Computer Networks

Lab Report

Submitted By,

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Year: 3rd
Semester: 1

Index:

Sl. No.	Assignment	Page No.	Submission Date
1	Assignment 1	2	14/8/22
2	Assignment 2	9	28/8/22
3	Assignment 3	34	12/9/22
4	Assignment 4	59	19/9/22
5	Assignment 5	64	31/10/22
6	Assignment 6	71	31/10/22
7	Assignment 7	87	7/11/22
8	Assignment 8	94	10/11/22

Computer Networks Lab

Assignment 1 Report

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

Design and implement an error detection module.

Design:

Solution Approach:

Two files, '*sender.txt*' and '*receiver.txt*' were used to put the input and output test cases separately. Another file, '*encode.py*' contained the encoding algorithms for all four schemes, Vertical Redundancy Check (VRC), Longitudinal Redundancy Check (LRC), Cyclic Redundancy Check (CRC) and Checksum.

For Vertical Redundancy Check and Cyclic Redundancy Check, we assume the entire file contents as 1 whole frame. For Longitudinal Redundancy Check and Checksum algorithms, we divide the file contents into frames of size four each.

For adding noise, we first choose how many bits get changed by noise by picking a random integer, and then change that many bits by choosing them at random using random.sample() method.

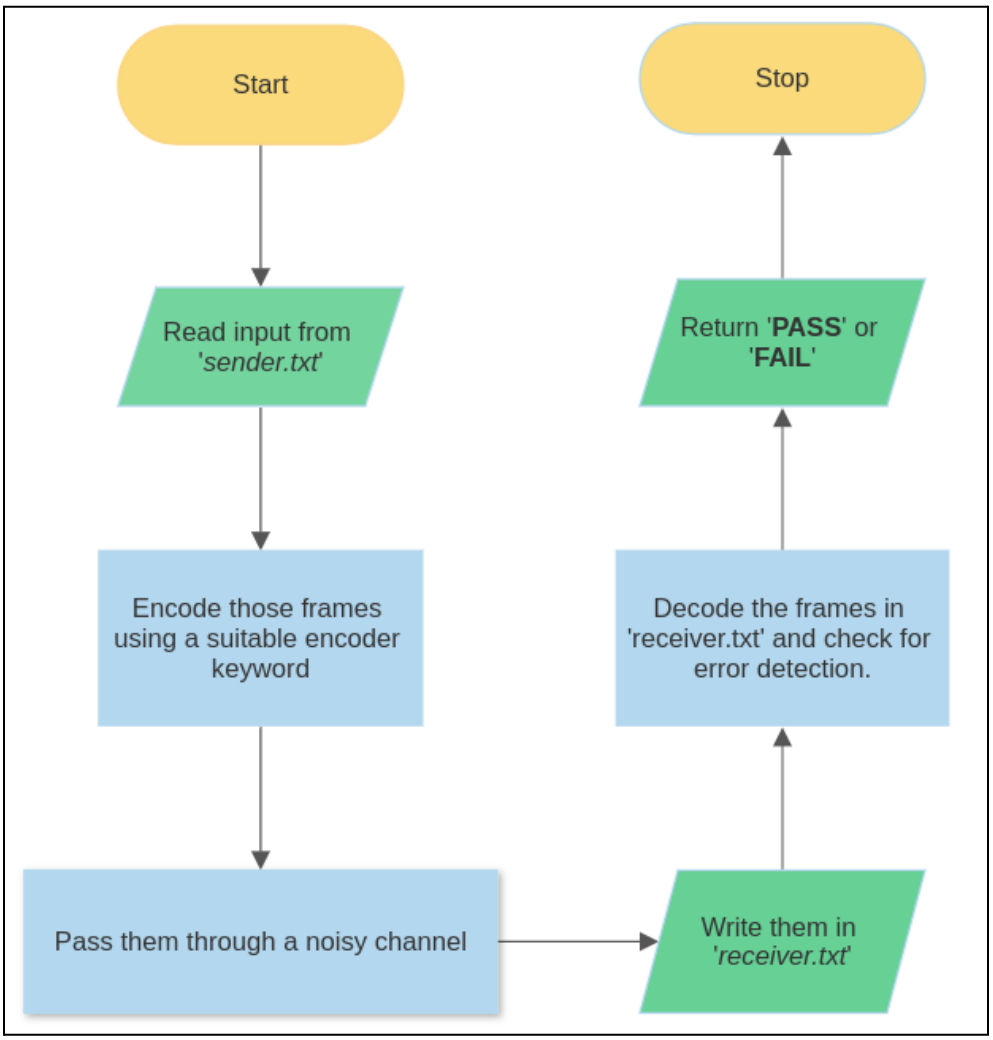
The file '*decoder.py*' decodes the messages from 'receiver.txt' and then uses the corresponding algorithms to detect any errors.

And finally the program is implemented in '*main.py*'

Methods for binary division and warped sum are also implemented.

CRC Polynomial used: CRC-10. That is, $x^{10} + x^9 + x^5 + x^4 + x + 1$ or 11000110011.

Diagram for how one error detection algorithm is implemented:



Input format:

A text file '*sender.txt*' containing binary bits.

Output format:

For each time the method *execute()* is called:

Testing case: <*case name*>

Result: <'*PASS*'/'*FAIL*'>

Example:

Testng case: "Error detected by all four schemes."
Result: PASS
Testing case: "Error detected by checksum but not by CRC."
Result: FAIL
Testing case: "Error detected by VRC but not by CRC."
Result: FAIL

Note: *Solely for evaluating performance over a large number of calls, we put execute() over a loop.*

Implementation:

Method description:

1. File: *encoder.py*
    a. **encode_vrc(frame):** Encodes a frame using the VRC algorithm.
    b. **encode_lrc(frame_list):** Encodes a list of frames using the LRC algorithm.

c. **encode_crc(frame, divisor):** Encodes a frame using the CRC algorithm using a suitable divisor.
d. **encode_checksum(frame_list):** Encodes a list of frames using the Checksum algorithm.

2. File: *decoder.py*
    a. **decode_vrc(frame):** Decodes a frame using the VRC algorithm.
    b. **decode_lrc(frame_list):** Decodes a list of frames using the LRC algorithm.
    c. **decode_crc(frame, divisor):** Decodes a frame using the CRC algorithm using a suitable divisor.
    d. **decode_checksum(frame_list):** Decodes a list of frames using the Checksum algorithm.

3. File: *operations.py*
    a. **noisy_channel(frame):** Emulates the passing of a frame through a noisy channel where error can occur randomly.
    b. **xor(a, b):** Finds out the XOR of two binary numbers a and b.
    c. **binary_division(dividend, divisor):** Implements binary division.
    d. **warped_sum(sum, frame_size):** Finds out the warped sum in checksum algorithm
    e. **readfile(filename, frame_size):** Extracts frames from a file.
    f. **writefile(filename, frame_list):** Writes a list of frames into a file.
    g. **execute():** Executes the program

4. File: *main.py*
    a. **vrc():** Implements VRC algorithm.
    b. **lrc():** Implements LRC algorithm.
    c. **crc():** Implements CRC algorithm.
    d. **checksum():** Implements checksum algorithm.

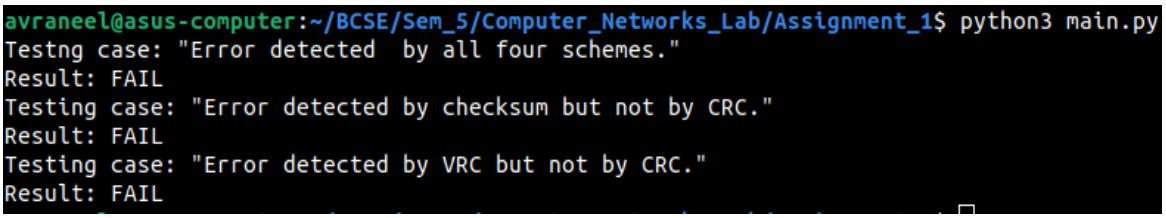
Test cases:

Contents of the sender file: 110101101011

1. *No. of bits changed = 0.*

    This means that the message will pass through a noiseless channel.

    Output:



    Thus, no error is detected.

2. *No. of bits changed = random*

    Error will occur at random positions.

    Output for one of these cases:

```
avraneel@asus-computer:~/BCSE/Sem_5/Computer_Networks_Lab/Assignment_1$ python3 main.py
Testng case: "Error detected  by all four schemes."
Result: FAIL
Testing case: "Error detected by checksum but not by CRC."
Result: PASS
Testing case: "Error detected by VRC but not by CRC."
Result: FAIL
```

Results:

For evaluating performance, we call the execute() function 1000 times at once and count how many times each case gives a ‘PASS’ result.

The sum of all the counts is less than 1000 because sometimes errors are not detected at all.

Sl. No.	Case 1 ‘PASS’ count	Case 2 ‘PASS’ count	Case 3 ‘PASS’ count
1	401	71	478
2	379	76	513
3	384	75	518
4	382	67	498

Analysis:

Possible Improvements:

- ➢ Socket programming could be used to further emulate the actual process of real-time communication.
- ➢ The error generating algorithm could be improved further.

➢ A better CRC divisor could be chosen.

Comments:

This assignment helped to learn how to implement various error detecting algorithms and also helped me in understanding how to implement various mathematical functions like binary division and warped sum.

Computer Networks Lab

Assignment 2 Report

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

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

Design:

Stats.py file will hold necessary socket information, and the frame format.

Frame Format:

The frame format will be of the form

[NN][LLL][DDDDDDDDDDDDDDD][CCCC]

Where,

N = packet number,
L = length of the data portion,
D = data portion, will only hold 0s and 1s

C = CRC portion

Assumptions Taken:

- Only error in the data portion can occur during transmission, no delay occurs.
- No error occurs for ACK and NAK.
- Resending will always send a correct frame.
- Data portion 11110000 is reserved as ACK.
- Data portion 00001111 is reserved as NAK.

Timeout Implementation:

- Timeout is implemented via exception handling.

Implementation:

stats.py:

```
import socket
import time
import random

# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)

# DISCONNECT MESSAGE
DISCONNECT = '1111'

# SIZE OF FRAME PARTS
N_SIZE = 2
LENGTH_SIZE = 3
DATA_SIZE = 11
CRC_SIZE = 4
```

```
ack_frame = "11110000"

# FRAME FORMAT = [NN][LLL][DDDDDDDDDDDDDDD][CCCCCC]
# where D --> data, L --> length, N --> pckt no., C --> CRC
```

operations.py:

```
import random
import stats as st
import time

# For sake of convinience we are only injecting error in the data part

def noisychannel(frame):

    frame_list = list(frame[-st.CRC_SIZE-st.DATA_SIZE:-st.CRC_SIZE])  #
Converting frame in string for to list
    no_of_bits_changed =  random.randint(0,len(frame_list)-1)
    enum = enumerate(frame_list)

    positions = random.sample(list(enum), no_of_bits_changed)

    for _, (index, _) in enumerate(positions):
        if frame_list[index] == '0':
            frame_list[index] = '1'
        elif frame_list[index] == '1':
            frame_list[index] = '0'

    new_frame = frame[0:st.N_SIZE] +
frame[st.N_SIZE:st.N_SIZE+st.LENGTH_SIZE] + ''.join(frame_list) +
frame[-st.CRC_SIZE:]
    return new_frame

def delay():
    time.sleep(random.randint()% 6)

def xor(a, b):
```

```
    val = ''
    for i in range(len(b)):
        if(a[i] == b[i]):
            val += '0'
        else:
            val += '1'

    return val

def binary_division(dividend, divisor):

    rem = dividend[0:len(divisor)]
    i = len(divisor)

    while(len(rem) == len(divisor)):

        if(rem[0] != '0'):
            rem = xor(rem, divisor)[1:]
        else:
            rem = rem[1:]

        if(i == len(dividend)):
            break

        rem += dividend[i]
        i += 1

    return rem


def crc4itu(frame):

    divisor = "10011"
    remainder = binary_division(frame + '0000', divisor)

    return str(remainder)


# Wraps the data into the specified frame format
```

```
def makeFrame(n, data):                                    # n = nth frame to
send
    l = str(len(data))
    rem = crc4itu(data)

    # PADDING
    msg_n = str(n).zfill(st.N_SIZE)
    msg_l = l.zfill(st.LENGTH_SIZE)
    msg_data = data.zfill(st.DATA_SIZE)
    msg_rem = rem.zfill(st.CRC_SIZE)

    frame = msg_n + msg_l + msg_data + msg_rem
    return frame

def receiveFrame(frame):
    crc = int(frame[-st.CRC_SIZE:])

    n = int(frame[:st.N_SIZE]) # Extracting N

    # Extracting length
    l = int(frame[st.N_SIZE:st.N_SIZE+st.LENGTH_SIZE])

    data = frame[-st.CRC_SIZE-l:-st.CRC_SIZE]
    if data == "q":
        crc = ""
    else:
        # Extracting CRC code
        crc = frame[-st.CRC_SIZE:]
    return n, l, data, crc
```

stop-and-wait:

sender.py:

```
# Assumptions:
# 1. Resending will resend without errors
# 2. Error only occurs in data part

import socket
```

```
import time
import random
import threading
import operations as op
import stats as st

sn = 0  # Sequence number

# To calculate timeout
TIMEOUT_LIMIT = 2
time1 = 0
time2 = 0

sender = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sender.bind(st.ADDR)

canSend = True

# Copy to store incase we need to resend
copy_sn = 0
copy_data = ""

def send(conn, data):

    global canSend, time1, sn, copy_sn, copy_data

    # Sending frames
    if canSend == True:

        # Making the frame
        frame = op.makeFrame(sn, data)
        print(f"[ENCODING] Encoded frame: {frame}")
        #time.sleep(random.randint()% 6)
        frame = op.noisychannel(frame)
        print(f"[NOISY] Frame after noise: {frame}")

        # Sending frame
        conn.send(frame.encode())

        # Storing Frame
```

```
            copy_data = data
            copy_sn = sn

            # Starting timer
            time1 = time.time()

            sn += 1

            canSend = False

        if data == "q":
            return

        # Receing ACK
        recv_Ack(conn)

        # Resending Frame
        while canSend == False:
            frame = op.makeFrame(copy_sn, copy_data)
            frame = op.noisychannel(frame)
            print(f"[RESENDING] Resending frame: {frame}")
            time1 = time.time();
            conn.send(frame.encode())

            # Receving Ack for the resent frame
            recv_Ack(conn)

def recv_Ack(conn):

    global sn, canSend, copy_data, copy_sn, time2
    print("[Reciving ACK]......")

    # Receiving ACK Frame
    try:
        conn.settimeout(0.5)    # Setting timeout time
        ack_frame = conn.recv(20).decode()
    except:
        # Timeout has occured, so we should resend the frame
        print("---[TIMEOUT OCCURED]----")
        return
```

```
    ackNo, _, data,_ = op.receiveFrame(ack_frame)

    # Checking if ACK Valid
    if ackNo == copy_sn and data == '11110000':

        # Stopping timer
        time2 = time.time()

        print(f"[ACK RECV] ACK {ackNo} successfully received.")

        # Purging data
        copy_sn = 0
        copy_data = ""

        canSend = True
        print("[TRANSACTION COMPLETED]")
        return


def start():

    # Listening
    sender.listen()
    print(f"[LISTENING] Server is listening on {st.HOST_IP}")

    # Accepting receiver
    conn, addr = sender.accept()
    print(f"[CONNECTED] Connected to Process Id: {addr}")

    while True:
        data = input('[INPUT] Enter data to send: ')

        send(conn, data)

        if data == 'q':
            print("[CLOSING] Closing the sender....")
            conn.close()
            break
```

```
        print("-------------------------------------------------")


start()
```

receiver.py:

```
import socket
import time
import operations as op
import stats as st

rn = 0 # Sequence number

receiver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
receiver.connect(st.ADDR)
# receiver.settimeout(3)

def isCorrupted(data, crc):
    if op.crc4itu(data) != crc:
        return True
    else:
        return False

def Recv():
    global rn

    while True:

        print("[LISTENING] Receiver is listening....")
        frame = receiver.recv(20).decode()
        recv_n, _, data, crc = op.receiveFrame(frame)
        print(f"[RECV] Received message: {data}")

        # Checking if disconnect statement is there
        if data == 'q':
            print("[CLOSING] Closing the receiver....")
            receiver.close()
```

```
            return

        if recv_n == rn:

                if isCorrupted(data, crc) == False: # No crc error
                    print(f"[CRC SUCCESS] {op.crc4itu(data)} and {crc}")
                    send_Ack()
                    rn += 1
                    print(f"[ACK SENT] Sent ACK")
                    print("[TRANSACTION COMPLETED]")

                else:
                    print(f"[CRC FAILURE] {op.crc4itu(data)} and {crc}")
                    #receiver.settimeout(3)
        else:
            print("Wrong receiveer")
        print("-------------------------------------------------")

def send_Ack():
    ack_frame = "11110000"
    ack_frame = op.makeFrame(rn, ack_frame)
    receiver.send(ack_frame.encode())

Recv()
```

