 [ACK] Seq=1 Ack=2 Win=501 Len=0 TSva…
377	2022-10-27 23:41:08.772750979	192.168.0.102	42266	18.155.107.121	443	TCP	66	42266 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=4133718232 TSecr=19878184…
379	2022-10-27 23:41:08.786952181	18.155.107.121	443	192.168.0.102	42266	TCP	66	[TCP ACKed unseen segment] 443 → 42266 [ACK] Seq=1 Ack=2 Win=146 Len=0 TSva…
389	2022-10-27 23:41:09.052623033	52.52.240.208	443	192.168.0.102	44478	TLSv…	101	Application Data
390	2022-10-27 23:41:09.052675840	192.168.0.102	44478	52.52.240.208	443	TCP	66	44478 → 443 [ACK] Seq=56 Ack=77 Win=501 Len=0 TSval=1408321400 TSecr=871941…
391	2022-10-27 23:41:09.052820677	192.168.0.102	44478	52.52.240.208	443	TLSv…	105	Application Data
404	2022-10-27 23:41:09.360631571	52.52.240.208	443	192.168.0.102	44478	TCP	66	443 → 44478 [ACK] Seq=77 Ack=95 Win=184 Len=0 TSval=871941435 TSecr=1408321…
409	2022-10-27 23:41:09.433016375	138.199.14.86	443	192.168.0.102	56792	TCP	66	[TCP Keep-Alive] [TCP ACKed unseen segment] 443 → 56792 [ACK] Seq=0 Ack=2 W…
410	2022-10-27 23:41:09.433060949	192.168.0.102	56792	138.199.14.86	443	TCP	66	[TCP Previous segment not captured] 56792 → 443 [ACK] Seq=2 Ack=1 Win=501 L…
511	2022-10-27 23:41:12.612829048	192.168.0.102	41336	104.26.13.177	443	TCP	54	41336 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0



DNS :
As the local system has cached ip of example.com, it need no dns resolution , So DNS request has been tracked by Wireshark

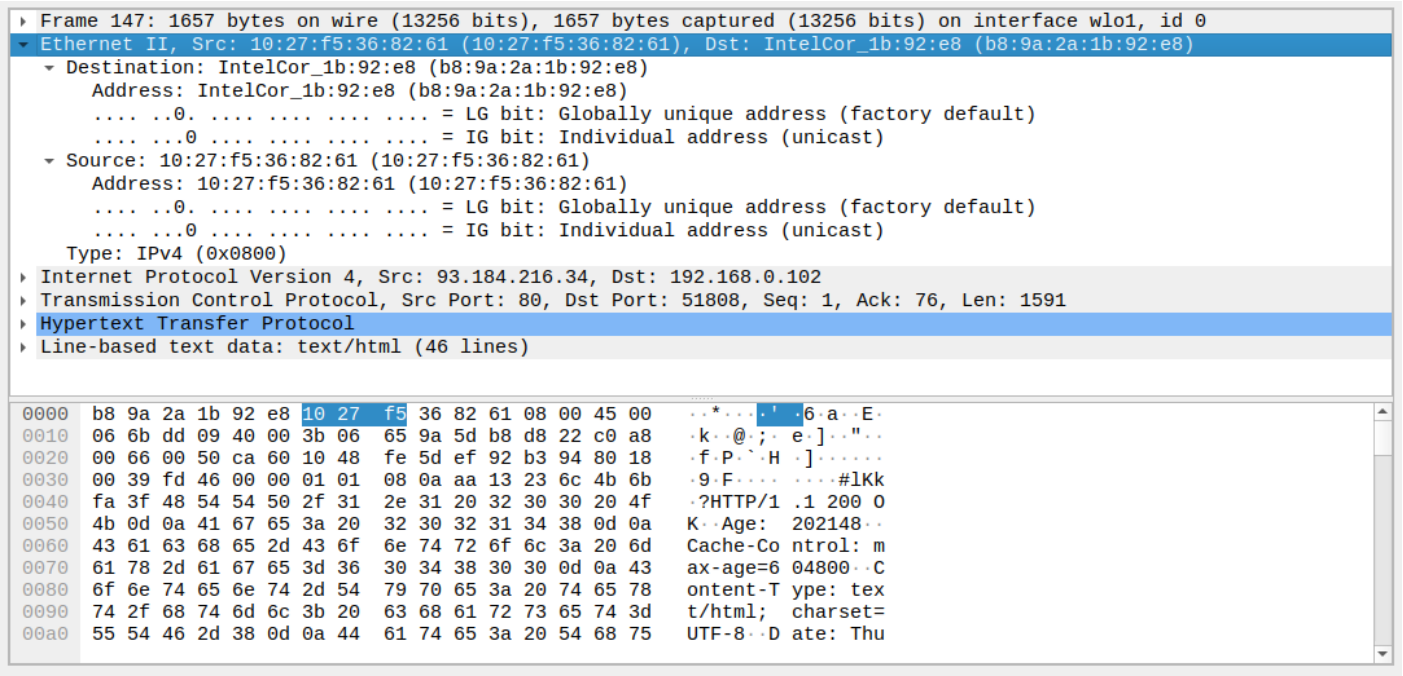
UDP :
As there is no DNS query for this particular request and the application need no additional UDP request , So nothing tracked in Wireshark

ARP :

arp

No.	Time	Source	Source I	Destination	Dest Port	Protoco	Length	Info
1659	2022-10-27 23:41:36.302046026	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
1660	2022-10-27 23:41:36.302065346	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
4122	2022-10-27 23:42:22.382523207	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
4123	2022-10-27 23:42:22.382544501	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
7505	2022-10-27 23:43:29.215336703	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
7506	2022-10-27 23:43:29.215388896	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
14063	2022-10-27 23:44:15.023737149	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
14064	2022-10-27 23:44:15.023750768	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
21260	2022-10-27 23:45:03.504256929	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
21261	2022-10-27 23:45:03.504267521	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
25471	2022-10-27 23:45:39.600639097	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
25472	2022-10-27 23:45:39.600657622	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
30371	2022-10-27 23:46:24.641135740	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
30372	2022-10-27 23:46:24.641159658	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
38893	2022-10-27 23:47:03.329616788	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
38894	2022-10-27 23:47:03.329634621	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
44813	2022-10-27 23:47:42.257944581	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
44814	2022-10-27 23:47:42.257968831	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
50026	2022-10-27 23:48:23.778404553	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
50027	2022-10-27 23:48:23.778436720	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
54681	2022-10-27 23:49:02.760005812	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
54682	2022-10-27 23:49:02.760031924	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
58983	2022-10-27 23:49:38.835183980	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
58984	2022-10-27 23:49:38.835195681	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
65598	2022-10-27 23:50:35.219742250	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1
65599	2022-10-27 23:50:35.219770288	IntelCor_1b:92…		10:27:f5:36:82:…		ARP	42	192.168.0.102 is at b8:9a:2a:1b:92:e8
78176	2022-10-27 23:50:57.171945354	10:27:f5:36:82…		IntelCor_1b:92:…		ARP	42	Who has 192.168.0.102? Tell 192.168.0.1

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.



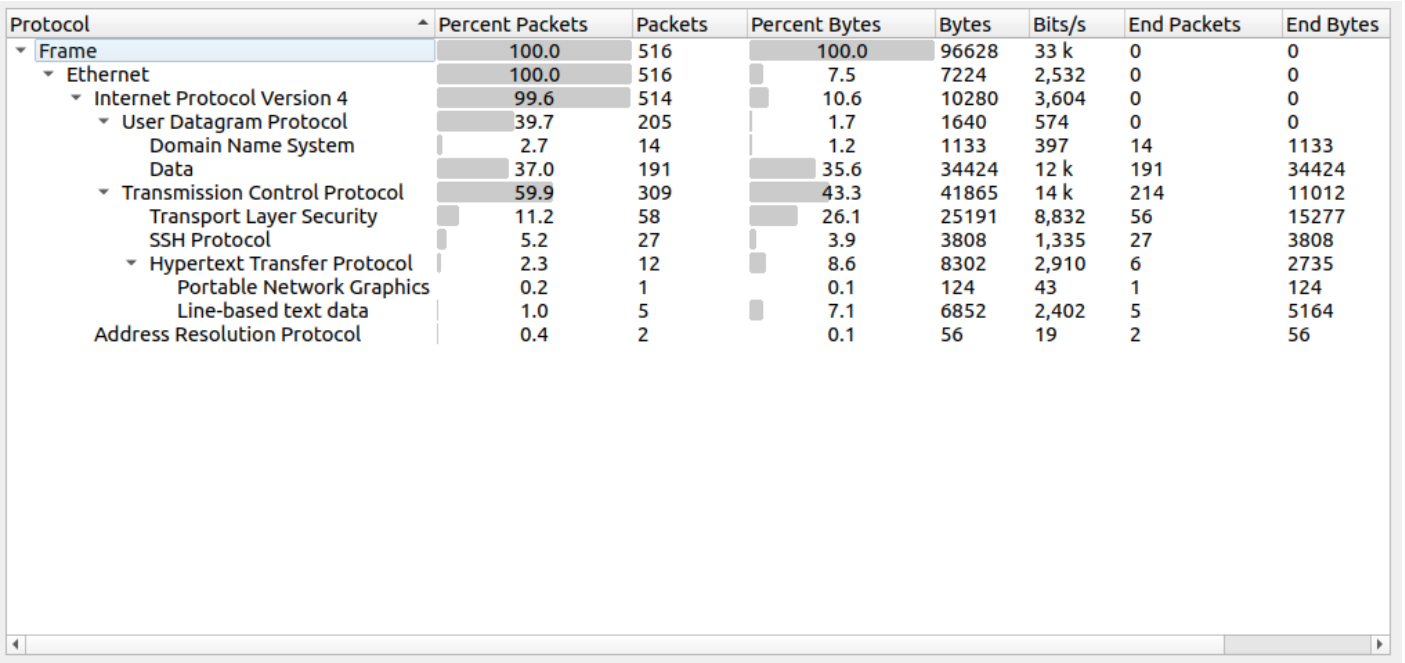
7. What are the manufacturers of your PC's Network Interface Card (NIC), and the servers NIC?

Manufacturer of my PC's NIC - Intel
Manufacturer of server's NIC - Can't be detected by Wireshark

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

Hex values of my PC's OUI - b8:9a:2a
Hex values of server's OUI - 10:27:f5

9. Find the following statistics:



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	516	100.0	96628	33 k	0	0
Ethernet	100.0	516	7.5	7224	2,532	0	0
Internet Protocol Version 4	99.6	514	10.6	10280	3,604	0	0
User Datagram Protocol	39.7	205	1.7	1640	574	0	0
Domain Name System	2.7	14	1.2	1133	397	14	1133
Data	37.0	191	35.6	34424	12 k	191	34424
Transmission Control Protocol	59.9	309	43.3	41865	14 k	214	11012
Transport Layer Security	11.2	58	26.1	25191	8,832	56	15277
SSH Protocol	5.2	27	3.9	3808	1,335	27	3808
Hypertext Transfer Protocol	2.3	12	8.6	8302	2,910	6	2735
Portable Network Graphics	0.2	1	0.1	124	43	1	124
Line-based text data	1.0	5	7.1	6852	2,402	5	5164
Address Resolution Protocol	0.4	2	0.1	56	19	2	56

