COMPUTER NETWORKS MINI PROJECT
SOCKET PROGRAMMING USING PYTHON

TEAM NUMBER - 5

TEAM MEMBERS:

- SOUMALYA BHATTACHARYA (21BIT0451)
- PARTHIB DEY (21BIT0178)
- AYANTIK CHATTERJEE (21BIT0041)
- ARVIND KUMAR (21BIT0237)

1) Project Title

Train collision detection system

2) Description

Our project aims to demonstrate a socket programming system between a server and client(s) to address the prevention of train collisions, which has become increasingly important after the recent incident in Odisha. The server is designed to be located in the station master's cabin, while the client is situated onboard the locomotive. The project follows a specific algorithm to detect collisions effectively.

Algorithm for Train Collision Detection:

1. Same Track, Opposite Direction:
    - If two trains are on the same track and moving in opposite directions, a warning message is sent to both trains, indicating a potential collision.
2. Same Track, Same Direction:
    - If two trains are on the same track and moving in the same direction, the speed of the trailing train determines the likelihood of a collision.
    - If the trailing train's speed is greater or equal, a warning message is sent to both trains, indicating a potential collision.
    - If the trailing train's speed is lower, a cautionary message is sent to the trailing train, advising it to proceed with caution.
3. Different Tracks/Lines:
    - If the trains are on different tracks or lines, an "all clear" signal is displayed, indicating that there are no imminent collision risks.

Important Note: It's crucial to emphasize that our project's implementation is not intended as a substitute for the existing collision detection system called "Kavach." Our implementation is a simplified demonstration meant to showcase the basic concepts of train collision detection using socket programming.

3) Which protocol is followed (TCP/UDP)

Our project utilizes the TCP (Transmission Control Protocol). The proof of its usage lies in the line

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

TCP stands for Transmission Control Protocol. It is one of the core protocols of the Internet Protocol Suite, commonly referred to as TCP/IP. TCP provides reliable, connection-oriented communication between devices over an IP network. It operates at the transport layer of the TCP/IP model and is responsible for breaking down data into smaller packets, ensuring their reliable delivery, and reassembling them at the receiving end.
TCP guarantees the reliable and ordered delivery of data by implementing several mechanisms. These include acknowledgment of received packets, retransmission of lost or corrupted packets, flow control to manage the rate of data transmission, and congestion control to prevent network congestion.

4) Server Code

```
import socket

def train_collision_detection(line1, speed1, direction1, line2, speed2, direction2):
    if line1 != line2:
        return 'All clear', 'green'

    if direction1 != direction2:
        return 'WARNING: COLLISION', 'red'

    if speed1 >= speed2:
        return 'Train 2 has to slow down and stop', 'yellow'

    if speed1 < speed2:
        return 'WARNING: COLLISION', 'red'

    return 'Unknown situation', 'green'

def start_server():
    host = '127.0.0.1'
    port = 5050
```

```
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen()
    print('Server listening on {}:{}'.format(host, port))

    while True:
        client_socket, addr = server_socket.accept()
        print('Connection established from:', addr)

        # Ask questions regarding line and direction
        data = client_socket.recv(1024).decode()

        if(data):
            line1, direction1, line2, direction2, speed1, speed2 = data.split(',')
            print(data)
        speed1 = int(speed1)
        speed2 = int(speed2)

        result, color = train_collision_detection(line1, speed1, direction1, line2, speed2, direction2)

        response = f'{result},{color}'
        client_socket.send(response.encode())

        client_socket.close()
        print("Client from ",addr," disconnected")

    server_socket.close()

start_server()
```