go-back-n:

sender.py:

```
import socket, threading, time, operations as op, stats as st

sf = 0
sn = 0
sw = 4

text = []
length = 0
flag2 = False   # used to terminate the recv_Ack thread
```

```
sender = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sender.bind(st.ADDR)

data_queue = [None]*sw

flag = False

def send(conn, f):

    global sn, sf, sw, data_queue, flag, text

    while True:
        if (sn - sf) < sw:

            try:
                data = text.pop(0)
            except:
                print("[FINISHED] All output read.")
                flag = True
                return

            frame = op.noisychannel(op.makeFrame(sn, data))
            print(f"[SENDING] Sending frame: {frame}")
            conn.send(frame.encode())

            data_queue[sn % sw] = data

            sn += 1

def recv_Ack(conn):

    global sn, sf, sw, data_queue, text, length

    # Receiving ACK Frame
    while True:
        global sf, sn, sw, data_queue, flag2
        try:
            conn.settimeout(0.5)
            recv_frame = conn.recv(20).decode()
```

```
            # Extracting frame details
            ackNo, _, data,_ = op.receiveFrame(recv_frame)

        except:
            resend_thread = threading.Thread(target=timeout_steps,
args=(conn,))
            resend_thread.start()
            resend_thread.join()
            if flag2 == True:
                print("[FINISHED] All ACK received.")
                return
            continue

        if ackNo >= sf and ackNo <= sn and data == '11110000': # if valid
ACK
            while(sf <= ackNo):
                print(f"[ACK RECV] ACK {ackNo} successfully received.")
                data_queue[sf % sw] = ""
                sf += 1

        if ackNo == (length - 1):  # All ACK received
            flag2 = True

def timeout_steps(conn):
    global sf, sn, sw, data_queue
    temp = sf
    while(temp < sn):
        data = data_queue[temp % sw]
        frame = op.makeFrame(temp, data)
        print(f"[RE SENDING] Resending frame: {frame}")
        conn.send(frame.encode())
        temp += 1


def start():
    # Listening
    global text, length

    sender.listen()
    print(f"[LISTENING] Server is listening on {st.HOST_IP}")
```

```
    # Accepting receiver
    conn, addr = sender.accept()
    print(f"[CONNECTED] Connected to Process Id: {addr}")

    f = open("data.txt", "r")

    text = f.readlines()
    for i in range(len(text)):
        text[i] = text[i].strip()

    length = len(text)

    sender_thread = threading.Thread(target=send, args=(conn, f))
    receiver_thread = threading.Thread(target=recv_Ack, args=(conn,))

    sender_thread.start()
    receiver_thread.start()
    sender_thread.join()
    receiver_thread.join()

    print("[CLOSING] Closing sender...")
    conn.close()

    f.close()


start()
```

receiver.py:

```
from ast import arg
import socket, time, threading, operations as op, stats as st

rn = 0

receiver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
receiver.connect(st.ADDR)
# receiver.settimeout(3)
```

```
def isCorrupted(data, crc):
    if op.crc4itu(data) != crc:
        return True
    else:
        return False

def Recv():
    global rn

    while True:

        # print("[LISTENING] Receiver is listening....")
        try:
            frame = receiver.recv(20).decode()
            recv_n, _, data, crc = op.receiveFrame(frame)
        except:
            # When everything is done, ValueError will be raised due to
receiver receiving empty string
            # so we use it to terminate this thread
            return
        print(f"[RECV] Received message: {recv_n}, {data}")

        # Checking if disconnect statement is there

        if isCorrupted(data, crc) == False and recv_n == rn : # No crc
error
            print(f"[CRC SUCCESS] {op.crc4itu(data)} and {crc}")
            send_Ack()
            rn += 1
            print(f"[ACK SENT] Sent ACK")
        else:
            print(f"[CRC FAILURE] {op.crc4itu(data)} and {crc}")

        #print("-----------------------------------------------")

def send_Ack():
    ack_frame = "11110000"
    ack_frame = op.makeFrame(rn, ack_frame)
    receiver.send(ack_frame.encode())
```

```
receiver_thread = threading.Thread(target=Recv)

receiver_thread.start()
receiver_thread.join()

print("[CLOSING] Closing receiver....")
receiver.close()
```

selective-Repeat:

sender.py:

```
# Assumptions:
# 1. Resending will resend without errors
# 2. Error only occurs in data part

import socket
import time
import random
import threading
import operations as op
import stats as st

sn = 0  # Sequence number

# To calculate timeout
TIMEOUT_LIMIT = 2
time1 = 0
time2 = 0

sender = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sender.bind(st.ADDR)

canSend = True

# Copy to store incase we need to resend
copy_sn = 0
copy_data = ""
```

```
def send(conn, data):

    global canSend, time1, sn, copy_sn, copy_data

    # Sending frames
    if canSend == True:

        # Making the frame
        frame = op.makeFrame(sn, data)
        print(f"[ENCODING] Encoded frame: {frame}")
        #time.sleep(random.randint()% 6)
        frame = op.noisychannel(frame)
        print(f"[NOISY] Frame after noise: {frame}")

        # Sending frame
        conn.send(frame.encode())

        # Storing Frame
        copy_data = data
        copy_sn = sn

        # Starting timer
        time1 = time.time()

        sn += 1

        canSend = False

    if data == "q":
        return

    # Receing ACK
    recv_Ack(conn)

    # Resending Frame
    while canSend == False:
        frame = op.makeFrame(copy_sn, copy_data)
        frame = op.noisychannel(frame)
        print(f"[RESENDING] Resending frame: {frame}")
```

```
        time1 = time.time();
        conn.send(frame.encode())

        # Receving Ack for the resent frame
        recv_Ack(conn)

def recv_Ack(conn):

    global sn, canSend, copy_data, copy_sn, time2
    print("[Reciving ACK]......")

    # Receiving ACK Frame
    try:
        conn.settimeout(0.5)    # Setting timeout time
        ack_frame = conn.recv(20).decode()
    except:
        # Timeout has occured, so we should resend the frame
        print("---[TIMEOUT OCCURED]----")
        return

    ackNo, _, data,_ = op.receiveFrame(ack_frame)

    # Checking if ACK Valid
    if ackNo == copy_sn and data == '11110000':

        # Stopping timer
        time2 = time.time()

        print(f"[ACK RECV] ACK {ackNo} successfully received.")

        # Purging data
        copy_sn = 0
        copy_data = ""

        canSend = True
        print("[TRANSACTION COMPLETED]")
        return


def start():
```

```
    # Listening
    sender.listen()
    print(f"[LISTENING] Server is listening on {st.HOST_IP}")

    # Accepting receiver
    conn, addr = sender.accept()
    print(f"[CONNECTED] Connected to Process Id: {addr}")

    while True:
        data = input('[INPUT] Enter data to send: ')

        send(conn, data)

        if data == 'q':
            print("[CLOSING] Closing the sender....")
            conn.close()
            break

        print("-----------------------------------------------------")


start()
```

receiver.py:

```
import socket
import time
import operations as op
import stats as st

rn = 0 # Sequence number

receiver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
receiver.connect(st.ADDR)
# receiver.settimeout(3)

def isCorrupted(data, crc):
    if op.crc4itu(data) != crc:
```

```
        return True
    else:
        return False

def Recv():
    global rn

    while True:

        print("[LISTENING] Receiver is listening....")
        frame = receiver.recv(20).decode()
        recv_n, _, data, crc = op.receiveFrame(frame)
        print(f"[RECV] Received message: {data}")

        # Checking if disconnect statement is there
        if data == 'q':
            print("[CLOSING] Closing the receiver....")
            receiver.close()
            return

        if recv_n == rn:

                if isCorrupted(data, crc) == False: # No crc error
                    print(f"[CRC SUCCESS] {op.crc4itu(data)} and {crc}")
                    send_Ack()
                    rn += 1
                    print(f"[ACK SENT] Sent ACK")
                    print("[TRANSACTION COMPLETED]")

                else:
                    print(f"[CRC FAILURE] {op.crc4itu(data)} and {crc}")
                    #receiver.settimeout(3)
        else:
            print("Wrong receiveer")
        print("-----------------------------------------------")

def send_Ack():
    ack_frame = "11110000"
    ack_frame = op.makeFrame(rn, ack_frame)
    receiver.send(ack_frame.encode())
```

`Recv()`

Test cases:

Stop-and-wait:

Sender side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/stopandwait
$ python3 sender.py
[LISTENING] Server is listening on 127.0.1.1
[CONNECTED] Connected to Process Id: ('127.0.0.1', 41362)
[INPUT] Enter data to send: 1101
[ENCODING] Encoded frame: 00004000000011010100
[NOISY] Frame after noise: 00004000000111000100
[Reciving ACK]......
---[TIMEOUT OCCURED]----
[RESENDING] Resending frame: 00004000100111100100
[Reciving ACK]......
---[TIMEOUT OCCURED]----
[RESENDING] Resending frame: 00004000010110000100
[Reciving ACK]......
---[TIMEOUT OCCURED]----
[RESENDING] Resending frame: 00004000000011010100
[Reciving ACK]......
[ACK RECV] ACK 0 successfully received.
[TRANSACTION COMPLETED]
-----------------------------------------------------
[INPUT] Enter data to send: 111010
[ENCODING] Encoded frame: 01006000001110100010
[NOISY] Frame after noise: 01006100101110000010
[Reciving ACK]......
---[TIMEOUT OCCURED]----
[RESENDING] Resending frame: 01006111000001010010
[Reciving ACK]......
---[TIMEOUT OCCURED]----
[RESENDING] Resending frame: 01006000001110100010
[Reciving ACK]......
[ACK RECV] ACK 1 successfully received.
[TRANSACTION COMPLETED]
-----------------------------------------------------
[INPUT] Enter data to send: q
[ENCODING] Encoded frame: 020010000000000q0011
[NOISY] Frame after noise: 020011100000001q0011
[CLOSING] Closing the sender....
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/stopandwait
$
```

Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/stopandwa
stopandwait$ python3 receiver.py
[LISTENING] Receiver is listening....
[RECV] Received message: 1100
[CRC FAILURE] 0111 and 0100
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 1110
[CRC FAILURE] 0001 and 0100
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 1000
[CRC FAILURE] 1011 and 0100
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 1101
[CRC SUCCESS] 0100 and 0100
[ACK SENT] Sent ACK
[TRANSACTION CO PLETED]
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 111000
[CRC FAILURE] 0100 and 0010
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 000101
[CRC FAILURE] 1111 and 0010
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message: 111010
[CRC SUCCESS] 0010 and 0010
[ACK SENT] Sent ACK
[TRANSACTION CO PLETED]
-----------------------------------------------
[LISTENING] Receiver is listening....
[RECV] Received message:
[CLOSING] Closing the receiver....
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/stopandwa
it$
```

Go-back-n:

Sender side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/gobackn$ py
thon3 sender.py
[LISTENING] Server is listening on 127.0.1.1
[CONNECTED] Connected to Process Id: ('127.0.0.1', 59436)
[SENDING] Sending frame: 00006000001001100101
[SENDING] Sending frame: 01006000000101100110
[SENDING] Sending frame: 02006100101001110101
[SENDING] Sending frame: 03006100001001010101
[RE SENDING] Resending frame: 00006000001101100101
[RE SENDING] Resending frame: 01006000001101110110
[RE SENDING] Resending frame: 02006000001101100101
[RE SENDING] Resending frame: 03006000001001010101
[ACK RECV] ACK 0 successfully received.
[ACK RECV] ACK 1 successfully received.
[SENDING] Sending frame: 04006101111000001010
[ACK RECV] ACK 2 successfully received.
[SENDING] Sending frame: 05006111001010000000
[SENDING] Sending frame: 06006100001110010111
[ACK RECV] ACK 3 successfully received.
[SENDING] Sending frame: 07006010101010100000
[ACK RECV] ACK 4 successfully received.
[FINISHED] All output read.
[RE SENDING] Resending frame: 05006000001001100000
[RE SENDING] Resending frame: 06006000001110010111
[RE SENDING] Resending frame: 07006000001001100000
[ACK RECV] ACK 5 successfully received.
[ACK RECV] ACK 6 successfully received.
[ACK RECV] ACK 7 successfully received.
[FINISHED] All ACK received.
[CLOSING] Closing sender...
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/gobackn$
```

Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/gobackn$ p
ython3 receiver.py
[RECV] Received message: 0, 100110
[CRC FAILURE] 0000 and 0101
[RECV] Received message: 1, 010110
[CRC FAILURE] 1111 and 0110
[RECV] Received message: 2, 100111
[CRC FAILURE] 0011 and 0101
[RECV] Received message: 3, 100101
[CRC FAILURE] 0101 and 0101
[RECV] Received message: 0, 110110
[CRC SUCCESS] 0101 and 0101
[ACK SENT] Sent ACK
[RECV] Received message: 1, 110111
[CRC SUCCESS] 0110 and 0110
[ACK SENT] Sent ACK
[RECV] Received message: 2, 110110
[CRC SUCCESS] 0101 and 0101
[ACK SENT] Sent ACK
[RECV] Received message: 3, 100101
[CRC SUCCESS] 0101 and 0101
[ACK SENT] Sent ACK
[RECV] Received message: 4, 100000
[CRC SUCCESS] 1010 and 1010
[ACK SENT] Sent ACK
[RECV] Received message: 5, 101000
[CRC FAILURE] 0001 and 0000
[RECV] Received message: 6, 111001
[CRC FAILURE] 0111 and 0111
[RECV] Received message: 7, 101010
[CRC FAILURE] 0111 and 0000
[RECV] Received message: 5, 100110
[CRC SUCCESS] 0000 and 0000
[ACK SENT] Sent ACK
[RECV] Received message: 6, 111001
[CRC SUCCESS] 0111 and 0111
[ACK SENT] Sent ACK
[RECV] Received message: 7, 100110
[CRC SUCCESS] 0000 and 0000
[ACK SENT] Sent ACK
[CLOSING] Closing receiver....
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/gobackn$
```

Selective-repeat:

Sender side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/selectivere
peat$ python3 sender.py
[LISTENING] Server is listening on 127.0.1.1
[CONNECTED] Connected to Process Id: ('127.0.0.1', 46632)
[SENDING] Sending frame: no. 0, 00006000000101100101
[SENDING] Sending frame: no. 1, 01006011111001000110
[NAK RECV] NAK 0 successfully received.
[SENDING] Sending frame: no. 2, 02006000001101100101
[RE SENDING] Resending frame: 0, 00006000001101100101
[SENDING] Sending frame: no. 3, 03006111100111100101
[ACK RECV] ACK 1 successfully received.
[ACK RECV] ACK 2 successfully received.
[NAK RECV] NAK 3 successfully received.
[SENDING] Sending frame: no. 4, 04006001101101111010
[RE SENDING] Resending frame: 3, 03006000001001010101
[SENDING] Sending frame: no. 5, 05006000001001100000
[SENDING] Sending frame: no. 6, 06006011111101110111
[NAK RECV] NAK 4 successfully received.
[RE SENDING] Resending frame: 4, 04006000001100111010
[ACK RECV] ACK 3 successfully received.
[ACK RECV] ACK 5 successfully received.
[NAK RECV] NAK 6 successfully received.
[SENDING] Sending frame: no. 7, 07006111110100010000
[RE SENDING] Resending frame: 6, 06006000001110010111
[FINISHED] All output read.
[NAK RECV] NAK 7 successfully received.
[RE SENDING] Resending frame: 7, 07006000001001100000
[ACK RECV] ACK 6 successfully received.
[ACK RECV] ACK 7 successfully received.
[CLOSING] Closing sender...
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/selectivere
peat$
```

Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 2/selectiver
epeat$ python3 receiver.py
[RECV] Received message: 0, 010110
[CRC FAILURE] 1111 and 0101
[NAK SENT] Sent NAK 0
-----------------------------------------------
[RECV] Received message: 1, 100100
[CRC SUCESS] 0110 and 0110
[ACK SENT] Sent ACK 1
-----------------------------------------------
[RECV] Received message: 2, 110110
[NAK SENT] Sent NAK 3
-----------------------------------------------
[RECV] Received message: 4, 110111
[CRC FAILURE] 0110 and 1010
[NAK SENT] Sent NAK 4
-----------------------------------------------
[RECV] Received message: 3, 100101
[CRC SUCESS] 0101 and 0101
[ACK SENT] Sent ACK 3
-----------------------------------------------
[RECV] Received message: 5, 100110
[CRC SUCESS] 0000 and 0000
[ACK SENT] Sent ACK 5
-----------------------------------------------
[RECV] Received message: 6, 110111
[CRC FAILURE] 0110 and 0111
[NAK SENT] Sent NAK 6
-----------------------------------------------
[RECV] Received message: 4, 110011
[CRC SUCESS] 1010 and 1010
[ACK SENT] Sent ACK 4
-----------------------------------------------
[RECV] Received message: 7, 010001
[CRC FAILURE] 0110 and 0000
[NAK SENT] Sent NAK 7
-----------------------------------------------
[RECV] Received message: 6, 111001
[CRC SUCESS] 0111 and 0111
[ACK SENT] Sent ACK 6
-----------------------------------------------
[RECV] Received message: 7, 100110
[CRC SUCESS] 0000 and 0000
[ACK SENT] Sent ACK 7
-----------------------------------------------
[CLOSING] Closing receiver....
```

Comments:

This assignment helped me to learn how to implement various flow control protocols using socket programming.