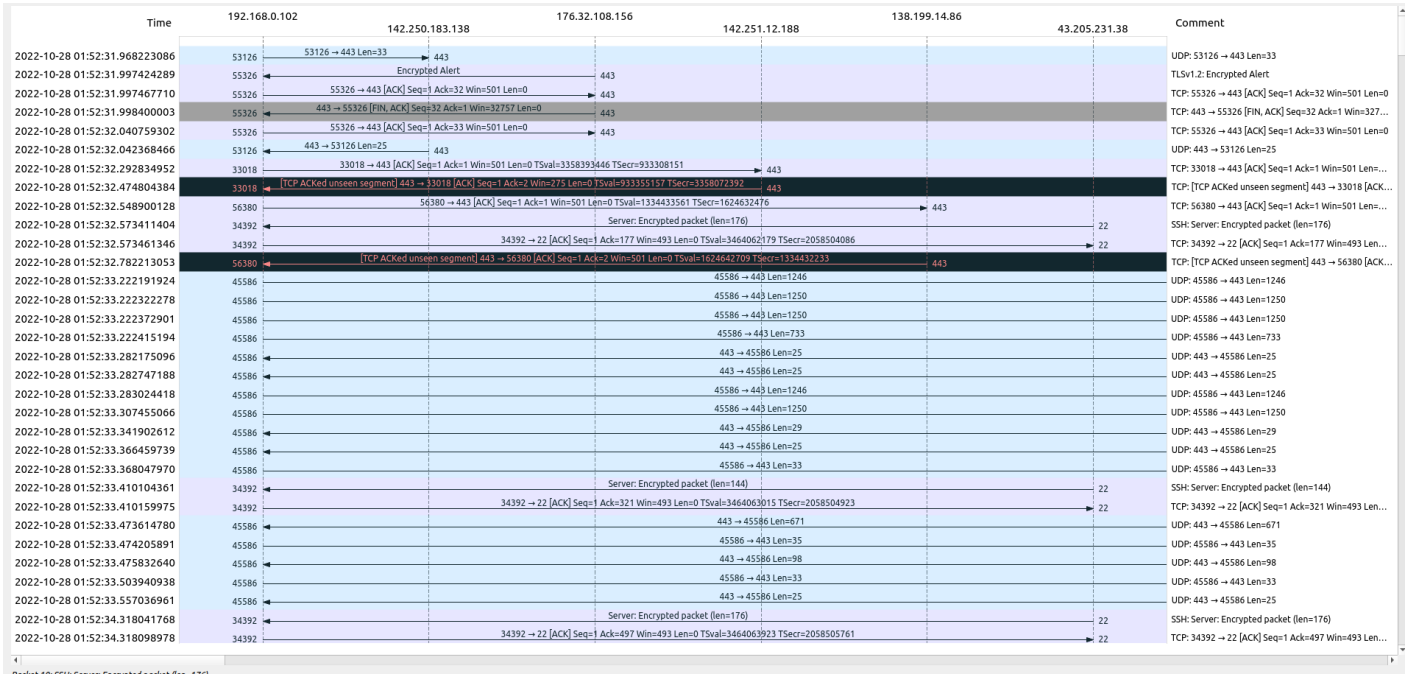
a. What percentage of packets in your capture are TCP, and give an example of the higher level protocol which uses TCP?

- 26.0 % of packets in the capture are TCP
- Higher level protocols using TCP - HTTP, FTP

b. What percentage of packets in your capture are UDP, and give an example of the higher level protocol which uses UDP?

- 73.6% packets are UDP
- Higher level protocols using UDP- SNMP

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

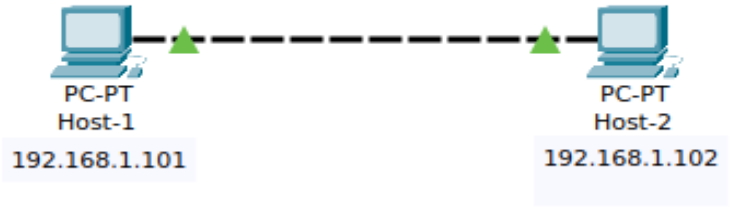

Time
192.168.0.102
142.250.183.138
176.32.108.156
142.251.12.188
138.199.14.86
43.205.231.38
Comment
2022-10-28 01:52:31.968223086 53126 53126 → 443 Len=33 443 UDP: 53126 → 443 Len=33
2022-10-28 01:52:31.997424289 55326 Encrypted Alert 443 TLSv1.2: Encrypted Alert
2022-10-28 01:52:31.997467710 55326 55326 → 443 [ACK] Seq=1 Ack=32 Win=501 Len=0 443 TCP: 55326 → 443 [ACK] Seq=1 Ack=32 Win=501 Len=0
2022-10-28 01:52:31.998400003 55326 443 → 55326 [FIN, ACK] Seq=32 Ack=1 Win=32757 Len=0 443 TCP: 443 → 55326 [FIN, ACK] Seq=32 Ack=1 Win=327…
2022-10-28 01:52:32.040759302 55326 55326 → 443 [ACK] Seq=1 Ack=33 Win=501 Len=0 443 TCP: 55326 → 443 [ACK] Seq=1 Ack=33 Win=501 Len=0
2022-10-28 01:52:32.042368466 53126 443 → 53126 Len=25 443 UDP: 443 → 53126 Len=25
2022-10-28 01:52:32.292834952 33018 33018 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=3358393446 TSecr=933308151 443 TCP: 33018 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=…
2022-10-28 01:52:32.474804384 33018 [TCP ACKed unseen segment] 443 → 33018 [ACK] Seq=1 Ack=2 Win=275 Len=0 TSval=933355157 TSecr=3358072392 443 TCP: [TCP ACKed unseen segment] 443 → 33018 [ACK…
2022-10-28 01:52:32.548900128 56380 56380 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=0 TSval=1334433561 TSecr=1624632476 443 TCP: 56380 → 443 [ACK] Seq=1 Ack=1 Win=501 Len=…
2022-10-28 01:52:32.573411404 34392 Server: Encrypted packet (len=176) 22 SSH: Server: Encrypted packet (len=176)
2022-10-28 01:52:32.573461346 34392 34392 → 22 [ACK] Seq=1 Ack=177 Win=493 Len=0 TSval=3464062179 TSecr=2058504086 22 TCP: 34392 → 22 [ACK] Seq=1 Ack=177 Win=493 Len…
2022-10-28 01:52:32.782213053 56380 [TCP ACKed unseen segment] 443 → 56380 [ACK] Seq=1 Ack=2 Win=501 Len=0 TSval=1624642709 TSecr=1334432233 443 TCP: [TCP ACKed unseen segment] 443 → 56380 [ACK…
2022-10-28 01:52:33.222191924 45586 45586 → 443 Len=1246 UDP: 45586 → 443 Len=1246
2022-10-28 01:52:33.222322278 45586 45586 → 443 Len=1250 UDP: 45586 → 443 Len=1250
2022-10-28 01:52:33.222372901 45586 45586 → 443 Len=1250 UDP: 45586 → 443 Len=1250
2022-10-28 01:52:33.222415194 45586 45586 → 443 Len=733 UDP: 45586 → 443 Len=733
2022-10-28 01:52:33.282175096 45586 443 → 45586 Len=25 UDP: 443 → 45586 Len=25
2022-10-28 01:52:33.282747188 45586 443 → 45586 Len=25 UDP: 443 → 45586 Len=25
2022-10-28 01:52:33.283024418 45586 45586 → 443 Len=1246 UDP: 45586 → 443 Len=1246
2022-10-28 01:52:33.307455066 45586 45586 → 443 Len=1250 UDP: 45586 → 443 Len=1250
2022-10-28 01:52:33.341902612 45586 443 → 45586 Len=29 UDP: 443 → 45586 Len=29
2022-10-28 01:52:33.366459739 45586 443 → 45586 Len=25 UDP: 443 → 45586 Len=25
2022-10-28 01:52:33.368047970 45586 45586 → 443 Len=33 UDP: 45586 → 443 Len=33
2022-10-28 01:52:33.410104361 34392 Server: Encrypted packet (len=144) 22 SSH: Server: Encrypted packet (len=144)
2022-10-28 01:52:33.410159975 34392 34392 → 22 [ACK] Seq=1 Ack=321 Win=493 Len=0 TSval=3464063015 TSecr=2058504923 22 TCP: 34392 → 22 [ACK] Seq=1 Ack=321 Win=493 Len…
2022-10-28 01:52:33.473614780 45586 443 → 45586 Len=671 UDP: 443 → 45586 Len=671
2022-10-28 01:52:33.474205891 45586 45586 → 443 Len=35 UDP: 45586 → 443 Len=35
2022-10-28 01:52:33.475832640 45586 443 → 45586 Len=98 UDP: 443 → 45586 Len=98
2022-10-28 01:52:33.503940938 45586 45586 → 443 Len=33 UDP: 45586 → 443 Len=33
2022-10-28 01:52:33.557036961 45586 443 → 45586 Len=25 UDP: 443 → 45586 Len=25
2022-10-28 01:52:34.318041768 34392 Server: Encrypted packet (len=176) 22 SSH: Server: Encrypted packet (len=176)
2022-10-28 01:52:34.318098978 34392 34392 → 22 [ACK] Seq=1 Ack=497 Win=493 Len=0 TSval=3464063923 TSecr=2058505761 22 TCP: 34392 → 22 [ACK] Seq=1 Ack=497 Win=493 Len…


Time
176.32.108.156
192.168.0.102
142.251.12.188
138.199.14.86
43.205.231.38
52.17.226.150
Comment
2022-10-28 01:52:31.997424289 443 PSH, ACK - Len: 31 55326 Seq = 1 Ack = 1
2022-10-28 01:52:31.997467710 443 ACK 55326 Seq = 1 Ack = 32
2022-10-28 01:52:31.998400003 443 FIN, ACK 55326 Seq = 32 Ack = 1
2022-10-28 01:52:32.040759302 443 ACK 55326 Seq = 1 Ack = 33
2022-10-28 01:52:32.292834952 33018 ACK 443 Seq = 1 Ack = 1
2022-10-28 01:52:32.474804384 33018 ACK 443 Seq = 1 Ack = 2
2022-10-28 01:52:32.548900128 56380 ACK 443 Seq = 1 Ack = 1
2022-10-28 01:52:32.573411404 34392 PSH, ACK - Len: 176 22 Seq = 1 Ack = 1
2022-10-28 01:52:32.573461346 34392 ACK 22 Seq = 1 Ack = 177
2022-10-28 01:52:32.782213053 56380 ACK 443 Seq = 1 Ack = 2
2022-10-28 01:52:33.410104361 34392 PSH, ACK - Len: 144 22 Seq = 177 Ack = 1
2022-10-28 01:52:33.410159975 34392 ACK 22 Seq = 1 Ack = 321
2022-10-28 01:52:34.318041768 34392 PSH, ACK - Len: 176 22 Seq = 321 Ack = 1
2022-10-28 01:52:34.318098978 34392 ACK 22 Seq = 1 Ack = 497
2022-10-28 01:52:35.137186280 34392 PSH, ACK - Len: 128 22 Seq = 497 Ack = 1
2022-10-28 01:52:35.137236253 34392 ACK 22 Seq = 1 Ack = 625
2022-10-28 01:52:35.353059889 443 FIN, ACK 55326 Seq = 1 Ack = 33
2022-10-28 01:52:35.353119475 34236 RST, ACK 443 Seq = 1 Ack = 1
2022-10-28 01:52:35.353164257 34230 RST, ACK 443 Seq = 1 Ack = 1
2022-10-28 01:52:35.503854890 47236 SYN Seq = 0
2022-10-28 01:52:35.503977436 47240 SYN Seq = 0
2022-10-28 01:52:35.504417778 47254 SYN Seq = 0
2022-10-28 01:52:35.537382073 443 ACK 55326 Seq = 33 Ack = 2
2022-10-28 01:52:35.603514901 47256 SYN Seq = 0
2022-10-28 01:52:35.604954650 47272 SYN Seq = 0
2022-10-28 01:52:35.854246061 47254 SYN, ACK Seq = 0 Ack = 1
2022-10-28 01:52:35.854338773 47254 ACK Seq = 1 Ack = 1
2022-10-28 01:52:35.854382040 47236 SYN, ACK Seq = 0 Ack = 1
2022-10-28 01:52:35.854410531 47236 ACK Seq = 1 Ack = 1
2022-10-28 01:52:35.854446817 47240 SYN, ACK Seq = 0 Ack = 1
2022-10-28 01:52:35.854471886 47240 ACK Seq = 1 Ack = 1
2022-10-28 01:52:35.854976725 47254 PSH, ACK - Len: 489 Seq = 1 Ack = 1

Name : Tanmoy Sarkar
Roll No : 002010501020
Class : BCSE III
Assignment No : 6
Subject : Computer Network
Group : A1