5) Client Code

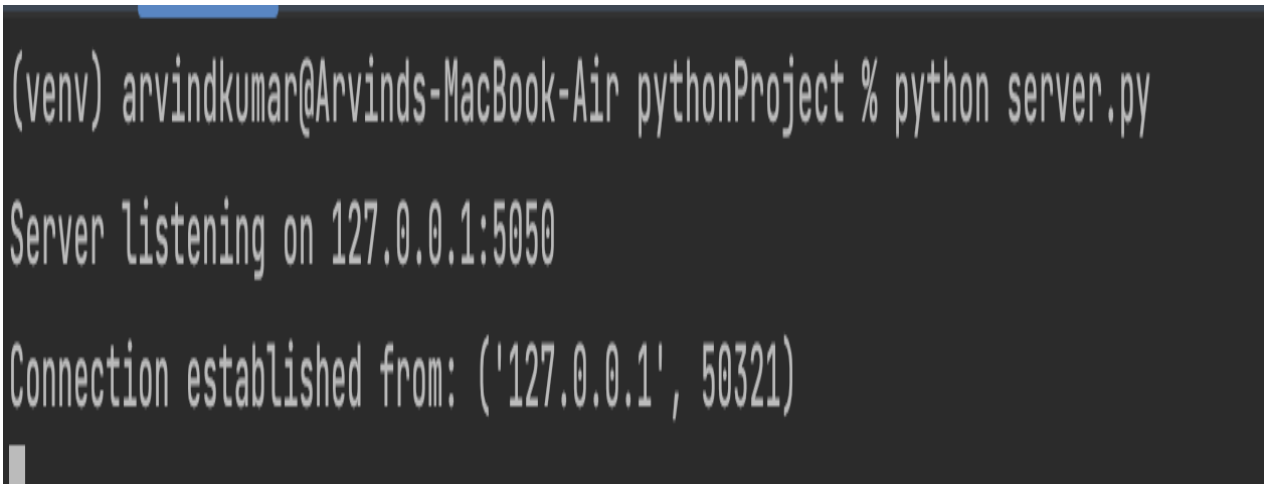
```
import tkinter
from tkinter import *
import socket

host = '192.168.80.133'
port = 5050

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((host, port))
```

```
screen = Tk()
screen.title("Train Data")
screen.geometry("840x510")
screen.resizable(False, False)

#heading
Label(screen, text="TRAIN 1", font=("sanskrit text", 20, "bold"), bg="blue", fg="white").place(x=0,
y=0)
Label(screen, text="TRAIN 2", font=("sanskrit text", 20, "bold"), bg="red",
fg="white").place(x=420, y=0)

#labels
Label(screen, text="Line1:", font=("roman",20)).place(x=40, y=60)
Label(screen, text="Direction1:", font=("roman",20)).place(x=40, y=110)
Label(screen, text="Speed1:", font=("roman",20)).place(x=40, y=170)

Label(screen, text="Line2:", font=("roman",20)).place(x=460, y=60)
Label(screen, text="Direction2:", font=("roman",20)).place(x=460, y=110)
Label(screen, text="Speed2:", font=("roman",20)).place(x=460, y=170)

#radio
ln1=StringVar()
ln1.set(None)
Radiobutton(screen, text='up', font=('small fonts', 20), variable=ln1, value='up').place(x=180,
y=60)
Radiobutton(screen, text='down', font=('small fonts', 20), variable=ln1,
value='down').place(x=280, y=60)

dir1=StringVar()
dir1.set(None)
Radiobutton(screen, text='left', font=('small fonts', 20), variable=dir1, value='left').place(x=180,
y=110)
Radiobutton(screen, text='right', font=('small fonts', 20), variable=dir1,
value='right').place(x=280, y=110)

ln2=StringVar()
ln2.set(None)
Radiobutton(screen, text='up', font=('small fonts', 20), variable=ln2, value='up').place(x=600,
y=60)
Radiobutton(screen, text='down', font=('small fonts', 20), variable=ln2,
value='down').place(x=700, y=60)

dir2=StringVar()
dir2.set(None)
```