Computer Networks Lab

Assignment 3 Report

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

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

Plot the comparison graphs for throughput and forwarding delay by varying p. State your observations on the impact of performance of different CSMA techniques.

Design:

Each channel-frame connection will have 2 threads. A sending thread for the main data communication, and a sensing thread to sense whether the channel is busy or not.

In order to check whether the channel is busy or not, we reserve a packet with the data 11110000 as idle and another packet with the data 00001111 as busy.

Implementation:

There will be multiple stations conneted to only 1 channel. The purpose of stats.py is to hold socket variables and busy/idle signal data.

stats.py:

```
import socket
import time
import random

# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)

# DISCONNECT MESSAGE
DISCONNECT = '1111'

# SIZE OF FRAME PARTS
N_SIZE = 2
LENGTH_SIZE = 3
DATA_SIZE = 11
CRC_SIZE = 4

idle_signal = "11110000"
busy_signal = "00001111"
```

1-persistent:

Channel.py:

```
import socket, threading, time, stats as st

ThreadCount = 0
isBusy = False
```

```
channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    channel.bind(st.ADDR)
except socket.error as e:
    print(str(e))

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

def send(conn):
    global isBusy
    conn.send("[CONNECTED] Server is working.".encode())
    senders = 0
    while True:
        if isBusy == False:
            data = conn.recv(2).decode()
            senders += 1
            isBusy = True
            print(f"[RECEIVED] Received message = {data}")
            isBusy = False

    conn.close()

def signal(conn):
    global isBusy
    while True:
        if isBusy == True:
            conn.send(st.busy_signal.encode())
        else:
            conn.send(st.idle_signal.encode())

while True:
    conn, address = channel.accept()
       print('[CONNECTED] Connected to: ' + address[0] + ':' +
str(address[1]))
    client_thread = threading.Thread(target=send, args=(conn, ))
    sensing_thread = threading.Thread(target=signal, args=(conn, ))
```

```
    ThreadCount += 1
    client_thread.start()
    sensing_thread.start()
    print('[ACTIVE COUNT] Thread Number: ' + str(ThreadCount))
```

station.py:

```
import socket, threading, time, stats as st

station = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

canSend = False

print("[WAITING] Waiting for connection response")

try:
    station.connect(st.ADDR)
except socket.error as e:
    print(str(e))

res = station.recv(30).decode()
print(res)

fileno = input("Enter fileno to start: ")

f = open("data"+fileno+".txt", "r")
list_of_frames = f.readlines()

def send():
    global list_of_frames
    i = 0
    while True and i < 10:
        if canSend:
            frame = list_of_frames[i].strip()
            station.send(frame.encode())
            print(f"[SENDING] Sent message: {frame}")
            time.sleep(1)
            i += 1
        else:
```

```
            continue

def sense():
    global canSend
    flag1 = False
    flag2 = False
    while True:
        signal = station.recv(8).decode()
        if signal == st.busy_signal and flag1 == False:
            print("[BUSY] Sensing channel to be busy")
            canSend = False
            print("Retrying...")
            flag1 = True
            flag2 = False
            continue
        elif signal == st.idle_signal and flag2 == False:
            print("[IDLE] Sensing channel to be idle")
            canSend = True
            flag2 = True
            flag1 = False

sense_thread = threading.Thread(target=sense)
send_thread = threading.Thread(target=send)

sense_thread.start()
send_thread.start()
```

Non-persistent:

channel.py:

```
import socket, threading, time, stats as st

ThreadCount = 0
isBusy = False

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
```

```
    channel.bind(st.ADDR)
except socket.error as e:
    print(str(e))

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

def send(conn):
    global isBusy
    conn.send("[CONNECTED] Server is working.".encode())
    senders = 0
    while True:
        if isBusy == False:
            data = conn.recv(2).decode()
            senders += 1
            isBusy = True
            print(f"[RECEIVED] Received message = {data}")
            isBusy = False

    conn.close()

def signal(conn):
    global isBusy
    while True:
        if isBusy == True:
            conn.send(st.busy_signal.encode())
        else:
            conn.send(st.idle_signal.encode())

while True:
    conn, address = channel.accept()
         print('[CONNECTED]  Connected  to:  '  +  address[0]  +  ':'  +
str(address[1]))
    client_thread = threading.Thread(target=send, args=(conn, ))
    sensing_thread = threading.Thread(target=signal, args=(conn, ))
    ThreadCount += 1
    client_thread.start()
    sensing_thread.start()
    print('[ACTIVE COUNT] Thread Number: ' + str(ThreadCount))
```

station.py:

```
import socket, threading, time, os, random, stats as st

station = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

canSend = False

random.seed(os.getpid())

print("[WAITING] Waiting for connection response")

try:
    station.connect(st.ADDR)
except socket.error as e:
    print(str(e))

res = station.recv(30).decode()
print(res)

fileno = input("Enter fileno to start: ")

f = open("data"+fileno+".txt", "r")
list_of_frames = f.readlines()

def send():
    global list_of_frames, canSend
    i = 0
    while True and i < 10:
        if canSend:
            frame = list_of_frames[i].strip()
            station.send(frame.encode())
            print(f"[SENDING] Sent message: {frame}")
            time.sleep(1)
            i += 1

def sense():
    global canSend
    flag1 = False
    flag2 = False
```

```
    while True:
        signal = station.recv(8).decode()
        if signal == st.busy_signal and flag1 == False:
            print("[BUSY] Sensing channel to be busy")
            canSend = False
            flag1 = True
            t = random.randint(1,8)
            print(f"Waiting {t} s......")
            time.sleep(t)
            flag2 = False
        elif signal == st.idle_signal and flag2 == False:
            print("[IDLE] Sensing channel to be idle")
            canSend = True
            flag2 = True
            flag1 = False

sense_thread = threading.Thread(target=sense)
send_thread = threading.Thread(target=send)

sense_thread.start()
send_thread.start()
```

p-persistent:

Channel.py:

```
import socket, threading, time, stats as st

ThreadCount = 0
isBusy = False

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    channel.bind(st.ADDR)
except socket.error as e:
    print(str(e))

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
```

```
channel.listen(5)

def send(conn):
    global isBusy
    conn.send("[CONNECTED] Server is working.".encode())
    senders = 0
    while True:
        if isBusy == False:
            data = conn.recv(2).decode()
            senders += 1
            isBusy = True
            print(f"[RECEIVED] Received message = {data}")
            isBusy = False

    conn.close()

def signal(conn):
    global isBusy
    while True:
        if isBusy == True:
            conn.send(st.busy_signal.encode())
        else:
            conn.send(st.idle_signal.encode())

while True:
    conn, address = channel.accept()
         print('[CONNECTED]  Connected  to:  '  +  address[0]  +  ':'  +
str(address[1]))
    client_thread = threading.Thread(target=send, args=(conn, ))
    sensing_thread = threading.Thread(target=signal, args=(conn, ))
    ThreadCount += 1
    client_thread.start()
    sensing_thread.start()
    print('[ACTIVE COUNT] Thread Number: ' + str(ThreadCount))
```

station.py:

In our test cases, we took only 2 stations, so p = ½ = 0.5

```
import socket, threading, time, os, random, stats as st

station = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

canSend = False

p = 0.5
time_slot = 3

random.seed(os.getpid())

print("[WAITING] Waiting for connection response")

try:
    station.connect(st.ADDR)
except socket.error as e:
    print(str(e))

res = station.recv(30).decode()
print(res)

fileno = input("Enter fileno to start: ")

f = open("data"+fileno+".txt", "r")
list_of_frames = f.readlines()

def send():
    global list_of_frames
    i = 0
    while True and i < 10:
        if canSend:
            frame = list_of_frames[i].strip()
            station.send(frame.encode())
            print(f"[SENDING] Sent message: {frame}")
            time.sleep(1)
            i += 1
        else:
            continue

def sense():
```

```
    global canSend
    flag1 = False
    flag2 = False
    while True:

        signal = station.recv(8).decode()

        if signal == st.busy_signal and flag1 == False:
            print("[BUSY] Sensing channel to be busy")
            canSend = False
            flag1 = True
            time.sleep(time_slot)
            flag2 = False

        elif signal == st.idle_signal and flag2 == False:
            print("[IDLE] Sensing channel to be idle")

            probability = random.randint(1,10)/10
            print(f"Probability is = {probability}")

            if(probability <= p):
                print("[ALLOWED] Transmission allowed")
                canSend = True
                flag2 = True
                flag1 = False

            elif probability > p:
                print(f"[WAIT] Waiting for {time_slot}")
                canSend = False
                flag1 = True
                time.sleep(time_slot)
                flag2 = False

sense_thread = threading.Thread(target=sense)
send_thread = threading.Thread(target=send)

sense_thread.start()
send_thread.start()
```

Test cases:

1-persistent:

Channel output:

```
mester/Lab - Computer Networks/Assignment 3/code/1
-persistent$ python3 channel.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:33270
[ACTIVE COUNT] Thread Number: 1
[CONNECTED] Connected to: 127.0.0.1:33272
[ACTIVE COUNT] Thread Number: 2
[CONNECTED] Connected to: 127.0.0.1:33274
[ACTIVE COUNT] Thread Number: 3
[RECEIVED] Received message = A0
[RECEIVED] Received message = B0
[RECEIVED] Received message = A1
[RECEIVED] Received message = C0
[RECEIVED] Received message = B1
[RECEIVED] Received message = A2
[RECEIVED] Received message = C1
[RECEIVED] Received message = B2
[RECEIVED] Received message = A3
[RECEIVED] Received message = C2
[RECEIVED] Received message = B3
[RECEIVED] Received message = A4
[RECEIVED] Received message = C3
[RECEIVED] Received message = B4
[RECEIVED] Received message = A5
[RECEIVED] Received message = C4
[RECEIVED] Received message = B5
[RECEIVED] Received message = A6
[RECEIVED] Received message = C5
[RECEIVED] Received message = B6
[RECEIVED] Received message = A7
[RECEIVED] Received message = C6
[RECEIVED] Received message = B7
[RECEIVED] Received message = A8
[RECEIVED] Received message = C7
[RECEIVED] Received message = B8
[RECEIVED] Received message = A9
[RECEIVED] Received message = C8
[RECEIVED] Received message = B9
[RECEIVED] Received message = C9
```

Station 1:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Se
mester/Lab - Computer Networks/Assignment 3/code/1
-persistent$ python3 station.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 1
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A0
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A1
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A2
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A3
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A4
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A5
[BUSY] Sensing channel to be busy
Retrying...
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A6
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
```

```
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A7
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A8
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A9
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
```

Station 2:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Se
mester/Lab - Computer Networks/Assignment 3/code/1
-persistent$ python3 station.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 2
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B0
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B1
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B2
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B3
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B4
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B5
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B6
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
```

```
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B7
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B8
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B9
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
```

Station 3:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st S
emester/Lab - Computer Networks/Assignment 3/code
/1-persistent$ python3 station.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 3
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C0
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C1
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C2
[SENDING] Sent message: C3
[BUSY] Sensing channel to be busy
[SENDING] Sent message: C4
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C5
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C6
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C7
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C8
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
```

```
[SENDING] Sent message: C8
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
[SENDING] Sent message: C9
[BUSY] Sensing channel to be busy
Retrying...
[IDLE] Sensing channel to be idle
```

Non-persistent:

Channel Output:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab -
 Computer Networks/Assignment 3/code/non-persistent$ python3 c
hannel.py
4[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:56662
[ACTIVE COUNT] Thread Number: 1
[CONNECTED] Connected to: 127.0.0.1:56664
[ACTIVE COUNT] Thread Number: 2
[RECEIVED] Received message = A0
[RECEIVED] Received message = A1
[RECEIVED] Received message = B0
[RECEIVED] Received message = B1
[RECEIVED] Received message = B2
[RECEIVED] Received message = B3
[RECEIVED] Received message = B4
[RECEIVED] Received message = A2
[RECEIVED] Received message = A3
[RECEIVED] Received message = A4
[RECEIVED] Received message = A5
[RECEIVED] Received message = B5
[RECEIVED] Received message = B6
[RECEIVED] Received message = B7
[RECEIVED] Received message = B8
[RECEIVED] Received message = B9
[RECEIVED] Received message = A6
[RECEIVED] Received message = A7
[RECEIVED] Received message = A8
[RECEIVED] Received message = A9
```

Station 1:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab -
 Computer Networks/Assignment 3/code/non-persistent$ python3 s
tation.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 1
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A0
[SENDING] Sent message: A1
[BUSY] Sensing channel to be busy
Waiting 7 s......
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A2
[SENDING] Sent message: A3
[SENDING] Sent message: A4
[SENDING] Sent message: A5
[BUSY] Sensing channel to be busy
Waiting 6 s......
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A6
[SENDING] Sent message: A7
[BUSY] Sensing channel to be busy
Waiting 7 s......
[IDLE] Sensing channel to be idle
[SENDING] Sent message: A8
[SENDING] Sent message: A9
[BUSY] Sensing channel to be busy
Waiting 5 s......
[IDLE] Sensing channel to be idle
```

Station 2:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab
- Computer Networks/Assignment 3/code/non-persistent$ python3
 station.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 2
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B0
[SENDING] Sent message: B1
[SENDING] Sent message: B2
[BUSY] Sensing channel to be busy
Waiting 3 s......
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B3
[SENDING] Sent message: B4
[BUSY] Sensing channel to be busy
Waiting 6 s......
[IDLE] Sensing channel to be idle
[SENDING] Sent message: B5
[SENDING] Sent message: B6
[SENDING] Sent message: B7
[SENDING] Sent message: B8
[SENDING] Sent message: B9
[BUSY] Sensing channel to be busy
Waiting 2 s......
[IDLE] Sensing channel to be idle
[BUSY] Sensing channel to be busy
Waiting 5 s......
[IDLE] Sensing channel to be idle
```

P-persistent:

Channel Output:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Compu
ter Networks/Assignment 3/code/p-persistent$ python3 channel.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:46588
[ACTIVE COUNT] Thread Number: 1
[CONNECTED] Connected to: 127.0.0.1:46590
[ACTIVE COUNT] Thread Number: 2
[RECEIVED] Received message = B0
[RECEIVED] Received message = B1
[RECEIVED] Received message = A0
[RECEIVED] Received message = B2
[RECEIVED] Received message = A1
[RECEIVED] Received message = A2
[RECEIVED] Received message = A3
[RECEIVED] Received message = B3
[RECEIVED] Received message = B4
[RECEIVED] Received message = A4
[RECEIVED] Received message = B5
[RECEIVED] Received message = B6
[RECEIVED] Received message = B7
[RECEIVED] Received message = A5
[RECEIVED] Received message = B8
[RECEIVED] Received message = B9
[RECEIVED] Received message = A6
[RECEIVED] Received message = A7
[RECEIVED] Received message = A8
[RECEIVED] Received message = A9
```

Station 1:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Compu
ter Networks/Assignment 3/code/p-pecode/p-persistent$ python3 statio
n.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 1
[IDLE] Sensing channel to be idle
Probability is = 0.7
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.7
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.2
[ALLOWED] Transmission allowed
[SENDING] Sent message: A0
[SENDING] Sent message: A1
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.2
[ALLOWED] Transmission allowed
[SENDING] Sent message: A2
[SENDING] Sent message: A3
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.6
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.8
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.6
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.1
[ALLOWED] Transmission allowed
[SENDING] Sent message: A4
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 1.0
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.3
[ALLOWED] Transmission allowed
[SENDING] Sent message: A5
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 1.0
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.6
```

```
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.6
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.5
[ALLOWED] Transmission allowed
[SENDING] Sent message: A6
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.6
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 1.0
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.5
[ALLOWED] Transmission allowed
[SENDING] Sent message: A7
[SENDING] Sent message: A8
[SENDING] Sent message: A9
[ US ] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.8
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 1.0
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.7
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.4
[ALLOWED] Transmission allowed
```