---

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



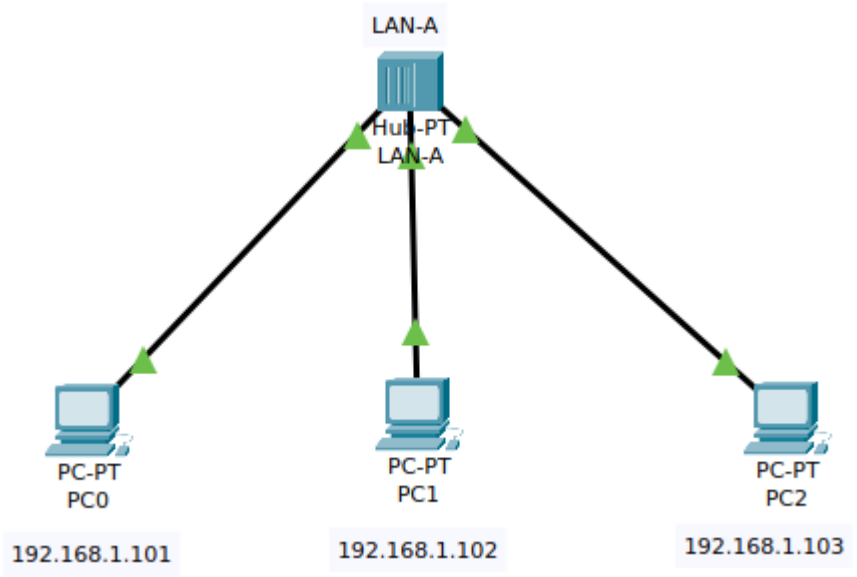
Host-1 pings Host-2

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.102

Pinging 192.168.1.102 with 32 bytes of data:

Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.102:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Host-2 pings Host-1

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.101

Pinging 192.168.1.101 with 32 bytes of data:

Reply from 192.168.1.101: bytes=32 time<1ms TTL=128
Reply from 192.168.1.101: bytes=32 time<1ms TTL=128
Reply from 192.168.1.101: bytes=32 time<1ms TTL=128
Reply from 192.168.1.101: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

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



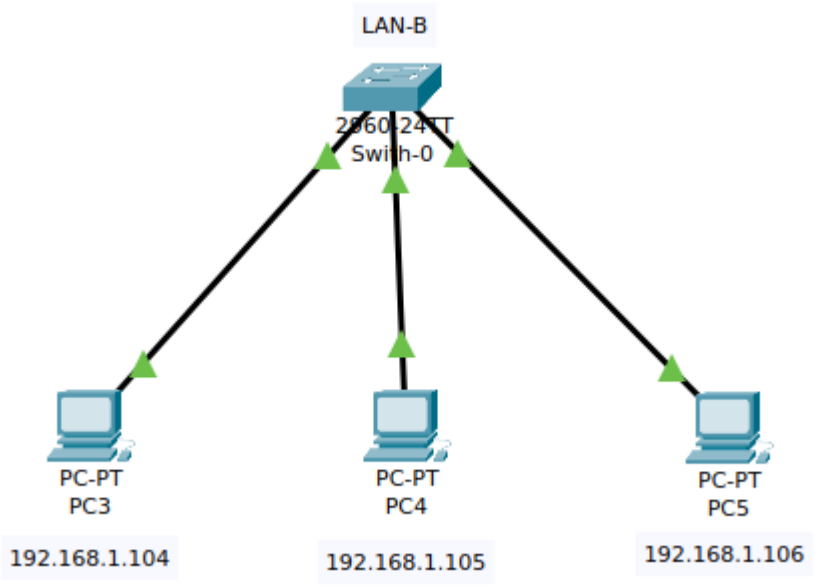
PC0 pings PC1

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.102

Pinging 192.168.1.102 with 32 bytes of data:

Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128
Reply from 192.168.1.102: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.102:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

PC0 pings PC2

```
C:\>ping 192.168.1.103

Pinging 192.168.1.103 with 32 bytes of data:

Reply from 192.168.1.103: bytes=32 time<1ms TTL=128
Reply from 192.168.1.103: bytes=32 time<1ms TTL=128
Reply from 192.168.1.103: bytes=32 time<1ms TTL=128
Reply from 192.168.1.103: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.103:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

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.



Before pinging

Arp table of PC3

```
C:\>arp -a
No ARP Entries Found
```

Arp table of PC4

```
C:\>arp -a
No ARP Entries Found
```

Arp table of PC5

```
C:\>arp -a
No ARP Entries Found
```

Arp table of Switch-0

```
Switch>show arp
```

PC3 will ping PC5

```
C:\>ping 192.168.1.106

Pinging 192.168.1.106 with 32 bytes of data:

Reply from 192.168.1.106: bytes=32 time<1ms TTL=128
Reply from 192.168.1.106: bytes=32 time<1ms TTL=128
Reply from 192.168.1.106: bytes=32 time<1ms TTL=128
Reply from 192.168.1.106: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.106:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

After pinging

ARP table of PC3

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.1.106         0001.c9e4.2512        dynamic
```

ARP table of PC4

```
C:\>arp -a
No ARP Entries Found
C:\>
```

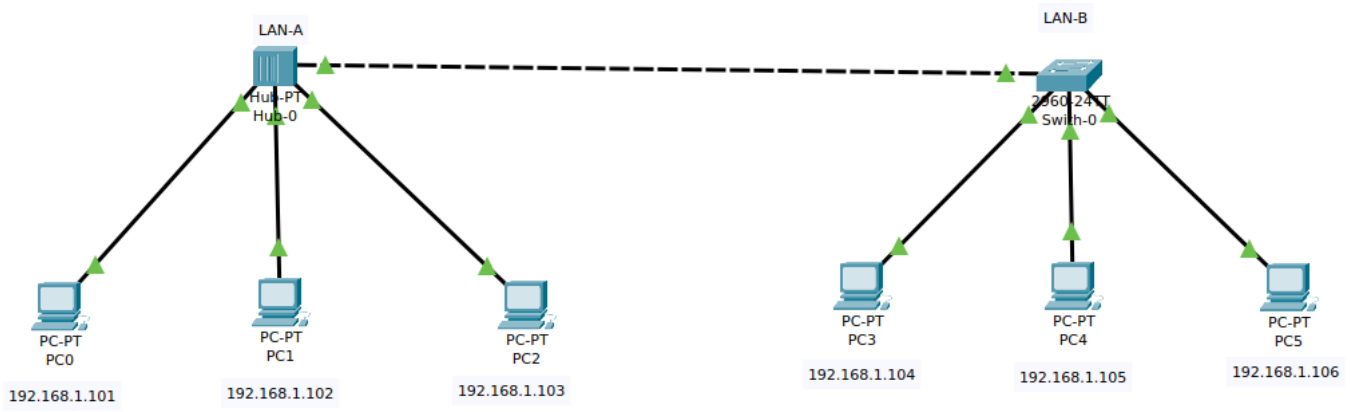
ARP table of PC5

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.1.104         0006.2a16.4617        dynamic
```

ARP table of Switch-0

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

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

   1    0001.c9e4.2512    DYNAMIC     Fa0/3
   1    0006.2a16.4617    DYNAMIC     Fa0/1
```

4. Connect LAN-A and LAN-B by connecting the hub and switch using a crossover 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.



PC0 pings PC5

```
C:\>ping 192.168.1.106

Pinging 192.168.1.106 with 32 bytes of data:

Reply from 192.168.1.106: bytes=32 time=20ms TTL=128
Reply from 192.168.1.106: bytes=32 time<1ms TTL=128
Reply from 192.168.1.106: bytes=32 time=28ms TTL=128
Reply from 192.168.1.106: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.106:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 28ms, Average = 12ms
```

ARP table for PC0

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.1.102         0004.9ae8.880c        dynamic
  192.168.1.103         0001.631a.7462        dynamic
  192.168.1.106         0001.c9e4.2512        dynamic
```

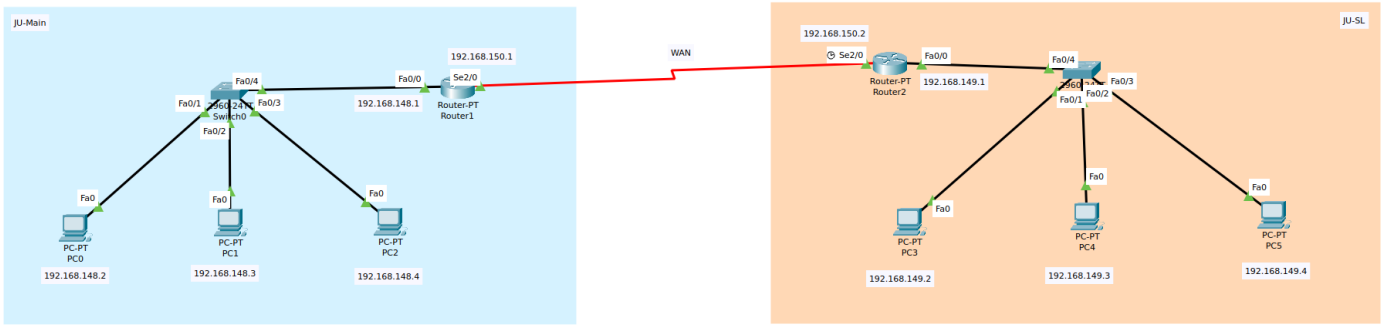
ARP table of PC5

```
C:\>arp -a
  Internet Address      Physical Address      Type
  192.168.1.101         0002.4aee.6722        dynamic
  192.168.1.104         0006.2a16.4617        dynamic
```

ARP table of Switch-0

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

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

   1    0001.c9e4.2512    DYNAMIC     Fa0/3
   1    0002.4aee.6722    DYNAMIC     Fa0/4
```

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 the default gateway of each host 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 the default gateway of each host 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.



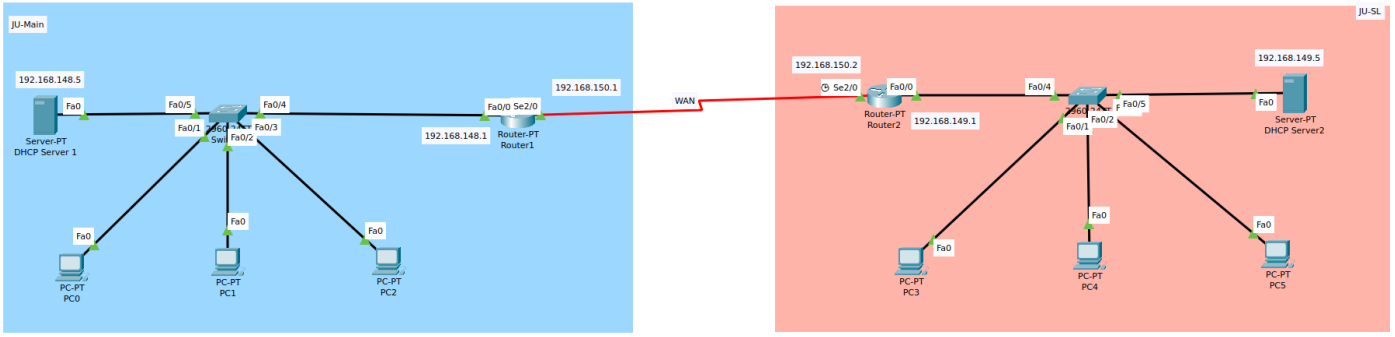
PC-0 and PC-1 is in JU-Main LAN
PC-0 pinging PC-1 : *Successful*

```
Pinging 192.168.148.3 with 32 bytes of data:

Reply from 192.168.148.3: bytes=32 time<1ms TTL=128
Reply from 192.168.148.3: bytes=32 time<1ms TTL=128
Reply from 192.168.148.3: bytes=32 time<1ms TTL=128
Reply from 192.168.148.3: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.148.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