Station 2:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Comp
- Computer Networks/Assignment 3/code/p-persistent$ python3 station
.py
[WAITING] Waiting for connection response
[CONNECTED] Server is working.
Enter fileno to start: 2
[IDLE] Sensing channel to be idle
Probability is = 0.9
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.1
[ALLOWED] Transmission allowed
[SENDING] Sent message: B0
[SENDING] Sent message: B1
[SENDING] Sent message: B2
[BUSY] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.8
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.9
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.5
[ALLOWED] Transmission allowed
[SENDING] Sent message: B3
[SENDING] Sent message: B4
[BUSY] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.9
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.7
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.4
[ALLOWED] Transmission allowed
[SENDING] Sent message: B5
[SENDING] Sent message: B6
[SENDING] Sent message: B7
[BUSY] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.2
[ALLOWED] Transmission allowed
[SENDING] Sent message: B8
[SENDING] Sent message: B9
[BUSY] Sensing channel to be busy
[IDLE] Sensing channel to be idle
Probability is = 0.6
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 1.0
```

```
[WAIT] Waiting for 3
[IDLE] Sensing channel to be idle
Probability is = 0.4
[ALLOWED] Transmission allowed
```

Comments:

In this assignment, we had to combine socket programming with multithreading in order to implement the various CSMA techniques.

Computer Networks Lab

Assignment 4 Report

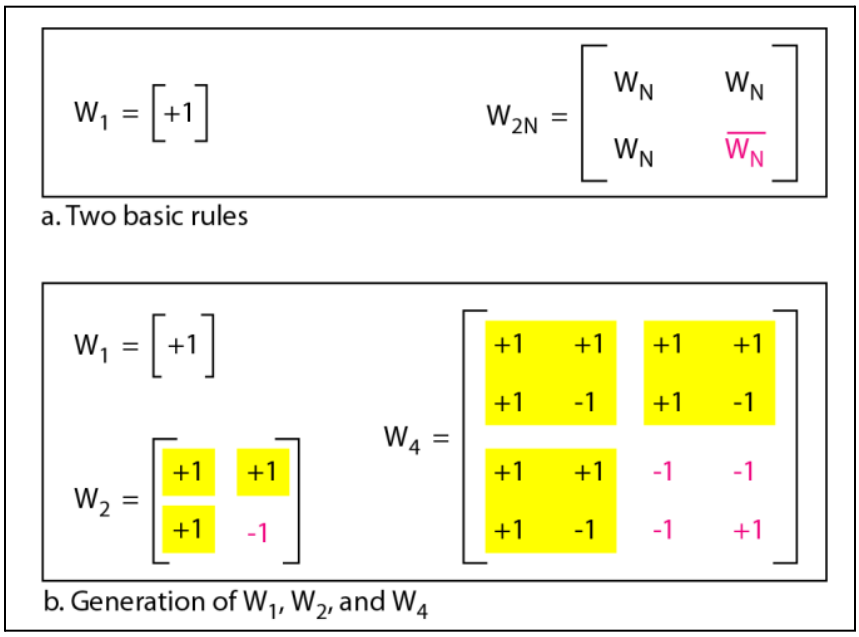
Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

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:

Walsh table is generated via a recursive rule as shown below:



In this program, walsh.txt will hold the walsh table, and channel.txt will hold the final bits in the channel after computation.

Implementation:

cdma.py:

```
import os, math

os.system("rm *.txt")   # refreshing all files

class Station: # Class for denoting every station

    def __init__(self, stn_num, num_data, cdma_code, frames):
        self.stn_num=stn_num
        self.num_data=num_data
        self.cdma_code=cdma_code
        self.frames=frames

    # Send data corresponding to index of frame list
    def sendData(self,index):

        if(self.frames[index]=='-'): # Meaning idle
            data=0
        elif(self.frames[index]=='0'):
            data=-1
        else:
            data=1

                            codeword=[data*self.cdma_code[i]   for   i   in
range(len(self.cdma_code))]
                                                print("[SENDING]      Station
{}:\t{}".format(self.stn_num,"\t".join(map(str, codeword))))
        return codeword

# Function to create walsh tables
def createWalsh(r):
```

```
    global walsh
    walsh=[[int(bin(x&y),13)%2 or -1 for x in range(r)]for y in range(r)]

# Decode dataword for every station
def decode_cdma(codeword, num_stn, max_num_stn):
    for i in range(num_stn):
        data=[codeword[j]*walsh[i][j] for j in range(len(walsh[i]))]
        data=sum(data)
        data=int(data/max_num_stn)
         with open("station_{}.txt".format(chr(65+i)), 'a') as opfile:   #
Writing decoded output in station
            if(data==-1):
                data=0
                opfile.write(str(data))
            elif(data==0):
                opfile.write("-")
            else:
                opfile.write(str(data))

def start():
    num_stn = int(input('Enter number of stations:\t'))
    x = num_stn
    if x&(x-1) != 0 and x != 0:
        print("Number must be power of 2.")
        exit()

    num_data = int(input('Enter the length of the message:\t'))

    createWalsh(num_stn)
    with open("walsh.txt", 'w') as opfile:
        for x in walsh:
            print(*x, sep='\t',file=opfile)

    stns=[]

    for i in range(num_stn):
         frames = input('Station {} | Enter the required string of length
{}:\t'.format(i,num_data))
        frames=list(frames)
```

```
        # Creating station object
        tempstn=Station(chr(65+i),num_data,walsh[i],frames)
        stns.append(tempstn)

    print("\nTransmitting")
    for i in range(num_stn):
        frames=num_data*'_'
        frames=list(frames)
        tempstn=Station(i,num_data,walsh[i],frames)
        stns.append(tempstn)


    # Send data for every data
    for i in range(num_data):
        code=[0 for i in range(num_stn)]
        # Send for every station
        for j in range(num_stn):
            code = [x+y for x,y in zip(code, stns[j].sendData(i))]
        print("SENT BIT {}\n".format(1+i))
        with open("channel.txt", 'a') as opfile:
             print("\t".join(map(str,code)),file=opfile)      # prints the
message passed through the channel

        decode_cdma(code,num_stn,num_stn)

walsh=[]
start()
print("\nTransmission Complete")
```

Output:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 4/code$ python3 cdma.py
Enter number of stations:       4
Enter the length of the message:        1
Station 0 | Enter the required string of length 1:      0
Station 1 | Enter the required string of length 1:      0
Station 2 | Enter the required string of length 1:      -
Station 3 | Enter the required string of length 1:      1

Transmitting
[SENDING] Station A:    -1      -1      -1      -1
[SENDING] Station B:    -1      1       -1      1
[SENDING] Station C:    0       0       0       0
[SENDING] Station D:    1       -1      -1      1
SENT BIT 1


Transmission Complete
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 4/code$
```

Bits in channel:

channel.txt | cdma.py

channel.txt

1	-1 -1 -3 1
2	

Walsh Matrix:

walsh.txt | channel.txt

walsh.txt

1	1	1	1	1
2	1	-1	1	-1
3	1	1	-1	-1
4	1	-1	-1	1

Comments:

In this assignment, we implemented the CDMA technique.

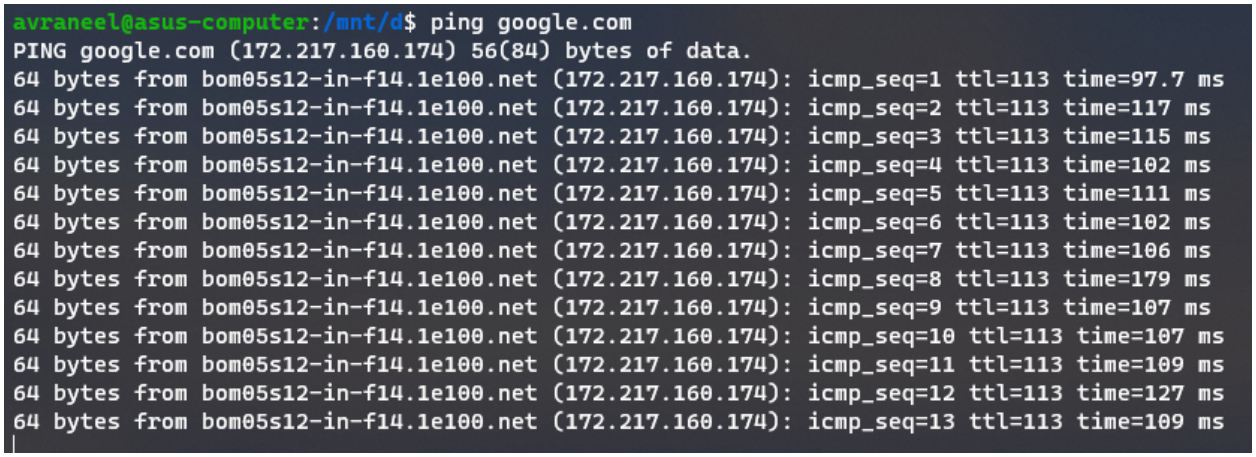
Computer Networks Lab

Assignment 5 Report

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statements and their Solutions:

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

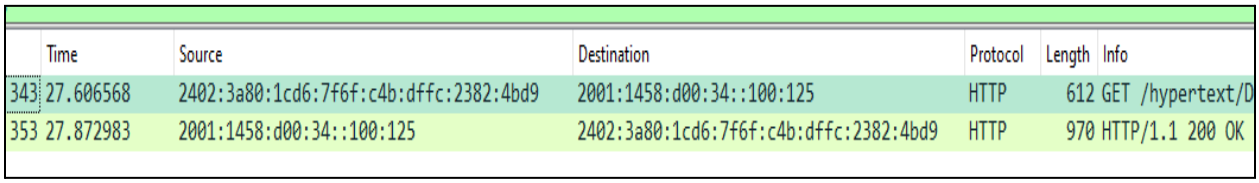


	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=61/15616, ttl=64 (reply in 2)
2	0.110286	172.217.160.174	172.20.66.164	ICMP	98	Echo (ping) reply id=0x0016, seq=61/15616, ttl=113 (request in 1)
3	1.002180	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=62/15872, ttl=64 (reply in 4)
4	1.124945	172.217.160.174	172.20.66.164	ICMP	98	Echo (ping) reply id=0x0016, seq=62/15872, ttl=113 (request in 3)
6	2.004072	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=63/16128, ttl=64 (reply in 7)
7	2.139150	172.217.160.174	172.20.66.164	ICMP	98	Echo (ping) reply id=0x0016, seq=63/16128, ttl=113 (request in 6)
8	3.006181	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=64/16384, ttl=64 (reply in 9)
9	3.106939	172.217.160.174	172.20.66.164	ICMP	98	Echo (ping) reply id=0x0016, seq=64/16384, ttl=113 (request in 8)
10	4.008036	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=65/16640, ttl=64 (reply in 11)
11	4.168503	172.217.160.174	172.20.66.164	ICMP	98	Echo (ping) reply id=0x0016, seq=65/16640, ttl=113 (request in 10)
12	5.010054	172.20.66.164	172.217.160.174	ICMP	98	Echo (ping) request id=0x0016, seq=66/16896, ttl=64 (reply in 13)

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.

	Time	Source	Destination	Protocol	Length	Info
22	1.917058	2402:3a80:1cd6:7f6f…	2404:6800:4003:c01:…	TCP	75	49331 → 5228 [ACK] Seq=1 Ack=1
21	1.717629	64:ff9b::b9c7:6c99	2402:3a80:1cd6:7f6f…	TCP	86	443 → 49324 [ACK] Seq=1 Ack=2 W
20	1.681387	192.168.100.167	52.182.143.67	TCP	54	49292 → 443 [ACK] Seq=2 Ack=2 W
19	1.681285	52.182.143.67	192.168.100.167	TCP	54	443 → 49292 [FIN, ACK] Seq=1 Ac
18	1.599067	2402:3a80:1cd6:7f6f…	64:ff9b::b9c7:6c99	TCP	75	49324 → 443 [ACK] Seq=1 Ack=1 W
17	1.443316	64:ff9b::b9c7:6c99	2402:3a80:1cd6:7f6f…	TCP	86	443 → 49318 [ACK] Seq=1 Ack=2 W
16	1.378334	192.168.100.167	52.182.143.67	TCP	54	49292 → 443 [FIN, ACK] Seq=1 Ac
15	1.329506	2402:3a80:1cd6:7f6f…	64:ff9b::b9c7:6c99	TCP	75	49318 → 443 [ACK] Seq=1 Ack=1 W
14	1.311700	2404:6800:4009:81f:…	2402:3a80:1cd6:7f6f…	TCP	86	443 → 49327 [ACK] Seq=1 Ack=2 W
13	1.234837	2402:3a80:1cd6:7f6f…	2404:6800:4009:81f:…	TCP	75	49327 → 443 [ACK] Seq=1 Ack=1 W
12	0.938029	172.217.160.163	192.168.100.167	TCP	66	443 → 49316 [ACK] Seq=1 Ack=2 W
11	0.841791	192.168.100.167	172.217.160.163	TCP	55	49316 → 443 [ACK] Seq=1 Ack=1 W
10	0.538769	2404:6800:4009:81f:…	2402:3a80:1cd6:7f6f…	TCP	86	443 → 49319 [ACK] Seq=1 Ack=2 W
9	0.462807	2402:3a80:1cd6:7f6f…	2404:6800:4009:81f:…	TCP	75	49319 → 443 [ACK] Seq=1 Ack=1 W
8	0.389646	2404:6800:4009:827:…	2402:3a80:1cd6:7f6f…	TCP	86	443 → 49313 [ACK] Seq=1 Ack=2 W
7	0.288162	2402:3a80:1cd6:7f6f…	2404:6800:4009:827:…	TCP	75	49313 → 443 [ACK] Seq=1 Ack=1 W
6	0.061718	2402:3a80:1cd6:7f6f…	2600:140f:2e00:2a7:…	TCP	74	49301 → 80 [ACK] Seq=2 Ack=2 Wi
5	0.061548	2600:140f:2e00:2a7:…	2402:3a80:1cd6:7f6f…	TCP	74	80 → 49301 [FIN, ACK] Seq=1 Ack
4	0.058150	2402:3a80:1cd6:7f6f…	2600:140f:2e00:2a7:…	TCP	74	49300 → 80 [ACK] Seq=2 Ack=2 Wi
3	0.057979	2600:140f:2e00:2a7:…	2402:3a80:1cd6:7f6f…	TCP	74	80 → 49300 [FIN, ACK] Seq=1 Ack
2	0.000388	2402:3a80:1cd6:7f6f…	2600:140f:2e00:2a7:…	TCP	74	49301 → 80 [FIN, ACK] Seq=1 Ack
1	0.000000	2402:3a80:1cd6:7f6f…	2600:140f:2e00:2a7:…	TCP	74	49300 → 80 [FIN, ACK] Seq=1 Ack
69	6.700127	64:ff9b::3dd:3461	2402:3a80:1cd6:7f6f…	SSL	80	Continuation Data
68	6.700127	64:ff9b::3dd:3461	2402:3a80:1cd6:7f6f…	SSL	80	Continuation Data
92	9.460979	2402:3a80:1cd6:7f6f…	fe80::ecac:64ff:fe1…	ICMPv6	86	Neighbor Advertisement 2402:3a8
91	9.460600	fe80::ecac:64ff:fe1…	2402:3a80:1cd6:7f6f…	ICMPv6	86	Neighbor Solicitation for 2402:
124	14.355982	192.168.100.45	192.168.100.167	DNS	138	Standard query response 0xdb12
122	14.306906	192.168.100.45	192.168.100.167	DNS	109	Standard query response 0x3071
121	14.306906	192.168.100.45	192.168.100.167	DNS	97	Standard query response 0xa29b
120	14.303651	192.168.100.167	192.168.100.45	DNS	81	Standard query 0xdb12 HTTPS upd
119	14.303363	192.168.100.167	192.168.100.45	DNS	81	Standard query 0x3071 AAAA upda
118	14.302998	192.168.100.167	192.168.100.45	DNS	81	Standard query 0xa29b A update.
168	15.093029	AzureWav_9c:f3:51	ee:ac:64:1c:dd:39	ARP	42	192.168.100.167 is at 34:6f:24:

b. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received?