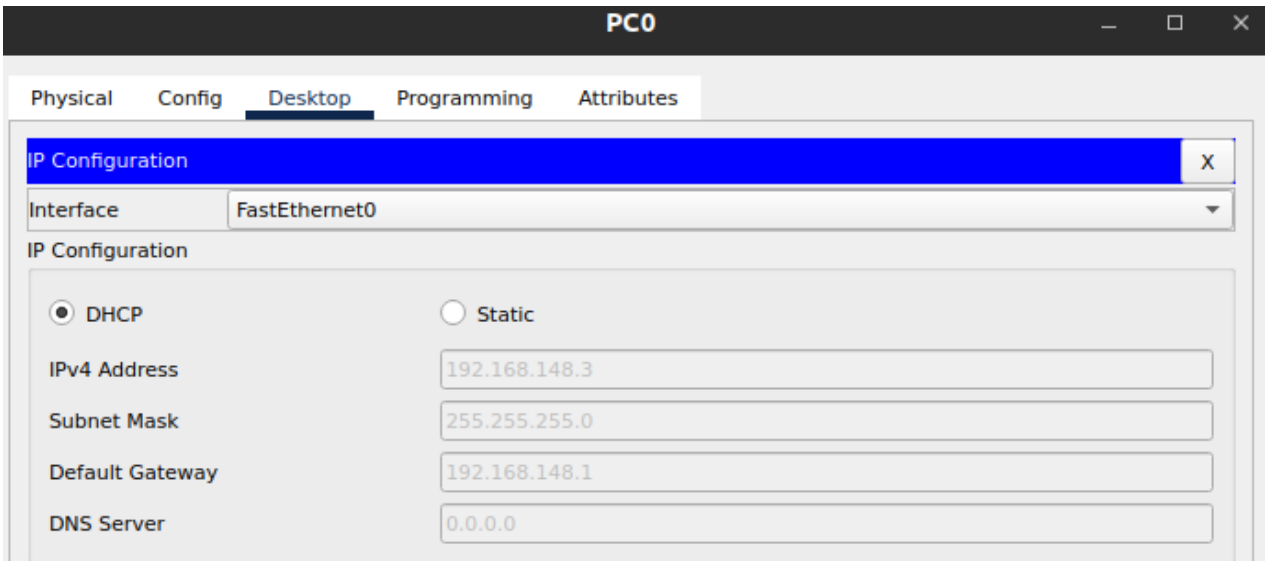
PC-0 is in JU-Main LAN and PC-4 is in JU-SL LAN
PC-0 [192.168.148.2] pinging PC-4[192.168.149.3] : *Successful*

```
Pinging 192.168.149.3 with 32 bytes of data:

Reply from 192.168.149.3: bytes=32 time=1ms TTL=126
Reply from 192.168.149.3: bytes=32 time=2ms TTL=126
Reply from 192.168.149.3: bytes=32 time=1ms TTL=126
Reply from 192.168.149.3: bytes=32 time=1ms TTL=126

Ping statistics for 192.168.149.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms
```

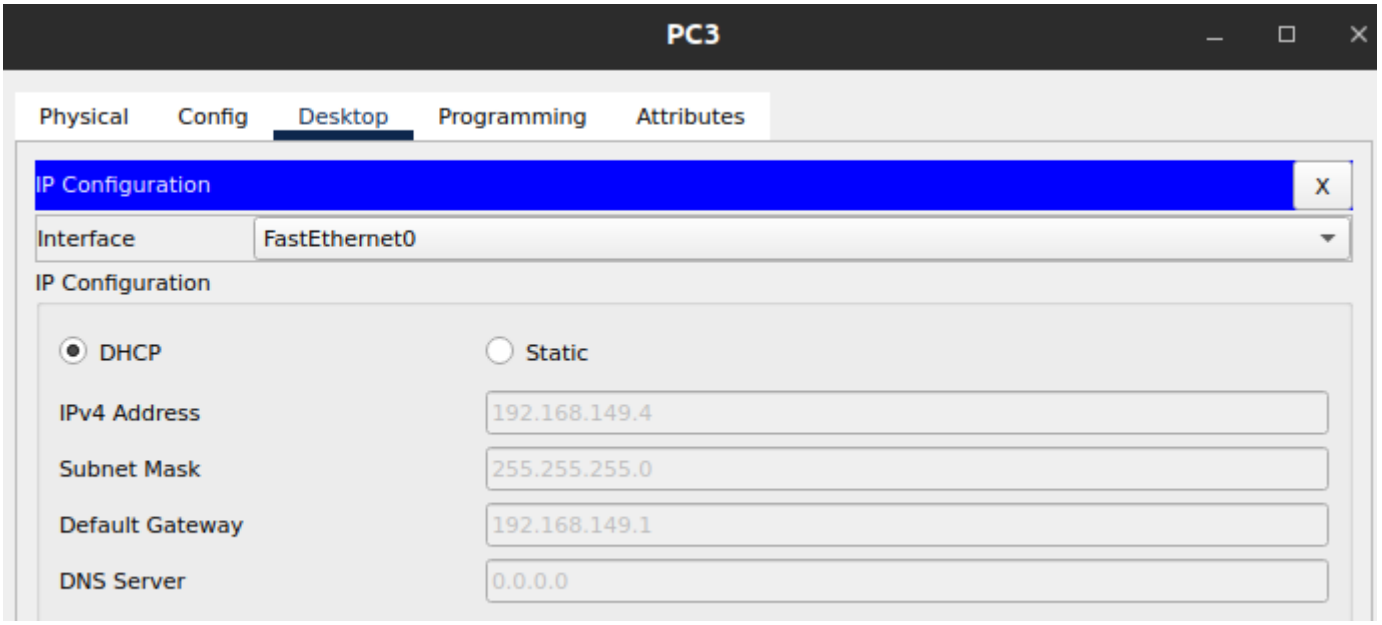
6. Add servers to the individual LANs (in problem 5) and configure them as a DHCP server. Configure the hosts in the individual LAN to obtain IP addresses and address of the default gateway via this DHCP server.



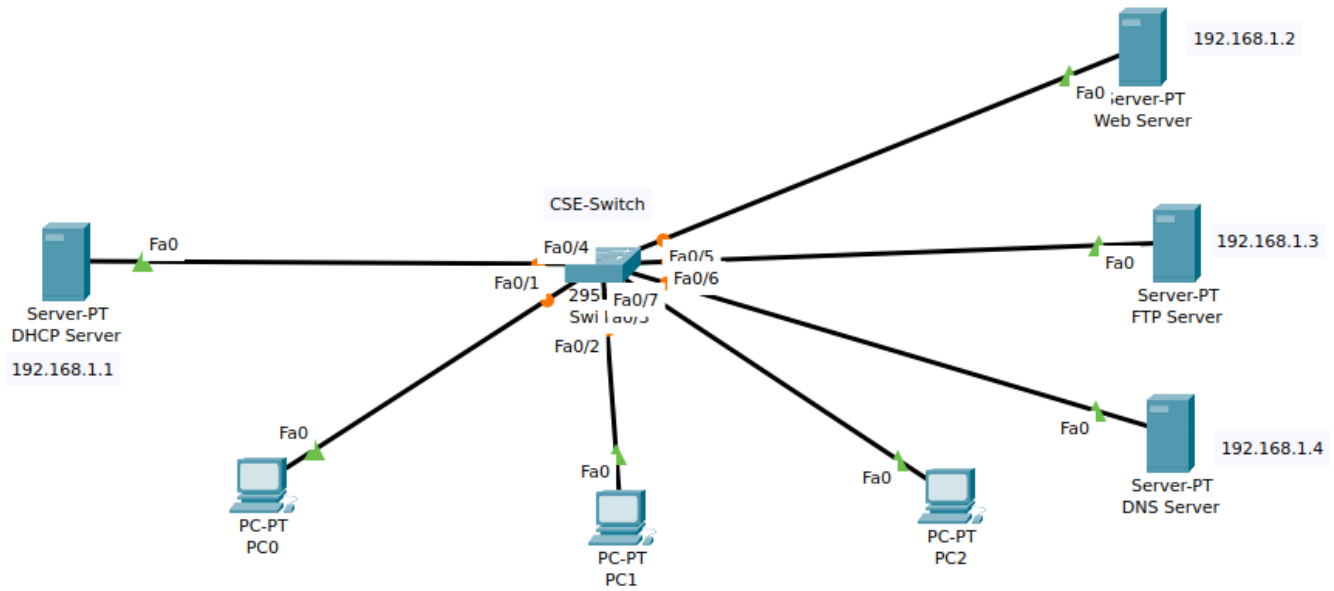
DHCP Server 1 is working. Here is a proof of working. DHCP Server has allocated ip 192.168.148.3 to PC0.



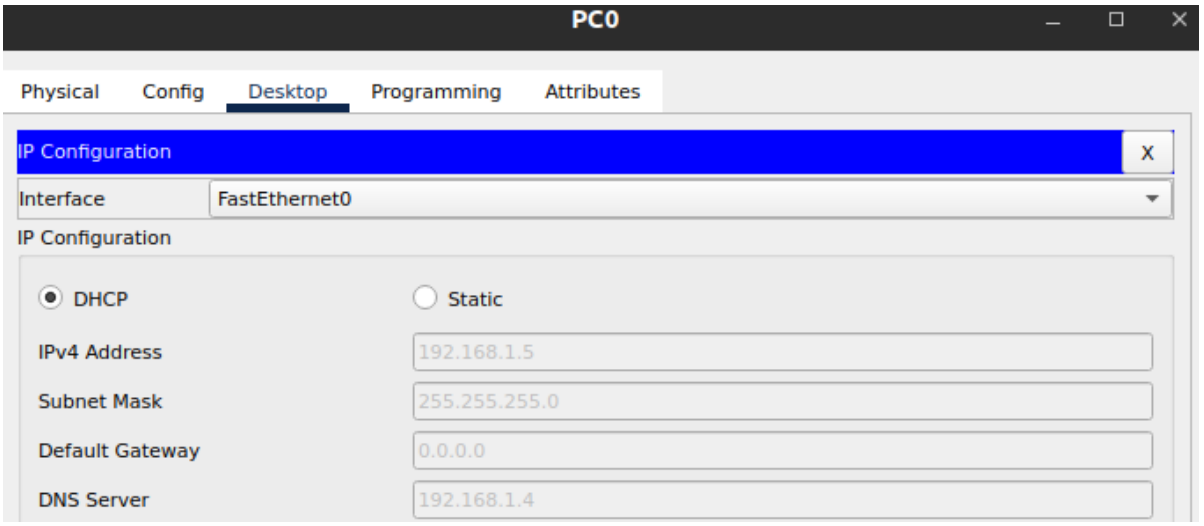
DHCP Server 2 is also working. DHCP Server 2 has allocated ip 192.168.149.4 to PC3.



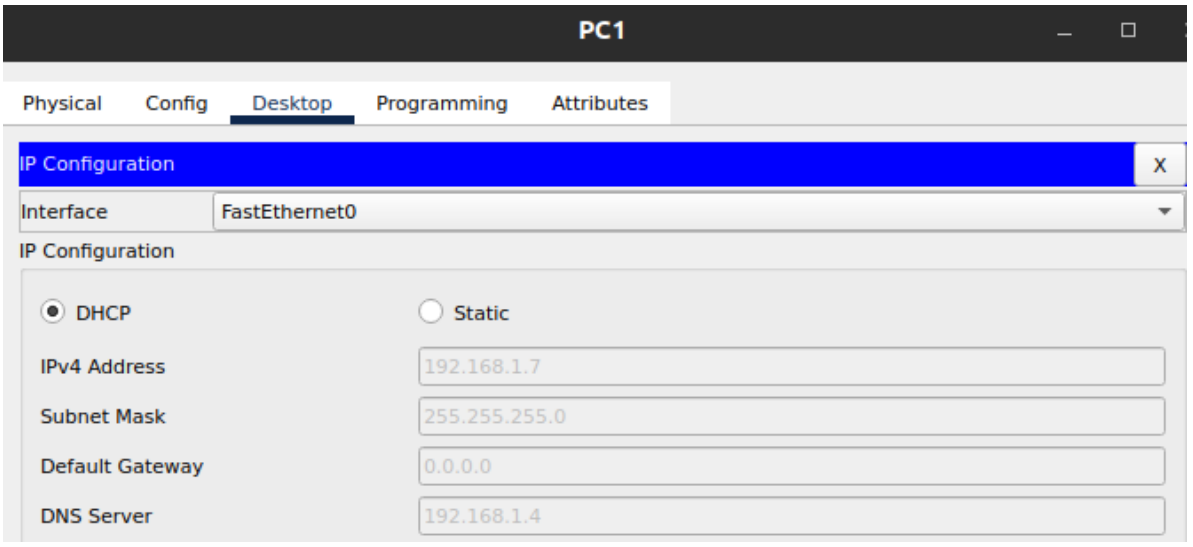
7. Create a LAN (CSE) with three hosts connected via a layer-2 switch (Cisco 2950 switch CSE-Switch). Also add a web server and a ftp server to this LAN. The hosts dynamically get their IP addresses from a local DHCP server. Servers are assigned fixed IP addresses. Configure the individual hosts to use 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.