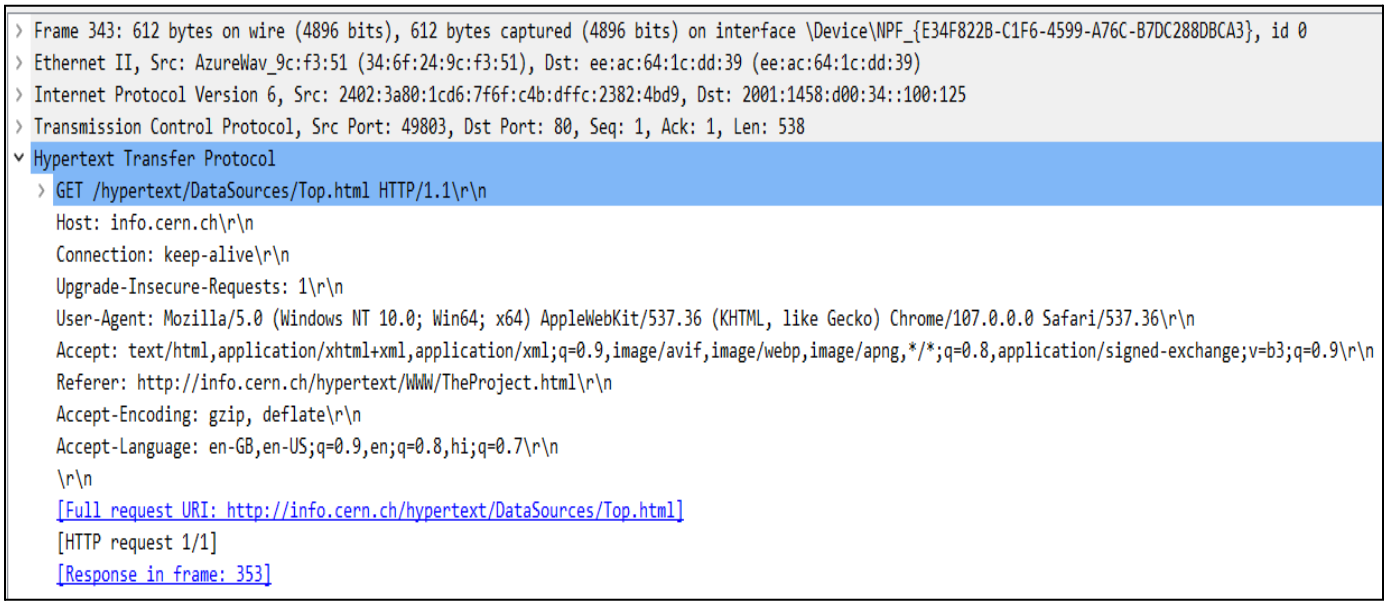
	Time	Source	Destination	Protocol	Length	Info
343	27.606568	2402:3a80:1cd6:7f6f:c4b:dffc:2382:4bd9	2001:1458:d00:34::100:125	HTTP	612	GET /hypertext/D
353	27.872983	2001:1458:d00:34::100:125	2402:3a80:1cd6:7f6f:c4b:dffc:2382:4bd9	HTTP	970	HTTP/1.1 200 OK

Time = 27.872983 - 27.606568 = **0.266415 s**

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

My computer's address = 2402:3a80:1cd6:7f6f:c4b:dffc:2382:4bd9
The Website's address = 2001:1458:d00:34:100:125

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.

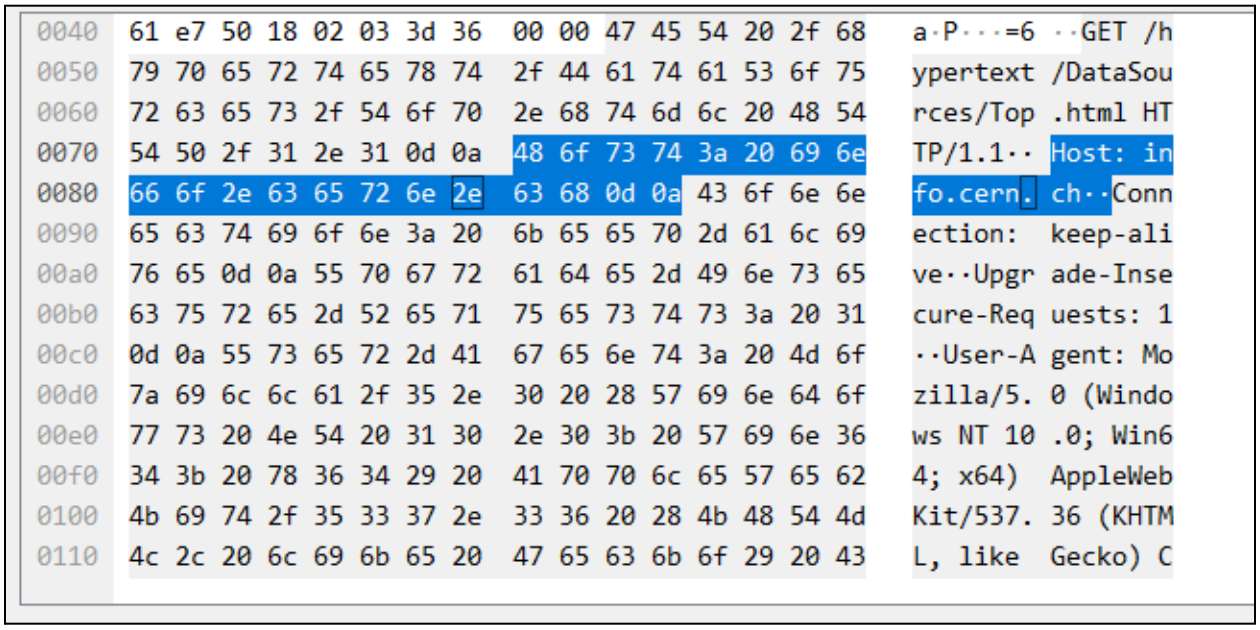


```
> Frame 343: 612 bytes on wire (4896 bits), 612 bytes captured (4896 bits) on interface \Device\NPF_{E34F822B-C1F6-4599-A76C-B7DC288DBCA3}, id 0
> Ethernet II, Src: AzureWav_9c:f3:51 (34:6f:24:9c:f3:51), Dst: ee:ac:64:1c:dd:39 (ee:ac:64:1c:dd:39)
> Internet Protocol Version 6, Src: 2402:3a80:1cd6:7f6f:c4b:dffc:2382:4bd9, Dst: 2001:1458:d00:34::100:125
> Transmission Control Protocol, Src Port: 49803, Dst Port: 80, Seq: 1, Ack: 1, Len: 538
v Hypertext Transfer Protocol
  > GET /hypertext/DataSources/Top.html HTTP/1.1\r\n
    Host: info.cern.ch\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
    Referer: http://info.cern.ch/hypertext/WWW/TheProject.html\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,hi;q=0.7\r\n
    \r\n
    [Full request URI: http://info.cern.ch/hypertext/DataSources/Top.html]
    [HTTP request 1/1]
    [Response in frame: 353]
```

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

Host = info.cern.ch\r\n

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.

```
GET /hypertext/WWW/WhatIs.html HTTP/1.1
Host: info.cern.ch
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
Referer: http://info.cern.ch/hypertext/WWW/TheProject.html
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,hi;q=0.7

HTTP/1.1 200 OK
Date: Wed, 23 Nov 2022 15:16:26 GMT
Server: Apache
Last-Modified: Mon, 18 May 1992 13:38:38 GMT
ETag: "47d-2824824ad1380"
Accept-Ranges: bytes
Content-Length: 1149
Connection: close
Content-Type: text/html

<TITLE>What is Hypertext?</TITLE>
<NEXTID 20>
<H1>What is HyperText</H1>Hypertext is text which is not constrained to be linear.<P>
Hypertext is text which contains <A NAME=0 HREF=Terms.html#link>links</A> to other texts. The term was
coined by <A NAME=1 HREF=Xanadu.html#Nelson>Ted Nelson</A> around 1965 (see <A NAME=12 HREF=../History.html>History</A> ).<P>
HyperMedia is a term used for hypertext which is not constrained to
be text: it can include graphics, video and <A NAME=9 HREF=Talks/YesWeCan.snd>sound</A> , for example. Apparently
Ted Nelson was the first to use this term too.<P>
Hypertext and HyperMedia are concepts, not products.<P>
See also:
<UL>
<LI><A NAME=2 HREF=Terms.html>A list of terms</A> used in hypertext litterature.
<LI><A NAME=19 HREF=../Conferences/Overview.html>Conferences</A>
<LI><A NAME=7 HREF=../Products/Overview.html>Commercial (and academic) products</A>
<LI>A newsgroup on hypertext, <A NAME=5 HREF=news:alt.hypertext>"alt.hypertext"</A> .
<LI><A NAME=4 HREF=TheProject.html>WorldWideWeb is a project</A>  which uses hypertext concepts.
<LI><A NAME=10 HREF=../Standards/Overview.html>Standards</A> .</A>
</UL>
```

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.

```
> Frame 1284: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{E34F822B-C1F6-4599-A76C-B7DC288DBCA3}, id 0
v Ethernet II, Src: f6:0e:60:ac:ca:96 (f6:0e:60:ac:ca:96), Dst: AzureWav_9c:f3:51 (34:6f:24:9c:f3:51)
  v Destination: AzureWav_9c:f3:51 (34:6f:24:9c:f3:51)
      Address: AzureWav_9c:f3:51 (34:6f:24:9c:f3:51)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: f6:0e:60:ac:ca:96 (f6:0e:60:ac:ca:96)
      Address: f6:0e:60:ac:ca:96 (f6:0e:60:ac:ca:96)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv6 (0x86dd)
> Internet Protocol Version 6, Src: 2001:1458:d00:34::100:125, Dst: 2409:4060:2d83:ed6b:8c0d:9bf7:ccbe:92e7
> Transmission Control Protocol, Src Port: 80, Dst Port: 52646, Seq: 1371, Ack: 534, Len: 12
> [2 Reassembled TCP Segments (1382 bytes): #1283(1370), #1284(12)]
> Hypertext Transfer Protocol
> Line-based text data: text/html (18 lines)
```

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

   Manufacturer of my NIC - AzureWav_9c:f3:51 (34:6f:24:9c:f3:51)

   Manufacturer of server NIC - f6:0e:60:ac:ca:96 (f6:0e:60:ac:ca:96)

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 = **34:6f:24:9c:f3:51**

   Hex value of server's OUI = **f6:0e:60:ac:ca:96**

9. Find the following statistics:

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

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

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	3878	100.0	2550894	688 k	0	0	0	3878
Ethernet	100.0	3878	2.1	54292	14 k	0	0	0	3878
Internet Protocol Version 6	70.0	2714	4.3	108560	29 k	0	0	0	2714
User Datagram Protocol	48.2	1871	0.6	14968	4040	0	0	0	1871
QUIC IETF	48.2	1870	58.9	1502243	405 k	1870	1491791	402 k	1888
Multicast Domain Name System	0.0	1	0.0	40	10	1	40	10	1
Transmission Control Protocol	21.6	839	13.6	347726	93 k	481	109463	29 k	839
Transport Layer Security	9.0	350	13.6	346868	93 k	350	327127	88 k	359
Hypertext Transfer Protocol	0.2	8	0.3	7758	2094	4	1880	507	8
Media Type	0.0	1	0.1	1406	379	1	1406	379	1
Line-based text data	0.1	3	0.1	3527	952	3	3527	952	3
Internet Control Message Protocol v6	0.1	4	0.0	128	34	4	128	34	4
Internet Protocol Version 4	30.0	1162	0.9	23240	6273	0	0	0	1162
User Datagram Protocol	5.4	208	0.1	1664	449	0	0	0	208
Simple Service Discovery Protocol	0.1	3	0.0	525	141	3	525	141	3
Multicast Domain Name System	0.0	1	0.0	40	10	1	40	10	1
Domain Name System	5.3	204	0.6	16280	4394	204	16280	4394	204
Transmission Control Protocol	24.6	954	18.9	483169	130 k	656	362835	97 k	954
Transport Layer Security	7.7	298	18.1	461399	124 k	298	435880	117 k	303
Address Resolution Protocol	0.1	2	0.0	56	15	2	56	15	2

a. Percentage of TCP packets in IPv6 = **21.6%**
   Percentage of TCP packets in IPv4 = **24.6%**

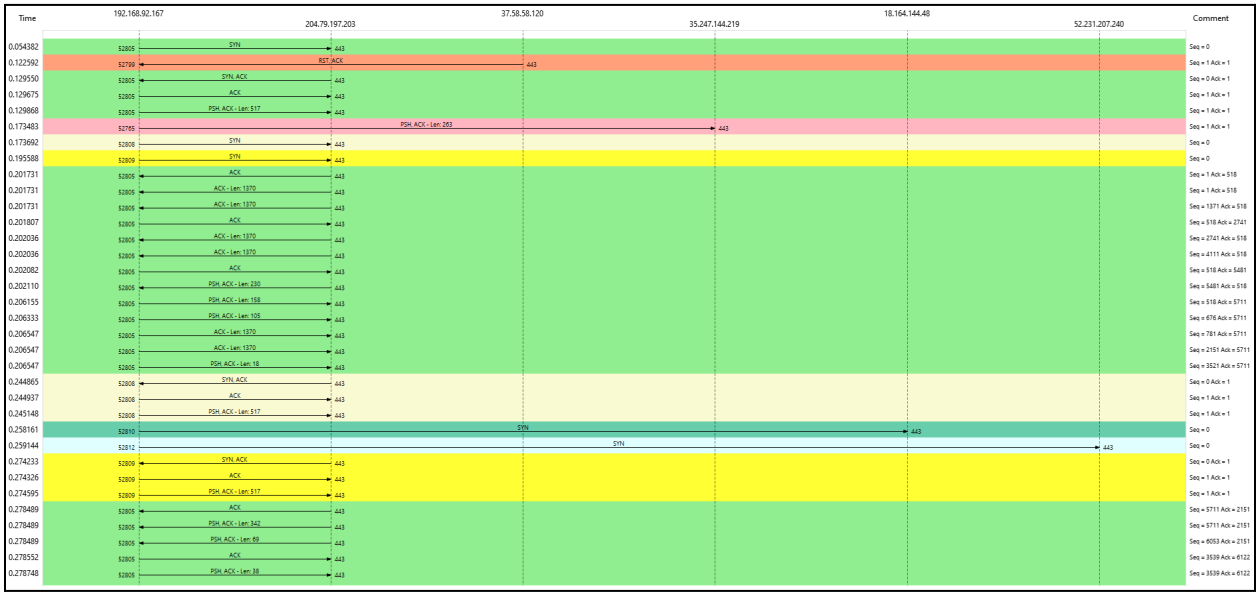
Examples of higher level protocols in TCP are **HTTP** and **FTP** protocols

b. Percentage of UDP packets in IPv6 = **48.2%**
Percentage of UDP packets in IPv4 = **5.4%**

Example of higher level protocol in UDP is **SNMP** protocol.

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

Showing the first few packets



Comments:

In this assignment, we used the Wireshark tool to analyze network packets in a practical situation.

Computer Networks Lab

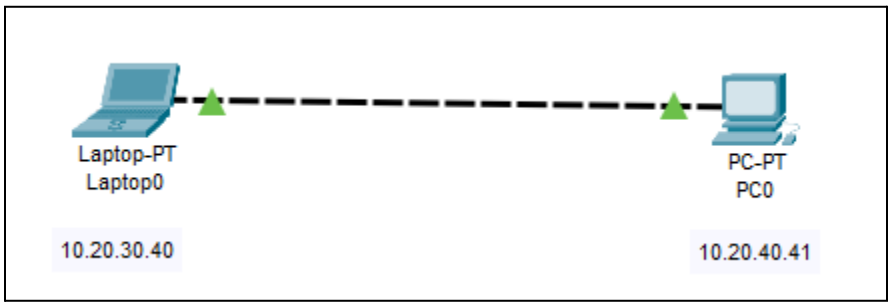
Assignment 6 Report

Submitted by,

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

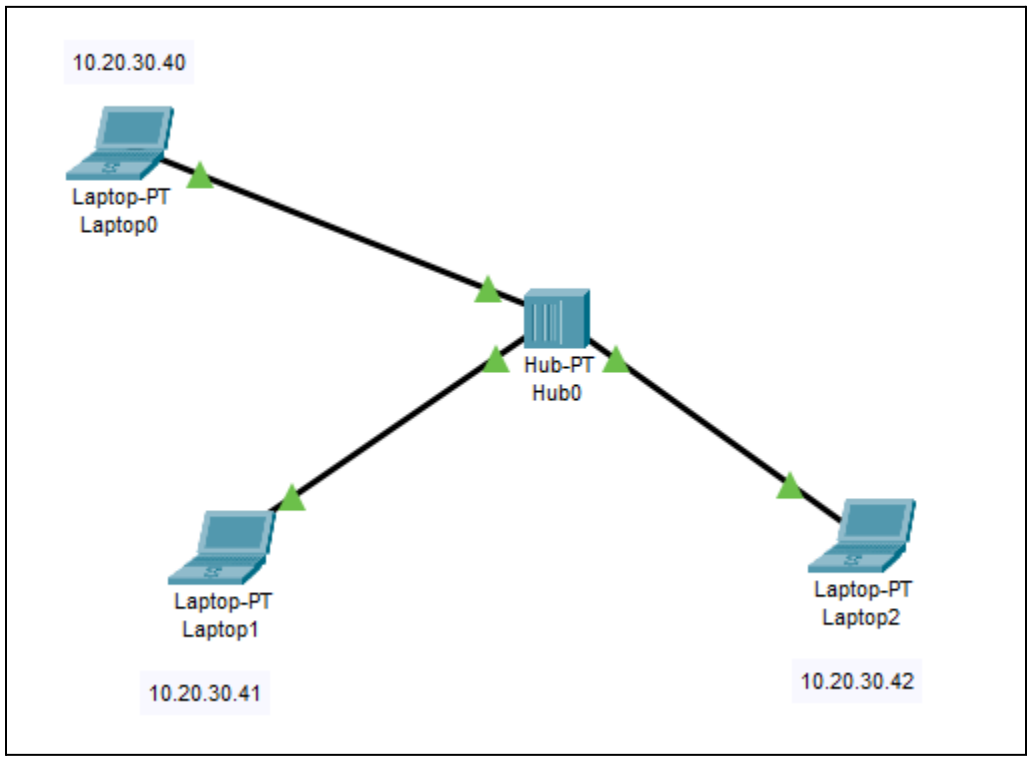
Problem Statements and their Solutions:

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



```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 10.20.40.41

Pinging 10.20.40.41 with 32 bytes of data:

Reply from 10.20.40.41: bytes=32 time<1ms TTL=128
Reply from 10.20.40.41: bytes=32 time<1ms TTL=128
Reply from 10.20.40.41: bytes=32 time<1ms TTL=128
Reply from 10.20.40.41: bytes=32 time<1ms TTL=128

Ping statistics for 10.20.40.41:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

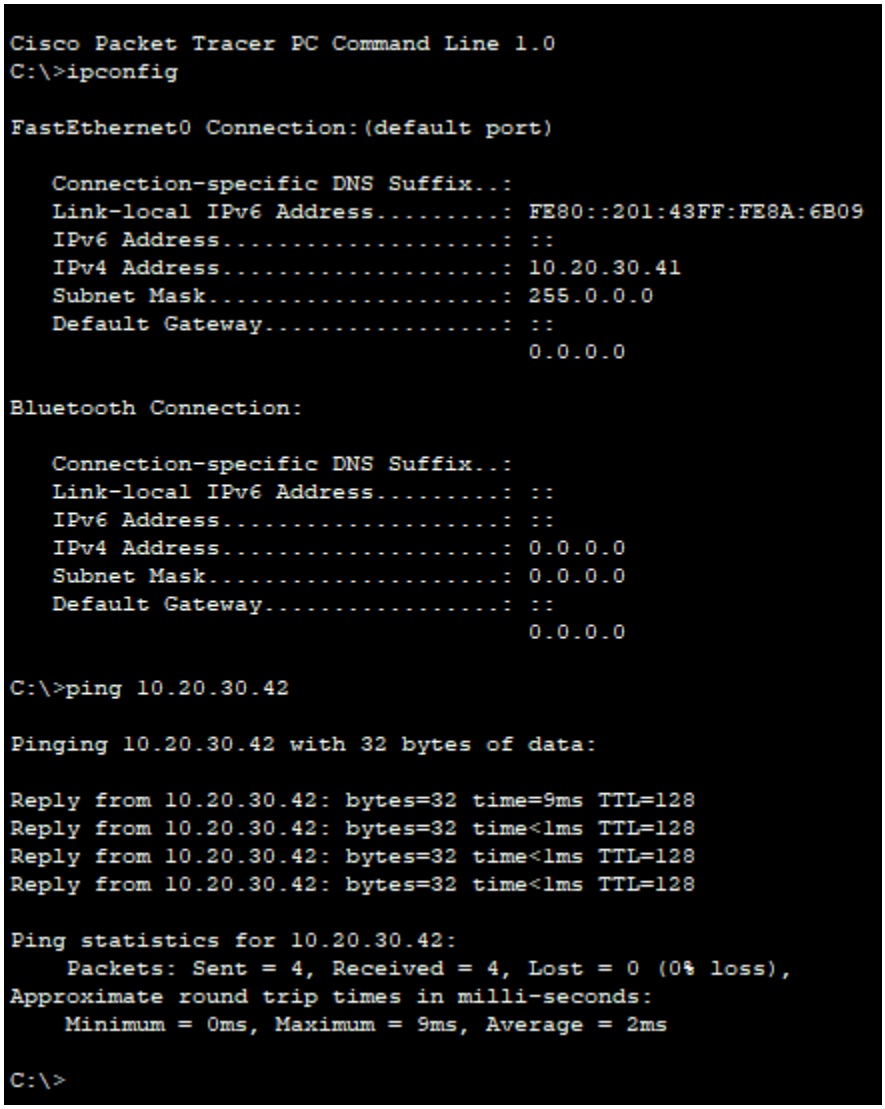
2. Create a LAN (named LAN-A) with 3 hosts using a hub. Ping each pair of nodes.



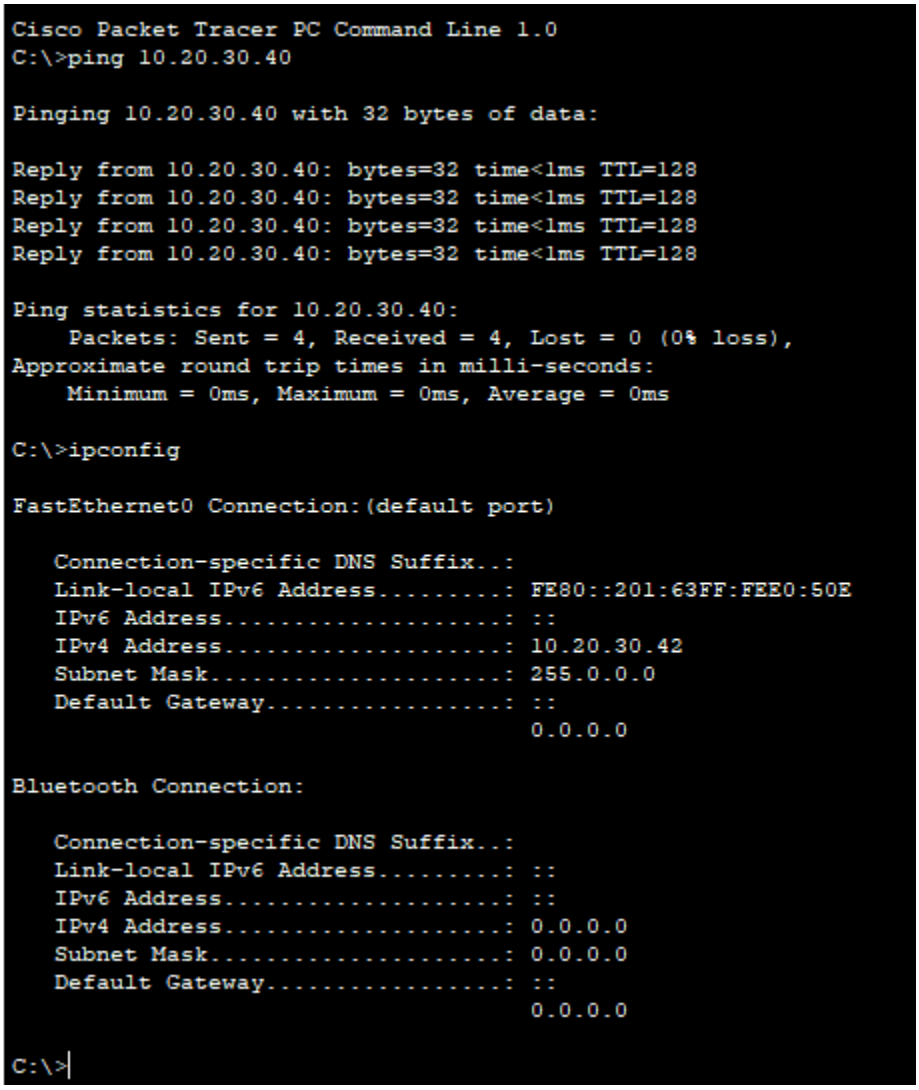
Pinging Laptop0 to Laptop1

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

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.....: FE80::20D:BDFF:FE69:9955
   IPv6 Address.....: ::
   IPv4 Address.....: 10.20.30.40
   Subnet Mask.....: 255.0.0.0
   Default Gateway.....: ::
                                 0.0.0.0

Bluetooth Connection:

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.....: ::
   IPv6 Address.....: ::
   IPv4 Address.....: 0.0.0.0
   Subnet Mask.....: 0.0.0.0
   Default Gateway.....: ::
                                 0.0.0.0

C:\>ping 10.20.30.41

Pinging 10.20.30.41 with 32 bytes of data:

Reply from 10.20.30.41: bytes=32 time<1ms TTL=128
Reply from 10.20.30.41: bytes=32 time<1ms TTL=128
Reply from 10.20.30.41: bytes=32 time<1ms TTL=128
Reply from 10.20.30.41: bytes=32 time<1ms TTL=128

Ping statistics for 10.20.30.41:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

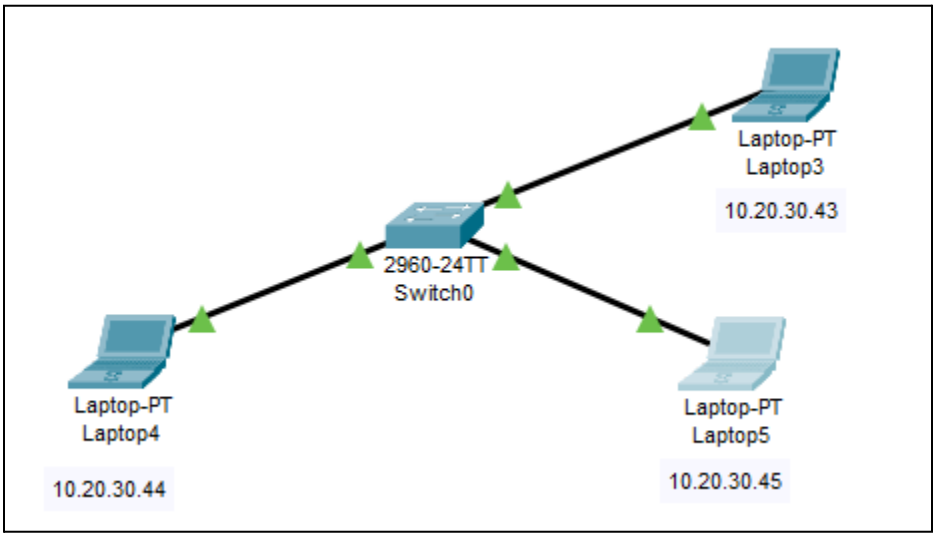
Pinging Laptop1 to Laptop2:



Pinging Laptop2 to Laptop0:



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.



Initially,
ARP table of each host:

```
Cisco Packet Tracer PC Command Line 1.0
C:\>arp -a
No ARP Entries Found
C:\>
```

Initially MAC address table of switch:

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

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

Switch#
```

After pinging,

ARP table of Laptop 3:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.44           0006.2a6a.c8bc        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

ARP table of Laptop 4:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

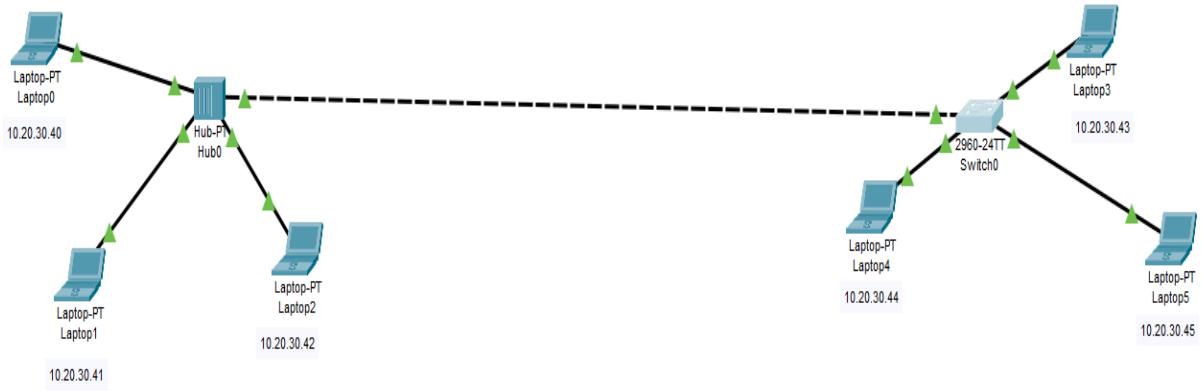
ARP table of Laptop 5:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
```

MAC Address Table of Switch:

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

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

   1    0002.1698.bc6d    DYNAMIC     Fa0/3
   1    0006.2a6a.c8bc    DYNAMIC     Fa0/2
   1    0030.a36c.436b    DYNAMIC     Fa0/1
```

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



Arp Tables of:

Laptop 0:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.41           0001.438a.6b09        dynamic
  10.20.30.42           0001.63e0.050e        dynamic
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

Laptop 1:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.40           000d.bd69.9955        dynamic
  10.20.30.42           0001.63e0.050e        dynamic
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

Laptop 2:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.40           000d.bd69.9955        dynamic
  10.20.30.41           0001.438a.6b09        dynamic
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

Laptop 3:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.40           000d.bd69.9955        dynamic
  10.20.30.41           0001.438a.6b09        dynamic
  10.20.30.42           0001.63e0.050e        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

Laptop 4:

```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.40           000d.bd69.9955        dynamic
  10.20.30.41           0001.438a.6b09        dynamic
  10.20.30.42           0001.63e0.050e        dynamic
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.45           0002.1698.bc6d        dynamic
```

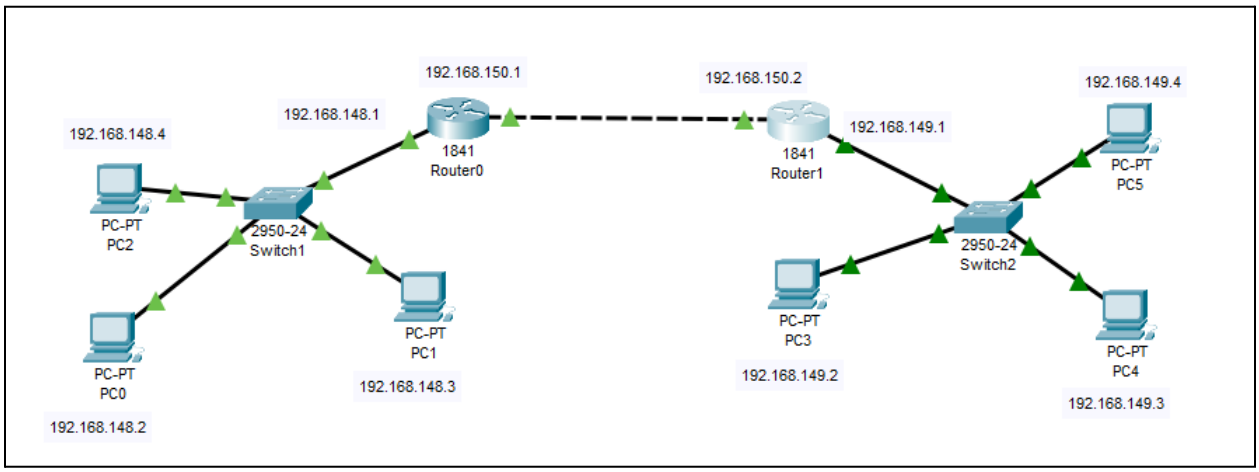
Laptop 5:

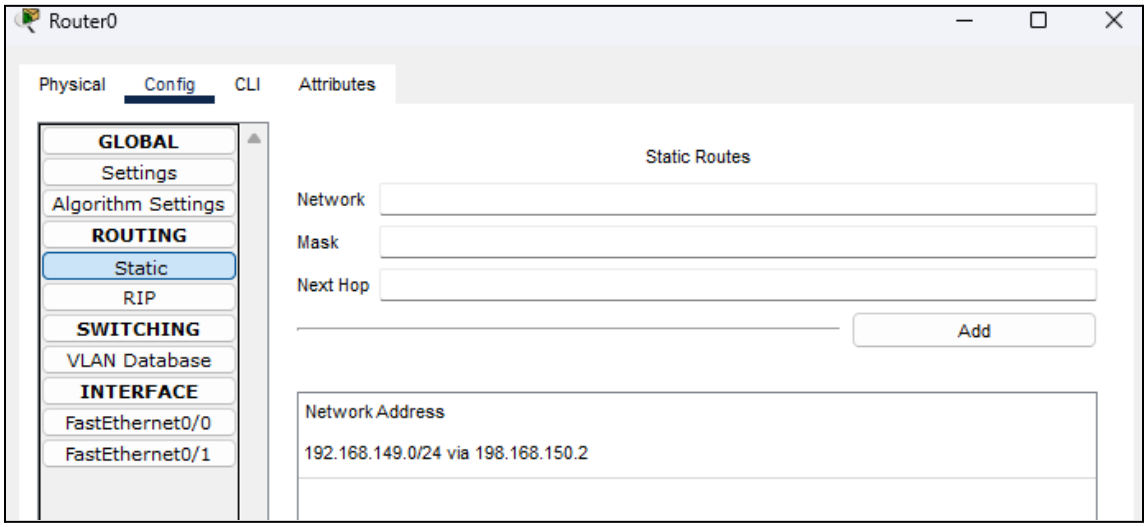
```
C:\>arp -a
  Internet Address      Physical Address      Type
  10.20.30.40           000d.bd69.9955        dynamic
  10.20.30.41           0001.438a.6b09        dynamic
  10.20.30.42           0001.63e0.050e        dynamic
  10.20.30.43           0030.a36c.436b        dynamic
  10.20.30.44           0006.2a6a.c8bc        dynamic
```