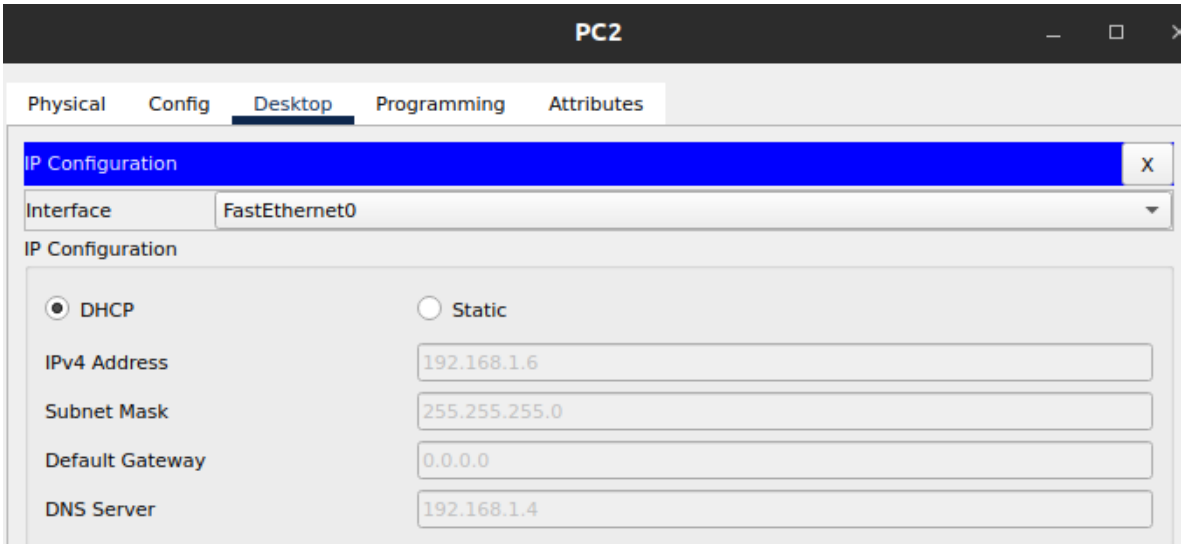
DHCP Server has allocated 192.168.1.5 to PC0



DHCP Server has allocated 192.168.1.7 to PC1

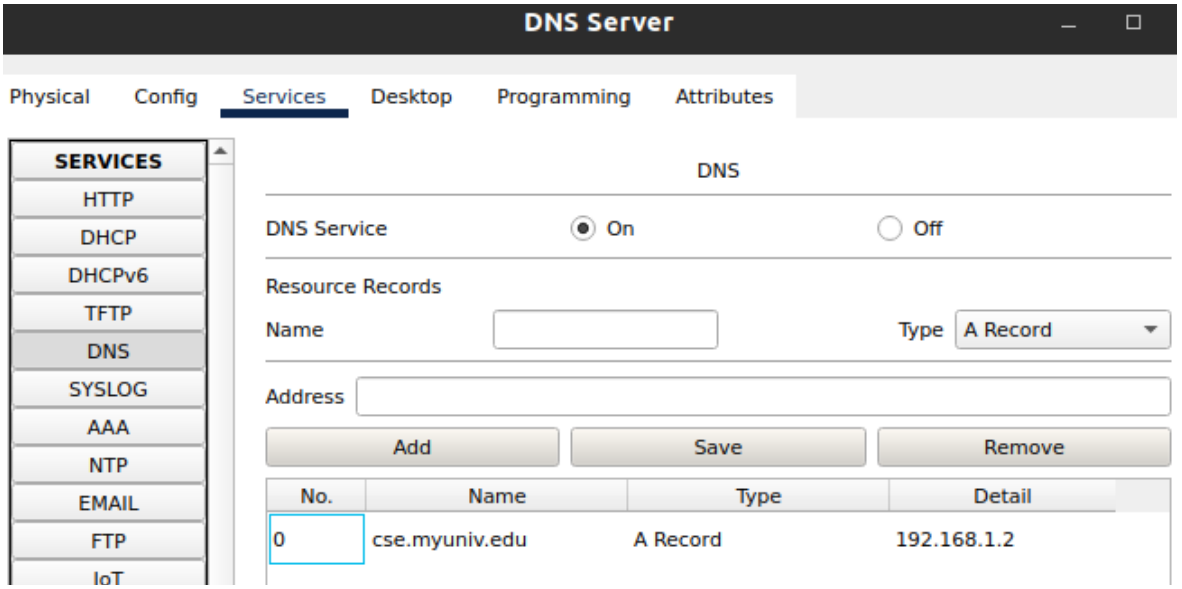


DHCP Server has allocated 192.168.1.5 to PC2

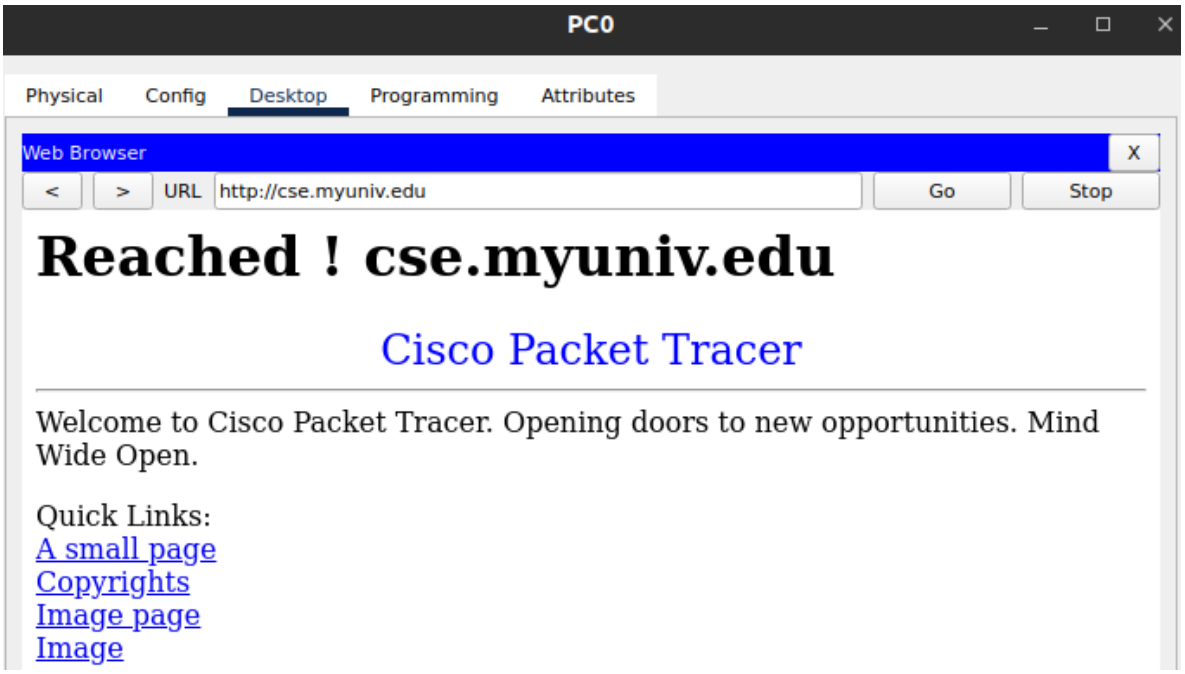


DNS Server : 192.168.1.4
FTP Server : 192.168.1.3
Web Server : 192.168.1.2

DNS Server Config



From PC0 accessing ***cse.myuniv.edu***



Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 7
Subject: Computer Network
Group: A1

ARP PROTOCOL

Full form of ARP is Address Resolution Protocol.

In a network, to communicate between two hosts, we need the MAC to address the NIC of the source and destination host. In a network initially, the hosts got a static IP or dynamic IP allocated by DHCP servers.

With only source and destination IPs, hosts can't communicate between them. Here the ARP protocol helps to translate the IP address to MAC.

How does it work?
-> Suppose,

	IP	MAC
Host A	192.168.1.1	1f:39:1b:71:4f:c9
Host B	192.168.1.2	d1:69:b7:c5:18:53

Situation: Host A wants to send a message to Host B, but Host A has not mac of Host B

Step A: Host A will create a packet, consists of
- IP of Host A
- MAC of Host A
- IP of Host B
- No MAC for Host B

and will broadcast the packet on the network.

Step B: All hosts will receive the packet, and Host B will detect that someone has a query for its MAC. Host B will form a packet, consisting of
- IP of Host B
- MAC of Host B
- IP of Host A
- MAC of Host A

and will broadcast the packet on the network.

Step C: All hosts in the network will receive the data, if the mac of that specific IP is not in the cache, the hosts will store the mac IP relation.

Step D: Host A now has the MAC of Host B and they can communicate between them.

Code Implementation -

```
from time import sleep
import socket
import random
import string
from threading import Thread

# Frame format
# IPSender:MACSender:ipRequested:macRequested

class Client:
    def __init__(self, ip, mac):
        # Set ip and mac of self
        self.ip = ip
        self.mac = mac
        # Create socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        # Bind to Client Port
        self.sock.bind(('', 1148))
        # Arp Table
        self.arp_table = {}

        print("Enter `arp` to show arp table")
        print("Enter IP to send ARP request")


    def startProcess(self):
        send_thread = Thread(target=self.sendMsg)
        receive_thread = Thread(target=self.receiveMsg)
        send_thread.start()
        receive_thread.start()
        send_thread.join()
        receive_thread.join()
        self.sock.close()
```

```
    def sendMsg(self):
        while True:
            ip = input()
            if ip == "arp":
                self.printArpTable()
            elif ip in self.arp_table:
                print("Cached MAC : " + self.arp_table[ip])
            else:
                self.sock.sendto((f"{self.ip}:{self.mac}:{ip}:").encode(),
('<broadcast>', 1148))

    def receiveMsg(self):
        while True:
            data, addr = self.sock.recvfrom(1024)
            msg = data.decode()
            msg_splitted = msg.split(":")

            ipSender = msg_splitted[0]
            macSender = msg_splitted[1]

            ipRequested = msg_splitted[2]
            macRequested = msg_splitted[3]

            # Store the sender ip and mac in arp table if not already stored
            if ipSender != self.ip:
                self.arp_table[ipSender] = macSender

            if ipRequested == self.ip:
                # Check whether the mac address is already present in frame
                if macRequested == "":

self.sock.sendto(f"{ipSender}:{macSender}:{self.ip}:{self.mac}".encode(),
('<broadcast>', 1148))
            elif macRequested not in self.arp_table:
                if macRequested != "":
                    self.arp_table[ipRequested] = macRequested

    def printArpTable(self):
        print("\nIP\tMAC")
        for ip, mac in self.arp_table.items():
            print(ip + "\t" + mac)
        print()


if __name__ == "__main__":
    # Create client
```

```
    ip = input("Enter IP: ")
    mac = input("Enter MAC (enter -1 for random mac): ")
    if mac == "-1":
        mac = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(12))
    client = Client(ip, mac)
    client.startProcess()
```

Conclusion:
The implementation of ARP Protocol bind itself to all network interfaces available and request MAC by directly multicast the requests to the network and caching all incoming responses and all stations maintain their ARP Table.