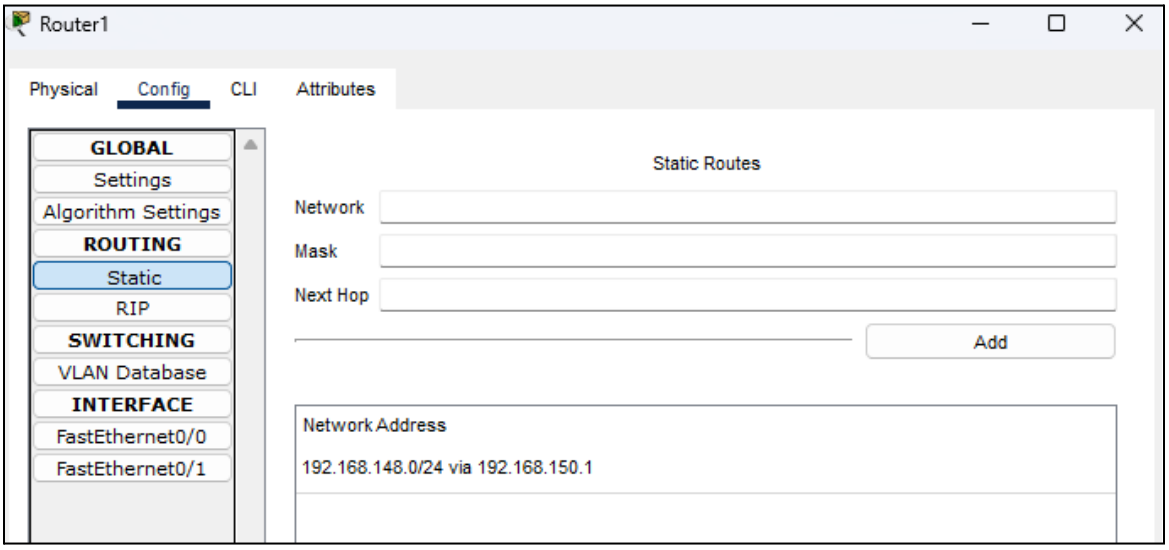
MAC forwarding table of switch:

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

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

   1    0001.438a.6b09    DYNAMIC     Fa0/4
   1    0001.63e0.050e    DYNAMIC     Fa0/4
   1    0002.1698.bc6d    DYNAMIC     Fa0/3
   1    0006.2a6a.c8bc    DYNAMIC     Fa0/2
   1    000d.bd69.9955    DYNAMIC     Fa0/4
   1    0030.a36c.436b    DYNAMIC     Fa0/1
Switch#
```

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.


192.168.150.1
192.168.150.2
192.168.149.4
192.168.148.1
192.168.149.1
192.168.148.4
1841
Router0
1841
Router1
PC-PT
PC5
PC-PT
PC2
2950-24
Switch1
2950-24
Switch2
PC-PT
PC3
PC-PT
PC1
PC-PT
PC4
PC-PT
PC0
192.168.148.3
192.168.149.2
192.168.149.3
192.168.148.2

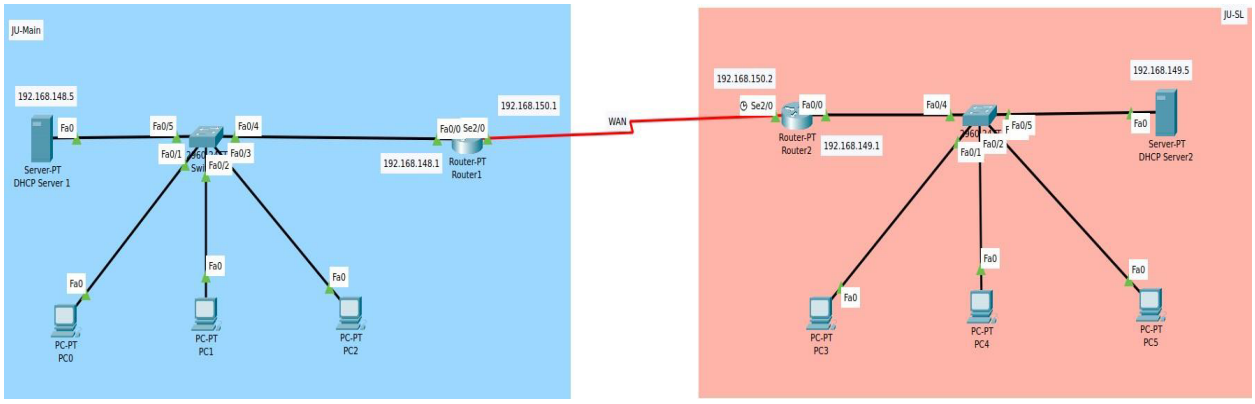

Router0
Physical
Config
CLI
Attributes
GLOBAL
Settings
Algorithm Settings
ROUTING
Static
RIP
SWITCHING
VLAN Database
INTERFACE
FastEthernet0/0
FastEthernet0/1
Static Routes
Network
Mask
Next Hop
Add
Network Address
192.168.149.0/24 via 198.168.150.2


Router1
Physical
Config
CLI
Attributes
GLOBAL
Settings
Algorithm Settings
ROUTING
Static
RIP
SWITCHING
VLAN Database
INTERFACE
FastEthernet0/0
FastEthernet0/1
Static Routes
Network
Mask
Next Hop
Add
Network Address
192.168.148.0/24 via 192.168.150.1

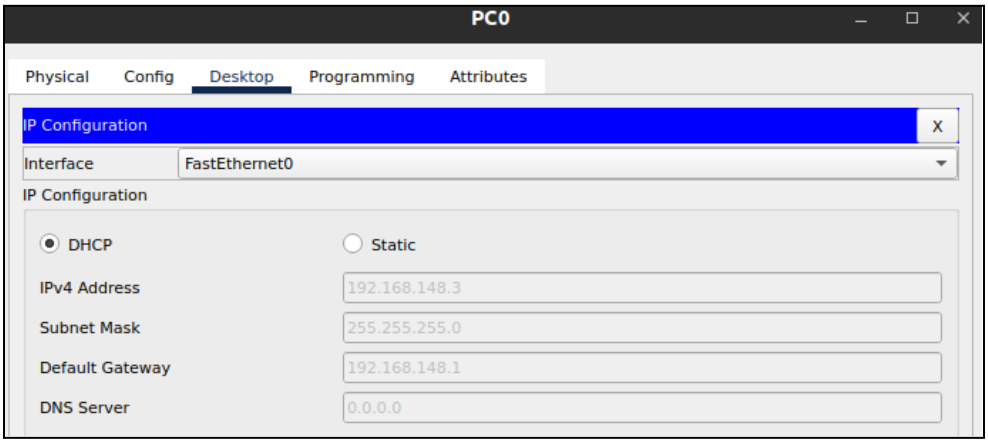
PC-0 pinging to Router-1: successful

```
C:\>ping 192.168.150.2

Pinging 192.168.150.2 with 32 bytes of data:

Reply from 192.168.150.2: bytes=32 time<1ms TTL=254
Reply from 192.168.150.2: bytes=32 time<1ms TTL=254
Reply from 192.168.150.2: bytes=32 time<1ms TTL=254
Reply from 192.168.150.2: bytes=32 time<1ms TTL=254

Ping statistics for 192.168.150.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

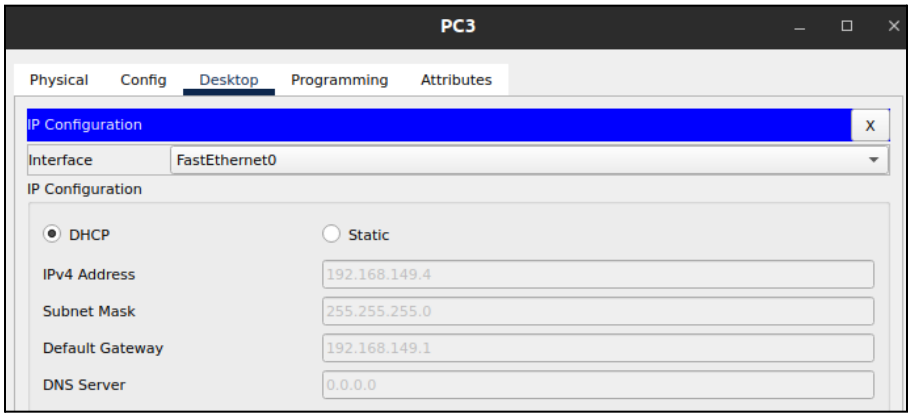
6. Add servers to the individual LANs (in problem 5) and configure them as a DHCP server. Configure the hosts in the individual LAN to obtain IP addresses and address of the default gateway via this DHCP server.



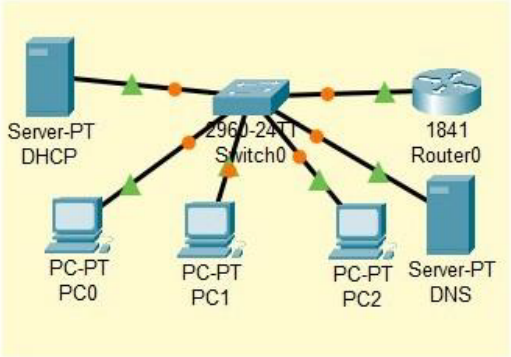
DHCP Server 1 is working. Here is a proof of working. DHCP Server has allocated ip 192.168.148.3 to PC0.



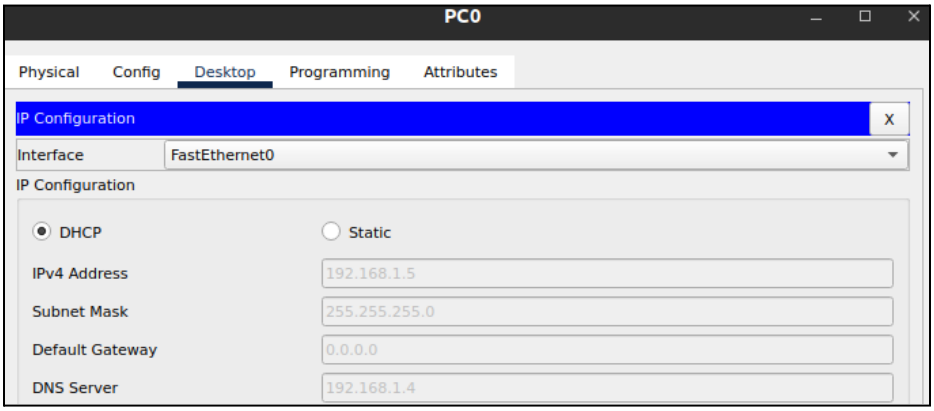
DHCP Server 2 is also working. DHCP Server 2 has allocated ip 192.168.149.4 to PC3.



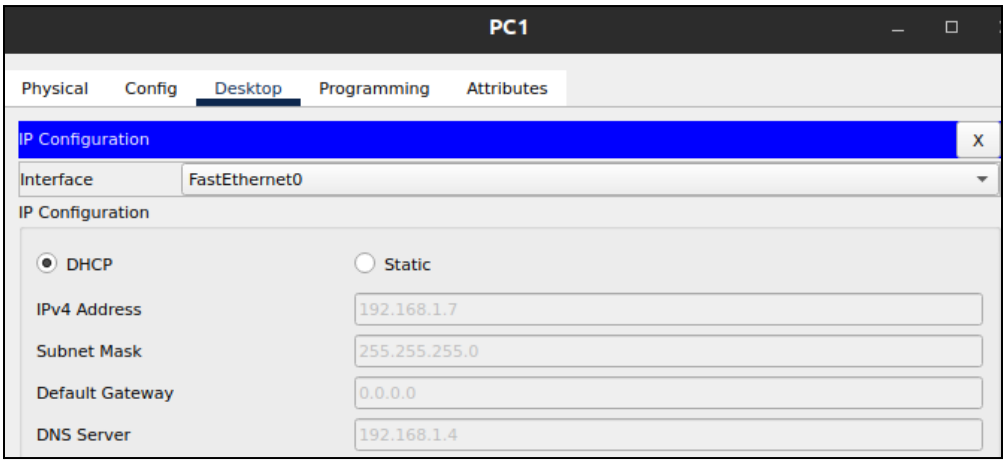
7. Create a LAN (CSE) with three hosts connected via a layer-2 switch (Cisco 2950 switch CSE-Switch). Also add a web server and a ftp server to this LAN. The hosts dynamically get their IP addresses from a local DHCP server. Servers are assigned fixed IP addresses. Configure the individual hosts to use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



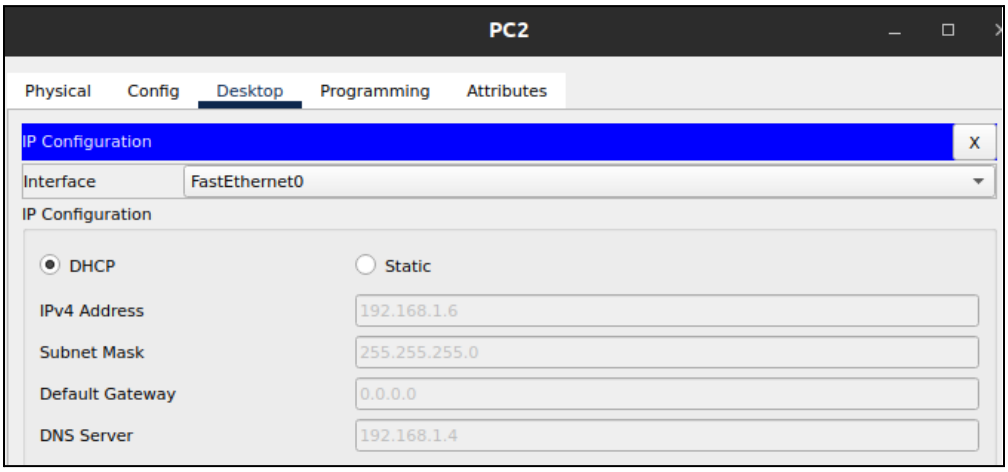
DHCP Server has allocated 192.168.1.5 to PC0