DHCP PROTOCOL

Full form of DHCP is Dynamic Host Configuration Protocol. It is based on the Bootstrap Protocol (BOOTP). This adds the capability of automatic allocation of reusable network addresses.

How does a station in the network get an IP address from DHCP Server?
There are various steps to acquire the IP.
Initially, the station will have an IP of 0.0.0.0

1. *DHCPDISCOVER*: Any station that needs IP, neither knows about the existence of the DHCP Server nor the IP of the DHCP Server. So the station will broadcast DHCPDISCOVER packet to the network.
2. *DHCPOFFER*: The DHCP Server will catch the DHCPDISCOVER packet that was relayed by the station. The DHCP server will offer the station an IP to acquire and will broadcast in the network.
3. *DHCPREQUEST* : The station will receive the DHCPOFFER and if everything looks fine, will request DHCP Server to assign the IP.
4. *DHCPACK:* After receiving DHCPREQUEST, the DHCP server will check again the availability of the specified IP and if available will send DHCPACK mentioning the confirmed IP address and leased time.
5. *DHCPNAK*: If IP in DHCPREQUEST is not available, DHCP Server will send *DHCPNAK*

After receiving DHCPACK, the station has got an IP address for a specific leased time. After ending of the leased time station will request the DHCP server for reallotment of the IP by the DHCPREQUEST packet.

Station and DHCP server decide the type of packet by a byte in packet, `message type`

Packet Name	Message Type
DHCPDISCOVER	1
DHCPOFFER	2
DHCPREQUEST	3

DHCPACK	5
DHCPNAK	6

Code Implementation -

Helper class [Frame] to encode and decode frame

```
# Frame format
# Request Type [1-> request, 2-> reply] | Message Type | Transaction ID | Client MAC
Address | Client IP Address | Lease Time | Subnet Mask | Gateway IP | DNS IP | Host
Name

class Frame:
    def __init__(self):
        self.requestType = ""
        self.messageType = ""
        self.transactionID = ""
        self.clientMAC = ""
        self.clientIP = ""
        self.leaseTime = ""
        self.subnetMask = ""
        self.gatewayIP = ""
        self.dnsIP = ""
        self.hostName = ""

    @staticmethod
    def decode(frameString):
        frame = Frame()
        frameSplitted = frameString.split("|")
        frame.requestType = frameSplitted[0]
        frame.messageType = frameSplitted[1]
        frame.transactionID = frameSplitted[2]
        frame.clientMAC = frameSplitted[3]
        frame.clientIP = frameSplitted[4]
        frame.leaseTime = frameSplitted[5]
        frame.subnetMask = frameSplitted[6]
        frame.gatewayIP = frameSplitted[7]
        frame.dnsIP = frameSplitted[8]
        frame.hostName = frameSplitted[9]

        return frame

    def encode(self):
        return str(self.requestType) + "|" + str(self.messageType) + "|" +
str(self.transactionID) + "|" + str(self.clientMAC) + "|" + str(self.clientIP) + "|" +
```

```
str(self.leaseTime) + "|" + str(self.subnetMask) + "|" + str(self.gatewayIP) + "|" +
str(self.dnsIP) + "|" + str(self.hostName)


if __name__ == "__main__":
    frame = Frame()
    s = frame.encode()
    print(s)
    x = s.split("|")
    print(x)
    print(len(x))
```

Code for DHCP Client [Station in network]

```
from time import sleep
import socket
import random
import string
from frame import Frame
from threading import Thread

class DhcpClient:
    def __init__(self, mac, host_name, dhcp_client_port=1168, dhcp_server_port=1167):
        self.mac = mac
        self.ip = '0.0.0.0'
        self.subnet_mask = '0.0.0.0'
        self.dns_ip = '0.0.0.0'
        self.gateway_ip = '0.0.0.0'
        self.lease_time = 0
        self.host_name = host_name
        self.dhcp_client_port = dhcp_client_port
        self.dhcp_server_port = dhcp_server_port

        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Bind to Client Port
        self.sock.bind(('', self.dhcp_client_port))

        # Transaction ID
        self.transacionID = "-1"
```

```
    def broadcast(self, message):
        self.sock.sendto(message.encode(), ("<broadcast>", self.dhcp_server_port))

    def receiveMsg(self):
        data, addr = self.sock.recvfrom(2048)
        return data.decode()

    def startProcess(self):
        # Generate transaction id
        self.transacionID = ''.join(random.choices(string.ascii_lowercase +
string.digits, k=8))
        receive_thread = Thread(target=self.receive_data)
        receive_thread.start()
        sleep(0.01)
        self.send_dhcp_discover()
        receive_thread.join()



    def send_dhcp_discover(self):
        # Send DHCP Discover
        frame = Frame()
        frame.requestType = 1
        frame.messageType = 1
        frame.transactionID = self.transacionID
        frame.clientMAC = self.mac
        frame.clientIP = '0.0.0.0'
        frame.hostName = self.host_name
        self.broadcast(frame.encode())

    def send_dhcp_request(self):
        frame = Frame()
        frame.requestType = 1
        frame.messageType = 3
        frame.transactionID = self.transacionID
        frame.clientMAC = self.mac
        frame.clientIP = self.ip
        frame.hostName = self.host_name
        self.broadcast(frame.encode())

    def receive_data(self):
        while True:
            receieved_frame_raw = self.receiveMsg()
            receieved_frame = Frame.decode(receieved_frame_raw)
            # DHCP OFFER
            if receieved_frame.requestType == "2" and receieved_frame.transactionID ==
```

```
self.transacionID and receieved_frame.messageType == "2" and receieved_frame.clientMAC
== self.mac:
                self.ip = receieved_frame.clientIP
                # Send DHCP Request
                self.send_dhcp_request()

            # DHCP ACK
            if receieved_frame.requestType == "2" and receieved_frame.transactionID ==
self.transacionID  and receieved_frame.clientMAC == self.mac:
                if receieved_frame.messageType == "5":
                    self.ip = receieved_frame.clientIP
                    self.subnet_mask = receieved_frame.subnetMask
                    self.gateway_ip = receieved_frame.gatewayIP
                    self.dns_ip = receieved_frame.dnsIP
                    self.lease_time = int(receieved_frame.leaseTime)
                    self.printDetails()
                    lease_thread = Thread(target=self.handleLeaseTime)
                    lease_thread.start()
                elif receieved_frame.messageType == "6":
                    # NACK
                    print("DHCP NAK received from DHCP server. Listening again ...")


    def handleLeaseTime(self):
        sleep(self.lease_time+0.1)
        print("Lease time expired. Listening again ...")
        self.send_dhcp_request()

    def printDetails(self):
        print("-----------------------------------------------")
        print("Host        : ", self.host_name)
        print("IP          : ", self.ip)
        print("MAC         : ", self.mac)
        print("Subnet mask : ", self.subnet_mask)
        print("Gateway IP  : ", self.gateway_ip)
        print("DNS         : ", self.dns_ip)
        print("Lease time  : ", self.lease_time)
        print("-----------------------------------------------")


    def close(self):
        self.sock.close()


# Steps
# 1. DHCP Discover
```

```
# 2. DHCP Request

if __name__ == "__main__":
   mac_id = "02:00:00:%02x:%02x:%02x" % (random.randint(0, 255),random.randint(0,
255), random.randint(0, 255))
   print("MAC ID: ", mac_id)
   client_name = "client-" + mac_id
   client = DhcpClient(mac_id, client_name)
   client.startProcess()
   client.close()
```

Code for DHCP Server

```
from re import S
import socket
import random
import string
from threading import Thread
from time import sleep
from turtle import pen
from frame import Frame

class DhcpServer:
    def __init__(self, mac, starting_ip="192.168.0.2", subnet_mask="255.255.255.0",
dhcp_client_port=1168, dhcp_server_port=1167):
        self.mac = mac
        self.dhcp_client_port = dhcp_client_port
        self.dhcp_server_port = dhcp_server_port

        self.dns_ip = "8.8.8.8"
        self.subnet_mask = subnet_mask
        self.gateway_ip = "192.168.0.1"

        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Bind to Client Port
        self.sock.bind(('', self.dhcp_server_port))

        self.starting_ip = [int(x) for x in starting_ip.split(".")]
        self.subnet_mask = [int(x) for x in subnet_mask.split(".")]

        # Transaction ID for DHCP
        self.transaction_id_offerred_ip = {}
        self.available_ips = set()

        # Lease time table
        self.lease_time_table = {}


    def broadcast(self, message):
        self.sock.sendto(message.encode(), ("<broadcast>", self.dhcp_client_port))

    def receiveMsg(self):
```

```
        data, addr = self.sock.recvfrom(1024)
        return data.decode()

    def startProcess(self):
        expiry_thread = Thread(target=self.timeoutExpiry)
        expiry_thread.start()
        while True:
            request_frame = Frame.decode(self.receiveMsg())
            if request_frame.requestType == "1":
                if request_frame.messageType == "1":
                    # Wait for DHCP Discover request
                    print("DHCP Discover request received from ",
request_frame.clientMAC)
                    # Generate IP
                    ip = self.generateNextIP()
                    self.transaction_id_offerred_ip[request_frame.transactionID] = ip

                    # DHCP OFFER
                    offer = Frame()
                    offer.requestType = "2"
                    offer.messageType = "2"
                    offer.clientMAC = request_frame.clientMAC
                    offer.clientIP = ip
                    offer.transactionID = request_frame.transactionID
                    offer.hostName = request_frame.hostName
                    # sleep(2)
                    print("Sending DHCP Offer to ", request_frame.clientMAC)
                    self.broadcast(offer.encode())

                    # Wait for DHCP Request
                    request = Frame.decode(self.receiveMsg())
                    if request.requestType == "1" and request.messageType == "3":
                        print("DHCP Request received from ", request.clientMAC)
                        # sleep(2)
                        # Check if transaction id is same
                        if request.transactionID in self.transaction_id_offerred_ip:
                            # Check if IP is same
                            if self.transaction_id_offerred_ip[request.transactionID]
== request.clientIP and (request.clientIP in self.available_ips or request.clientIP
not in self.lease_time_table):
                                # DHCP ACK
                                ack = Frame()
                                ack.requestType = "2"
                                ack.messageType = "5"
                                ack.clientMAC = request.clientMAC
                                ack.clientIP = request.clientIP
```

```
                                ack.hostName = request.hostName
                                ack.transactionID = request.transactionID
                                ack.dnsIP = self.dns_ip
                                ack.subnetMask = '.'.join([str(x) for x in
self.subnet_mask])
                                ack.gatewayIP = self.gateway_ip
                                ack.leaseTime = "10"
                                self.broadcast(ack.encode())
                                # Delete from available ips
                                if request.clientIP in self.available_ips:
                                    self.available_ips.remove(request.clientIP)
                                self.lease_time_table[request.clientIP] = 10
                                print("DHCP ACK sent to ", request.clientMAC)
                            else:
                                print("IP address not same as offerred")
                        else:
                            print("Transaction ID not same as offerred")
                elif request_frame.messageType == "3":
                    # DHCP ACK
                    ack = Frame()
                    ack.requestType = "2"
                    ack.messageType = "5"
                    ack.clientMAC = request.clientMAC
                    ack.clientIP = request.clientIP
                    ack.hostName = request.hostName
                    ack.transactionID = request.transactionID
                    ack.dnsIP = self.dns_ip
                    ack.subnetMask = '.'.join([str(x) for x in self.subnet_mask])
                    ack.gatewayIP = self.gateway_ip
                    ack.leaseTime = "10"
                    self.broadcast(ack.encode())
                    # Delete from available ips
                    if request.clientIP in self.available_ips:
                        self.available_ips.remove(request.clientIP)
                    self.lease_time_table[request.clientIP] = 10

    def timeoutExpiry(self):
        while True:
            ips_need_to_be_removed = []
            for ip, time in self.lease_time_table.items():
                if time <= 0:
                    ips_need_to_be_removed.append(ip)
                    print("IP address released ", ip)
                else:
                    self.lease_time_table[ip] -= 1
```

```
            for ip in ips_need_to_be_removed:
                del self.lease_time_table[ip]
                self.available_ips.add(ip)
            sleep(1)

    def generateIP(self):
        return ".".join([str(x) for x in self.starting_ip])

    def generateNextIP(self):
        # TODO do according to subnet mask
        self.starting_ip[3] += 1
        return ".".join([str(x) for x in self.starting_ip])

    def close(self):
        self.sock.close()


if __name__ == "__main__":
    server = DhcpServer("02:02:01:02:02:00")
    server.startProcess()
    server.close()
```

Conclusion:
Basic features of DHCP protocol has been implemented and it can provide stations in the network an IP address. Some advanced packet format like DHCPREALEASE, DHCPINFORM, and DHCPDECLINE is not implemented for this scope.

Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 8
Subject: Computer Network
Group: A1

DNS Protocol

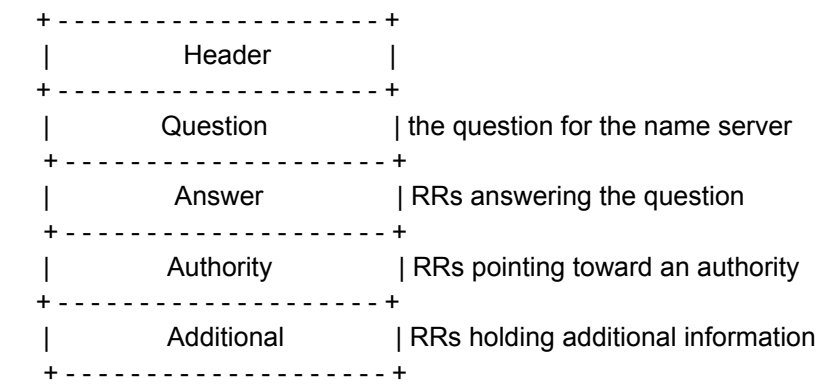
Overview : On the internet for communication between client and server , client requires server/host's IP address, but in this crowded network it's not possible to remember Ip of every host to connect to it. So there comes the domain name [example.com] . It's much easier to remember domain names rather than IP addresses. But the client in the lower lever requires an IP address to communicate. Here the DNS come. Clients can request DNS to send the domain's IP address, so that client can use the IP to connect.

How it works :
DNS protocol works over UDP protocol.

- Client send and dns request consists of domain name, type of query [A, NS, CNAME, MX, TXT etc.], query class type [usually it is IN (internet)].
- The DNS server will check for the domain in the zone records .
- If DNS has zone records of that domain, will send the Answers of the query.
- If there is no zone record for that domain, it will send the dns request to top level TLD root servers and from there the request will be send to the latest nameservers of that domain . The nameserver will forward the result to that client.

Implementation :
DNS protocol and its packet encoding and decoding have been implemented as per specified in RFC 1035.

DNS request has this structure



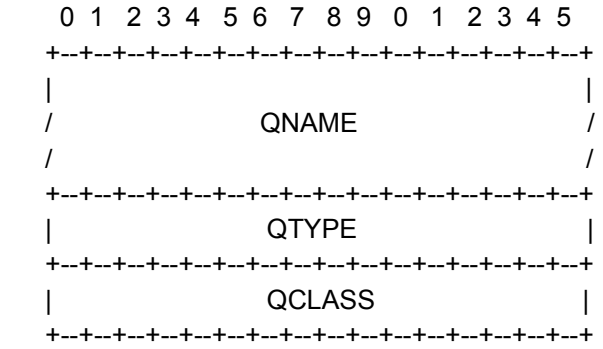
For a basic DNS server, the implementation of the Header, Question and Answer section is mandatory.

Header section consists of

- 16 byte of transaction ID (to track the requests between client , dns server and multiple root servers)
- A one bit field that specifies whether this message is a query (0), or a response (1).
- OPCODE : A four bit field that specifies a kind of query in this message.
    - 0 - Standard Query
    - 1 - an inverse query
    - 2 - a server status request

- AA : Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.
- TC : TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
- QDCOUNT : an unsigned 16 bit integer specifying the number of entries in the question section.
- ANCOUNT : an unsigned 16 bit integer specifying the number of resource records in the answer section.
- NSCOUNT : an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.
- ARCOUNT : an unsigned 16 bit integer specifying the number of resource records in the additional records section.
- Other parameters also there Recursion Available [RA], Response Code [RCODE]

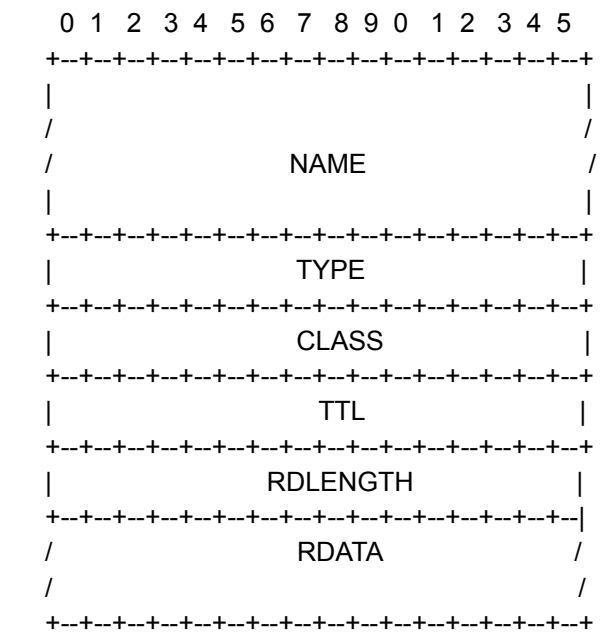
Question section format



It's consists of :

- QNAME : Domain name of query [ex. Google.com]. It's variable length
- QTYPE : An numeric value which represents Record Type. [A, CNAME, MX, TXT, AAAA]
- QCLASS : A two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.

Answer Section Format :

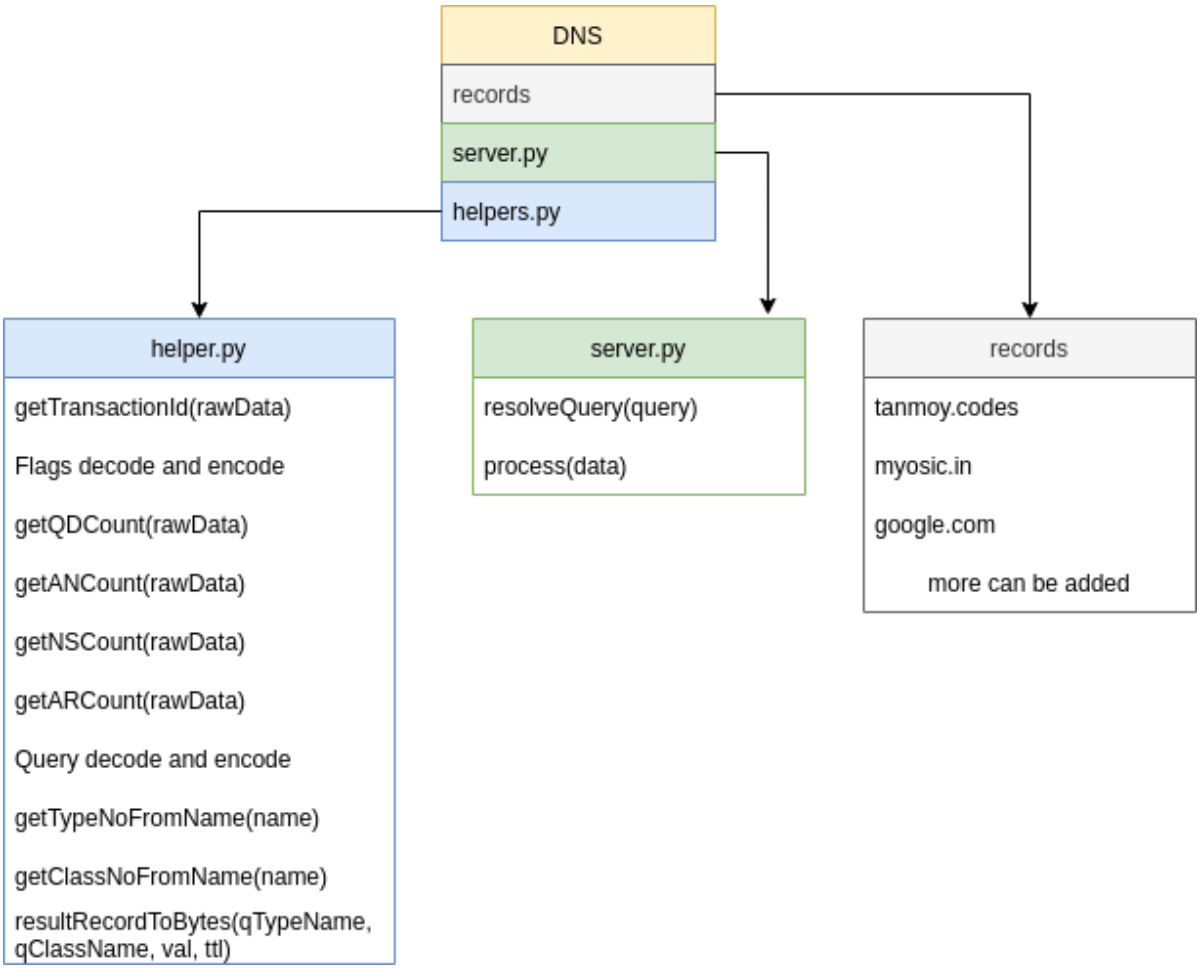


It consists of :

- NAME : a domain name to which this resource record pertains.
- TYPE : Type of record [A, NS, CNAME, MX, TXT etc.]

- CLASS : A two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet
- TTL : a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded
- RDLENGTH : an unsigned 16 bit integer that specifies the length in octets of the RDATA field.
- RDATA : a variable length string of octets that describes the resource.

Folder Structure :



Code Implementation -

server.py

```
import socket
from helpers import *
import  os

UDP_IP_ADDRESS = "127.0.0.1"
UDP_PORT_NO = 53


serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSock.bind((UDP_IP_ADDRESS, UDP_PORT_NO))
```

```
def resolveQuery(query:Query):
   record_path = os.path.join(os.path.dirname(__file__), "records",
query.baseDomainName())
   # If the file exists, then we have a record for this domain
   if os.path.exists(record_path):
       answers = []
       nameservers = []
       # Read the file and parse the answers
       with open(record_path, "r") as f:
           for curLine in f:
               curLine = curLine.strip() # Strip to remove newlines
               curLineParts = curLine.split("\t")
               if len(curLineParts) < 3: continue
               if curLineParts[0] == query.domainName(endDot=True) and curLineParts[1]
== query.queryClass():
                   if curLineParts[2] == query.queryType():
                       answers.append(curLine)
                   if curLineParts[2] == "NS":
                       nameservers.append(curLine)
       return (True, answers, nameservers)

   # else , we have no record for this domain
   return (False, [], [])


def process(data):
   transactionID = getTransactionID(data)
   flags = Flags.fromBytes(data, debug=False)
   QDCount = getQDCount(data) # Usually 1
   ANCount = getANCount(data)
   NSCount = getNSCount(data)
   # ARCount = getARCount(data)
   ARCount = 0
   query = Query.fromBytes(data)

   print("Transaction ID: " + transactionID)

   #  Find the record for this query
   result = resolveQuery(query)

   if result[0]:
       transactionID = data[:2]
       QDCount = (QDCount).to_bytes(2, "big")
       ANCount = len(result[1]).to_bytes(2, "big")
       NSCount = len(result[2]).to_bytes(2, "big")
       ARCount = (0).to_bytes(2, "big")
   else:
       transactionID = data[:2]
       QDCount = (QDCount).to_bytes(2, "big")
       ANCount = (0).to_bytes(2, "big")
```

```
        NSCount = (0).to_bytes(2, "big")
        ARCount = (0).to_bytes(2, "big")

    # Build the response
    # Header
    responseHeader = transactionID + flags.toBytes() + QDCount + ANCount + NSCount +
ARCount
    print("Response Header: " + str(responseHeader))

    # Question
    responseQuestion = query.toBytes()
    print("Response Question: " + str(responseQuestion))

    # Body
    responseBody = b""
    for record in result[1]:
        record_split = record.split("\t")
        # Name | Class | Type | Val | TTL
        responseBody += resultRecordToBytes(record_split[2], record_split[1],
record_split[3], int(record_split[4]))

    return responseHeader + responseQuestion + responseBody

while True:
    data, addr = serverSock.recvfrom(512)
    serverSock.sendto(process(data), addr)
```

helpers.py

```
### Required datas
queryTypesName = ["A", "NS", "MD", "MF", "CNAME", "SOA", "MB", "MG", "MR", "NULL", "WKS",
"PTR", "HINFO", "MINFO", "MX", "TXT"]
queryClassesName = ["IN", "CS", "CH", "HS"]

### Functions
# Get Transaction ID from data
def getTransactionID(rawData:bytes):
    transactionIDRaw = rawData[:2]
    transactionID = ""
    for x in transactionIDRaw:
        transactionID += hex(x)[2:]
    return transactionID

# Get flags from data
class Flags:
    def __init__(self, QR, OPCODE, AA, TC, RD, RA, Z, RCODE):
        self.QR = QR
        self.OPCODE = OPCODE
        self.AA = AA
        self.TC = TC
```

```
        self.RD = RD
        self.RA = RA
        self.Z = Z
        self.RCODE = RCODE

    def __str__(self):
        return str(self.QR) + " " + str(self.OPCODE) + " " + str(self.AA) + " " +
str(self.TC) + " " + str(self.RD) + " " + str(self.RA) + " " + str(self.Z) + " " +
str(self.RCODE)

    @staticmethod
    def fromBytes(rawData:bytes, debug=False):
        #       first byte       |second byte
        # | 1 |  4   | 1 | 1 | 1 | 1 | 3|  4  |
        # | QR|Opcode| AA| TC| RD| RA| Z|RCODE|

        first_byte = ord(rawData[2:3])
        second_byte = ord(rawData[3:4])

        if debug:
            print("[DEBUG] First byte: " + str(first_byte)+" "+bin(first_byte))
            print("[DEBUG] Second byte: " + str(second_byte)+" "+bin(second_byte))

        QR = first_byte >> 7
        OPCODE = (first_byte  & 0b01111000 ) >> 3
        AA = (first_byte & 0b00000100) >> 2
        TC = (first_byte & 0b00000010) >> 1
        RD = (first_byte & 0b00000001)

        RA = second_byte >> 7
        Z = (second_byte & 0b1110000) >> 4
        RCODE = second_byte & 0b1111

        if debug:
            print("[DEBUG] QR: " + str(QR))
            print("[DEBUG] OPCODE: " + str(OPCODE))
            print("[DEBUG] AA: " + str(AA))
            print("[DEBUG] TC: " + str(TC))
            print("[DEBUG] RD: " + str(RD))
            print("[DEBUG] RA: " + str(RA))
            print("[DEBUG] Z: " + str(Z))
            print("[DEBUG] RCODE: " + str(RCODE))

        return Flags(QR, OPCODE, AA, TC, RD, RA, Z, RCODE)

    def toBytes(self):
        first_byte = (self.QR << 7) + (self.OPCODE << 3) + (self.AA << 2) + (self.TC << 1)
+ self.RD
        second_byte = (self.RA << 7) + (self.Z << 4) + self.RCODE
        return first_byte.to_bytes(1, "big") + second_byte.to_bytes(1, "big")

# Get QD count from data
def getQDCount(rawData:bytes):
```

```
    QDCountRaw = rawData[4:6]
    QDCount = ""
    for x in QDCountRaw:
        QDCount += hex(x)[2:]
    return int(QDCount)

# Get AN count from data
def getANCount(rawData:bytes):
    ANCountRaw = rawData[6:8]
    ANCount = ""
    for x in ANCountRaw:
        ANCount += hex(x)[2:]
    return int(ANCount)

# Get NS count from data
def getNSCount(rawData:bytes):
    NSCountRaw = rawData[8:10]
    NSCount = ""
    for x in NSCountRaw:
        NSCount += hex(x)[2:]
    return int(NSCount)

# Get AR count from data
def getARCount(rawData:bytes):
    ARCountRaw = rawData[10:12]
    ARCount = ""
    for x in ARCountRaw:
        ARCount += hex(x)[2:]
    return int(ARCount)

# Query Questions parser
class Query:
    def __init__(self, nameParts, type, qclass):
        self.nameParts = nameParts
        self.type = type
        self.qclass = qclass

    def baseDomainName(self):
        return self.nameParts[-2] + "." + self.nameParts[-1]

    def domainName(self, endDot=False):
        res = '.'.join(self.nameParts)
        if endDot and res[-1] != ".":
            res += "."
        return res

    def queryType(self):
        if self.type > 16:
            return "Unknown"
        return queryTypesName[self.type - 1]

    def queryClass(self):
        if self.qclass > 4:
```

```
            return "Unknown"
        return queryClassesName[self.qclass - 1]

    def __str__(self):
        return self.domainName() + " " + str(self.type) + " " + str(self.qclass)

    @staticmethod
    def fromBytes(rawData:bytes, debug=False):
        data = rawData[12:]
        if debug:
            print("[DEBUG] Data: ")
            print(data)
        # Format : {length of characters + [string]} untill 0x00

        domainParts = []

        totalLengthOfDomainPartsWithLengthCharacter = 0
        length = -1
        tmp = ""
        for byte in data:
            totalLengthOfDomainPartsWithLengthCharacter += 1
            if byte == 0: # End of parts
                if tmp != "":
                    domainParts.append(tmp)
                break

            if length == -1: # Yet not started, so first byte is length
                length = byte
            elif length == 0: # Read one part, so next byte is length
                domainParts.append(tmp)
                tmp = ""
                length = byte
            else: # Its part of domain name
                tmp += chr(byte)
                length -= 1

        tmp =
data[totalLengthOfDomainPartsWithLengthCharacter:totalLengthOfDomainPartsWithLengthCharact
er+4]
        # QType
        QTypeRaw = tmp[:2]
        QType = ""
        for x in QTypeRaw:
            QType += hex(x)[2:]
        QType = int(QType, 16)
        # QClass
        QClassRaw = tmp[2:]
        QClass = ""
        for x in QClassRaw:
            QClass += hex(x)[2:]
        QClass = int(QClass, 16)
        if debug:
            print("Domain Parts: " + str(domainParts))
```

```
            print("QType: " + str(QType))
            print("QClass: " + str(QClass))

        return Query(domainParts, QType, QClass)

    def toBytes(self):
        res = b""
        for part in self.nameParts:
            res += bytes([len(part)]) + bytes(part, "utf-8")
        res += b"\x00"
        res += self.type.to_bytes(2, "big")
        res += self.qclass.to_bytes(2, "big")
        return res

# Get type no from Query type name
def getTypeNoFromName(name):
    for i in range(len(queryTypesName)):
        if name == queryTypesName[i]:
            return i + 1
    return 0

# Get class no from Query class name
def getClassNoFromName(name):
    for i in range(len(queryClassesName)):
        if name == queryClassesName[i]:
            return i + 1
    return 0

# Result Record to bytes
def resultRecordToBytes(qTypeName, qClassName, val:str, ttl:int):
    print("Converting " + qTypeName + " " + qClassName + " " + val + " " + str(ttl))
    res = b"\xc0\x0c"
    res += getTypeNoFromName(qTypeName).to_bytes(2, "big")
    res += getClassNoFromName(qClassName).to_bytes(2, "big")
    res += ttl.to_bytes(4, "big")
    if qTypeName == "A":
        res += b"\x00\x04"
        res += bytes(map(int, val.split(".")))
    elif qTypeName == "AAAA":
        val = [int(x) for x in val.split(":")]
        res += len(val).to_bytes(2, "big")
        for x in val:
            res += x.to_bytes(2, "big")
    else:
        totalLen = 0
        data = b""
        for x in val.split("\n"):
            r = bytes(x, "utf-8")
            rLen = len(r)
            totalLen += rLen+1
            data += rLen.to_bytes(1, "big") + r

        res += totalLen.to_bytes(2, "big")
```

```
        res += data

    return res
```

Working Demo [via dig] :

```
tanmoy@tanmoy-laptop:~$ dig A tanmoy.codes @127.0.0.1
;; Warning: query response not set
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.16.1-Ubuntu <<>> A tanmoy.codes @127.0.0.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9037
;; flags: rd ad; QUERY: 1, ANSWER: 3, AUTHORITY: 2, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;tanmoy.codes.                  IN      A

;; ANSWER SECTION:
tanmoy.codes.           440     IN      A       76.76.21.20
tanmoy.codes.           460     IN      A       76.76.21.21
tanmoy.codes.           460     IN      A       76.76.21.22

;; Query time: 4 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Nov 06 12:31:53 IST 2022
;; MSG SIZE  rcvd: 78

tanmoy@tanmoy-laptop:~$ dig TXT tanmoy.codes @127.0.0.1
;; Warning: query response not set
;; Warning: Message parser reports malformed message packet.

; <<>> DiG 9.16.1-Ubuntu <<>> TXT tanmoy.codes @127.0.0.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39337
;; flags: rd ad; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;tanmoy.codes.                  IN      TXT

;; ANSWER SECTION:
tanmoy.codes.           450     IN      TXT     "v=spf1"
tanmoy.codes.           450     IN      TXT     "\"v=spf1 -al\""

;; Query time: 8 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sun Nov 06 12:32:53 IST 2022
;; MSG SIZE  rcvd: 74
```

Conclusion : The implementation of the DNS server is working fine with the DNS client. However it can only send results based on the zone records available to it, So we can extend that by giving support for querying the root server.

Telnet Protocol

Overview -Telnet is a network protocol used to virtually access a computer and to provide a two-way, collaborative and text-based communication channel between two machines. It follows a user command Transmission Control Protocol/Internet Protocol (TCP/IP) networking protocol for creating remote sessions

Code Implementation -

```
import socket
import subprocess
from threading import Thread
from time import sleep

class TelentServer:
    def __init__(self, host="127.0.0.1", port=5000):
        self.host = host
        self.port = port
        self.exited = False
        self.exitedListening = False
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind((self.host, self.port))
        self.sock.settimeout(2)
        self.sock.listen()
        self.threads_pool = []
        print("Server is listening on port", port)

    def startProcess(self):
        while True:
            try:
                conn, addr = self.sock.accept()
                conn.settimeout(2)
                print("Connected by", addr)
                t = Thread(target=self.handleClient, args=(conn, addr))
                t.start()
                self.threads_pool.append(t)
            except socket.timeout:
                if self.exited:
                    break
                else :
                    continue
            except:
                break
        self.exitedListening = True
```

```
    def handleClient(self, conn:socket.socket, addr):
        while True:
            try:
                data = conn.recv(1024)
                res = subprocess.run(data.decode().strip(), shell=True,
capture_output=True)
                if res.returncode == 0:
                    conn.sendall(res.stdout)
                else:
                    conn.sendall(res.stderr)
                if not data:
                    break
                if data.decode().strip() == "exit":
                    break
            except socket.timeout:
                if self.exited:
                    break
                continue
            except:
                break
        conn.close()

    def stopProcess(self):
        self.exited = True
        while not self.exitedListening:
            sleep(1)
        for t in self.threads_pool:
            try:
                t.join()
            except:
                pass
        self.sock.close()

if __name__ == "__main__":
    server = TelentServer()

    try:
        server.startProcess()
    except KeyboardInterrupt:
        server.exited = True
        server.stopProcess()
        print("Server is closed")
```

Working Demo [via Telnet Client] :

```
(base) tanmoy@tanmoy-laptop:~$ telnet 127.0.0.1 5000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
ls
telnet.py
pwd
/home/tanmoy/Desktop/Lab/Computer Network/Ass8/telnet
uname -r
5.15.0-52-generic
```

Conclusion : Telnet is a very important protocol to operate remote systems. We have built the Telnet Server on top of the TCP/IP stack and it's compatible with any telnet client

THANK YOU