DHCP Server has allocated 192.168.1.7 to PC1

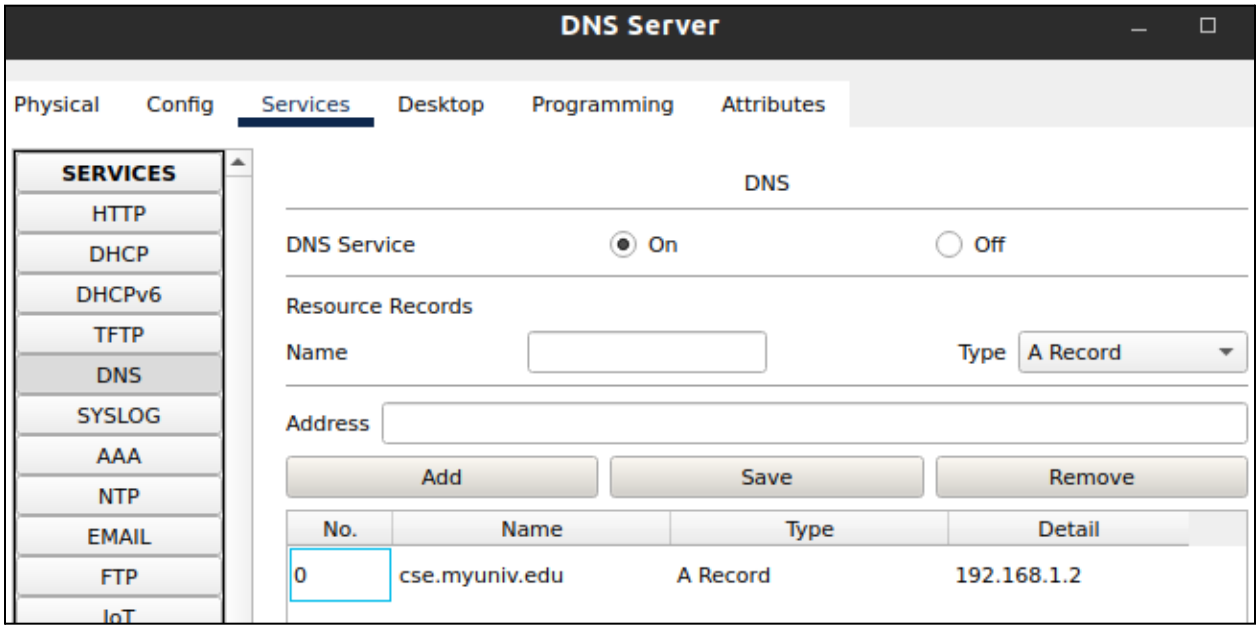


DHCP Server has allocated 192.168.1.5 to PC2

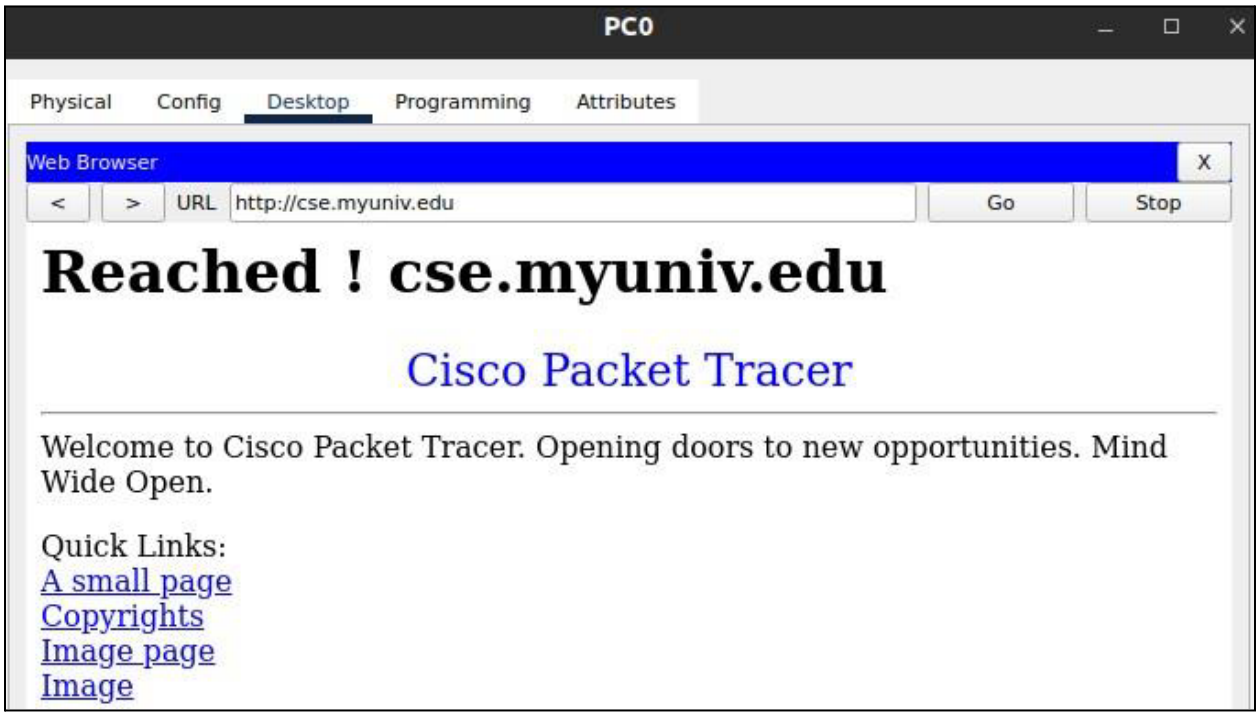


DNS Server : 192.168.1.4
FTP Server : 192.168.1.3
Web Server : 192.168.1.2

DNS Server Config



From PC0 accessing *cse.myuniv.edu*



Comments:

In this assignment we learnt how to simulate various network structures with CISCO Packet Tracer.

Computer Networks Lab

Assignment 7 Report

Submitted by,

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

Implement any two protocols using TCP/UDP Socket as suitable.
1. ARP
2. BOOTP
3. DHCP

Implementation:

ARP Protocol:

Design:

Sender stores ip and mac addresses in a dictionary. Receiver sends ip address to sender and returns the corresponding mac address.

Stats.py:

```
import socket
import time
import random

# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)
```

Sender.py:

```
import socket, stats as st, threading

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.bind(st.ADDR)

addrDict = {
    "192.1.67.8" : "00:00:5e:00:53:af",
    "88.255.145.56" : "00:1b:63:84:45:e6",
    "55.67.32.11" : "00:11:20:c5:bc:b0",
    "11.34.128.128" : "53:ff:c1:7a:8b:9c"
}

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

conn, address = channel.accept()

def send(conn):
    global addrDict
    ip = conn.recv(1024).decode()
    print(f"[RECEIVED] Received mac address {ip}")
    try:
        mac = addrDict[ip]
    except:
        mac = "Doesn't exist in table"
```

```
    print(f"[SENDING] Sending ip address...")
    conn.send(mac.encode())

print('[CONNECTED] Connected to: ' + address[0] + ':' + str(address[1]))
client_thread = threading.Thread(target=send, args=(conn, ))
client_thread.start()
client_thread.join()
```

Receiver.py:

```
import socket, os, threading, stats as st

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.connect(st.ADDR)

def send(channel):
    ip = input("Enter ip address = ")
    channel.send(ip.encode())
    mac = channel.recv(1024).decode()
    print(f"[RECEIVED] Your mac address is: {mac}")

send(channel)
```

Test cases:

Sender side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$ python
3 sender.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:45468
[ ECEI ED]  eceived  ac address 55.67.32.11
[SENDING] Sending ip address...
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$ python
3 sender.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:45470
[ ECEI ED]  eceived  ac address 88.255.145.56
[SENDING] Sending ip address...
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$
```

Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$ pyth
on3 receiver.py
Enter ip address = 55.67.32.11
[RECEIVED] Your mac address is: 00:11:20:c5:bc:b0
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$ pyth
on3 receiver.py
Enter ip address = 88.255.145.56
[RECEIVED] Your mac address is: 00:1b:63:84:45:e6
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/ARP$
```

BOOTP Protocol:

Design:

Server stores ip and mac addresses in a dictionary. Client sends ip address to sender and returns the corresponding mac address of that client and also that of the server.

Implementation:

Stats.py:

```
import socket
import time
import random

# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)
```

Server.py:

```
import socket, stats as st, threading
```

```
channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.bind(st.ADDR)

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

conn, address = channel.accept()

server_ip = "140.78.129.31"

addrDict = {
    "2150::0020::3415::30cf" : "192.1.67.8",
    "1000::2000::3000::2a00" : "88.255.145.56",
    "3c56a::80bb::ac7c::5921" : "55.67.32.11",
    "8a72::b052::410c::6cce" : "11.34.128.128"
}

def send(conn):
    global addrDict, server_ip
    mac = conn.recv(1024).decode()
    print(f"[RECEIVED] Received MAC address {mac}")
    try:
        ip = addrDict[mac]
    except:
        ip = "Doesn'tt exist in table"
    print(f"[SENDING] Sending IP address...")
    conn.send(ip.encode())
    conn.send(server_ip.encode())

print('[CONNECTED] Connected to: ' + address[0] + ':' +
str(address[1]))
client_thread = threading.Thread(target=send, args=(conn, ))
client_thread.start()
client_thread.join()
```

client.py:

```
import socket, os, threading, stats as st

ThreadCount = 0

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.connect(st.ADDR)

mac = "3c56a::80bb::ac7c::5921"

def send(channel):
    global mac
    print("[REQUESTING] Sending MAC address: ", mac)
    channel.send(mac.encode())
    ip = channel.recv(1024).decode()
    server_ip = channel.recv(1024).decode()
    print(f"[RECEIVED] My ip address is: {ip}")
    print(f"[RECEIVED] Server ip address is: {server_ip}")

send(channel)
```

Test cases:

Server side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/BOOTP$ pyth
on3 server.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:46592
[RECEIVED] Received MAC address 3c56a::80bb::ac7c::5921
[SENDING] Sending IP address...
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/BOOTP$
```

Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/BOOTP$ py
thon3 client.py
[REQUESTING] Sending MAC address:  3c56a::80bb::ac7c::5921
[RECEIVED] My ip address is: 55.67.32.11140.78.129.31
[RECEIVED] Server ip address is:
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 7/BOOTP$
```

Comments:

This assignment helped us to learn and implement ARP and BOOTP Protocols.

Computer Networks Lab

Assignment 8 Report

Name: Avraneel Pal
Roll: 002010501047
Department: BCSE
Group: A2
Semester: 3rd Year Semester 1
Year: 2020-24

Problem Statement:

Implement any two protocols using TCP/UDP Socket as suitable.
1. FTP
2. DNS
3. Telnet

Solution:

FTP Protocol:

Design:

There is a sender file “sender.py” and a receiver file “receiver.py” and a common stats.py file which holds the hostname and port number.

There are two files “file1.txt” and “file2.txt”.

The sender file will extract the contents of file1 and send it via socket to receiver file which will write it to “file2.txt”

Implementation:

sender.py:

```
import socket, os, threading, stats as st

ThreadCount = 0

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.bind(st.ADDR)

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

def send(conn):
    f = open("file1.txt", "r")
    data = f.read()
    print("[SENDING] Sending file....")
    conn.send(data.encode())

conn, address = channel.accept()
print('[CONNECTED] Connected to: ' + address[0] + ':' +
str(address[1]))
client_thread = threading.Thread(target=send, args=(conn, ))
client_thread.start()
client_thread.join()
```

receiver.py:

```
import socket, os, threading, stats as st

ThreadCount = 0

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.connect(st.ADDR)
```

```
def send(channel):
    f = open("file2.txt", "w")
    data = channel.recv(1024).decode()
    print("[RECEIVING] Receiving file....")
    f.write(data)

send(channel)
```

stats.py:

```
import socket

# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)
```

Test cases:

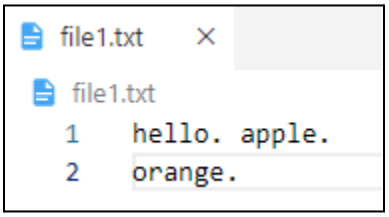
Sender side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/F
TP$ python3 sender.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:45472
[SENDING] Sending file....
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/F
TP$
```

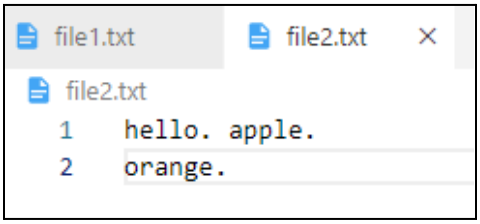
Receiver side:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/
FTP$ python3 receiver.py
[RECEIVING] Receiving file....
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/
FTP$
```

File1.txt:



File2.txt:



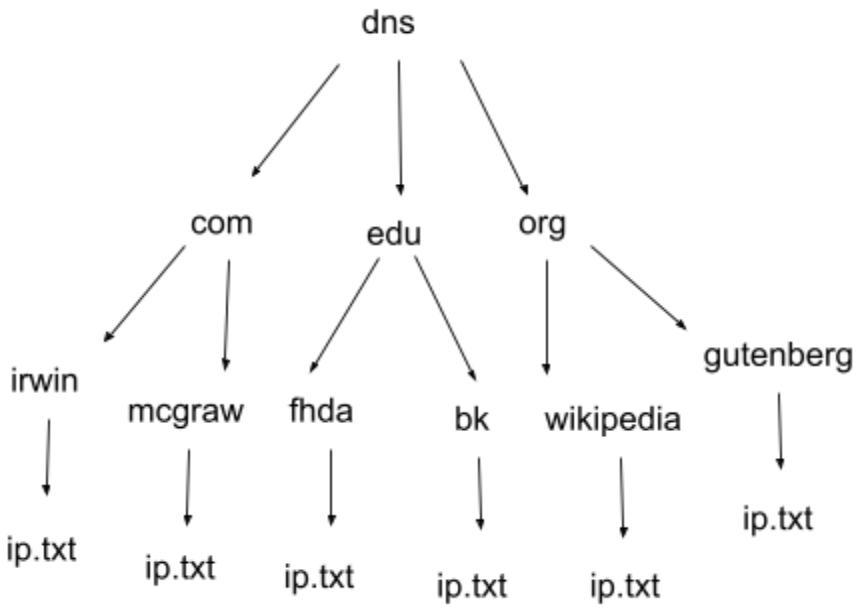
DNS:

Design:

There is a sender file “sender.py” and a client file “client.py” and a common stats.py file which holds the hostname and port number.

The domain name system is implemented via a nested folder structure.

Each terminal folder will have the corresponding ip address in a text file as shown below:



Implementation:

sender.py:

```
import socket, os, threading, stats as st

ThreadCount = 0

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.bind(st.ADDR)

print(f"[LISTENING] Channel is listening on {st.HOST_IP}")
channel.listen(5)

path = "dns/"

def recv(conn):
    global path
    packet = conn.recv(1024).decode()
    req = packet.split(".")
    req.reverse()
    for i in range(0, len(req), 1):
        path += req[i] + "/"
```

```
    path += "ip.txt"
    ip = open(path, "r").read()
    print(path)
    print(ip)
    conn.send(ip.encode())

conn, address = channel.accept()
print('[CONNECTED] Connected to: ' + address[0] + ':' + str(address[1]))
client_thread = threading.Thread(target=recv, args=(conn, ))
client_thread.start()
client_thread.join()
channel.close()
```

client.py:

```
import socket, os, threading, stats as st

ThreadCount = 0

channel = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

channel.connect(st.ADDR)

def send(channel):
    print("[INPUT] Enter domain name: ")
    dn = input()
    channel.send(dn.encode())
    ip = channel.recv(1024).decode()
    print("[IP] IP address is: ", ip)

send(channel=channel)
channel.close()
```

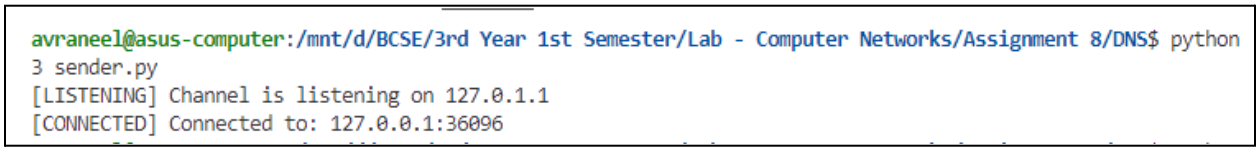
stats.py:

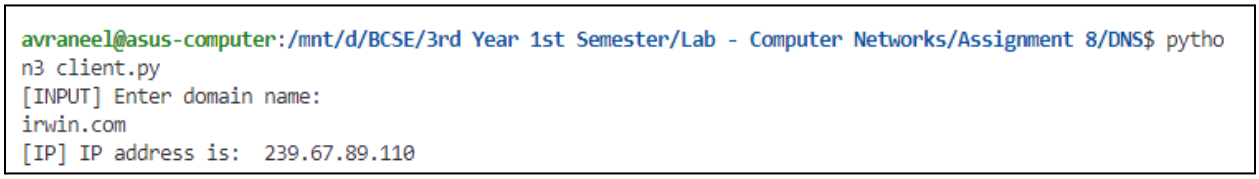
```
import socket
import time
import random
```

```
# SOCKET VARIABLES
PORT = 8081
HOST_IP = socket.gethostbyname(socket.gethostname())
ADDR = (HOST_IP, PORT)
```

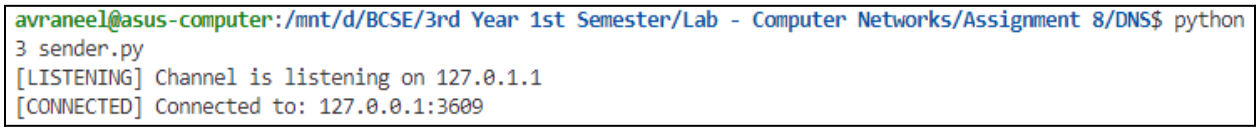
Test Cases:

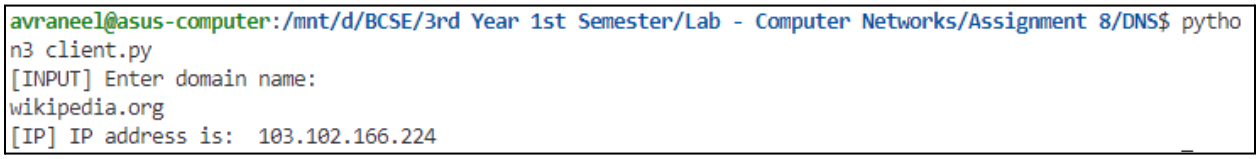
Case 1:





Case 2:





Case 3:

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/DNS$ python
3 sender.py
[LISTENING] Channel is listening on 127.0.1.1
[CONNECTED] Connected to: 127.0.0.1:36102
```

```
avraneel@asus-computer:/mnt/d/BCSE/3rd Year 1st Semester/Lab - Computer Networks/Assignment 8/DNS$ pytho
n3 client.py
[INPUT] Enter domain name:
fhda.edu
[IP] IP address is:  30.159.200.213
```

Comments:

In this assignment we learn about FTP and DNS Protocols.