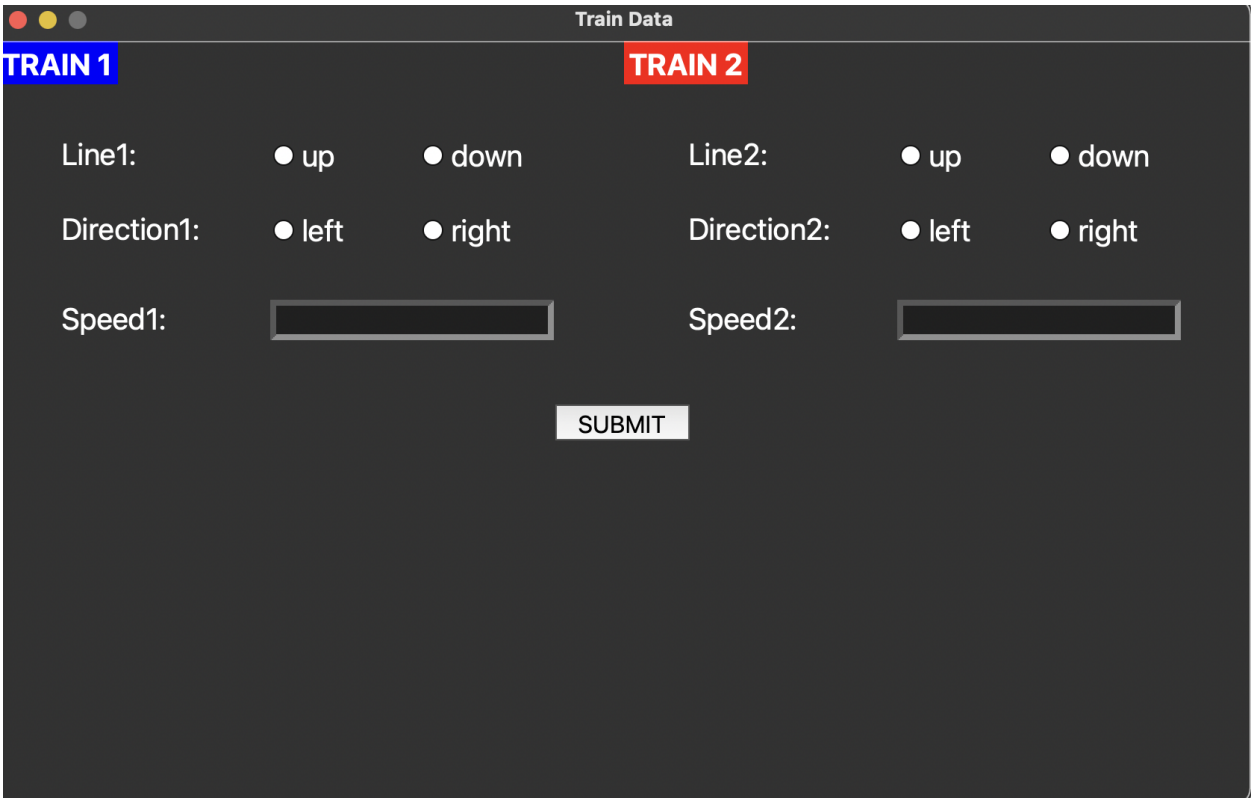
```
Radiobutton(screen, text='left', font=('small fonts', 20), variable=dir2, value='left').place(x=600,
y=110)
Radiobutton(screen, text='right', font=('small fonts', 20), variable=dir2,
value='right').place(x=700, y=110)

#text entry
spd1=Entry(screen, font=('small fonts', 14), bd=4)
spd1.place(x=180, y=170)
spd2=Entry(screen, font=('small fonts', 14), bd=4)
spd2.place(x=600, y=170)

data = ''

def process():
    line1=ln1.get()
    direction1=dir1.get()
    line2=ln2.get()
    direction2=dir2.get()
    speed1=spd1.get()
    speed2=spd2.get()

    data = f'{line1},{direction1},{line2},{direction2},{speed1},{speed2}'
    client_socket.send(data.encode())

    response = client_socket.recv(1024).decode()
    if(response):
        result, color = response.split(',')

        signal = tkinter.Canvas(screen, height=100, width=100)
        signal.place(x=370, y=310)
        signal.create_oval((10,10,90,90),fill=color)

        Label(screen, text=result, font=("impact", 40, "bold"), bg="black",
fg="white").pack(side='bottom', fill='both')

#button
Button(screen, text='SUBMIT', font=("small fonts",16), command=process).place(x=371, y=240)

screen.mainloop()

client_socket.close()
```

6) Screenshots

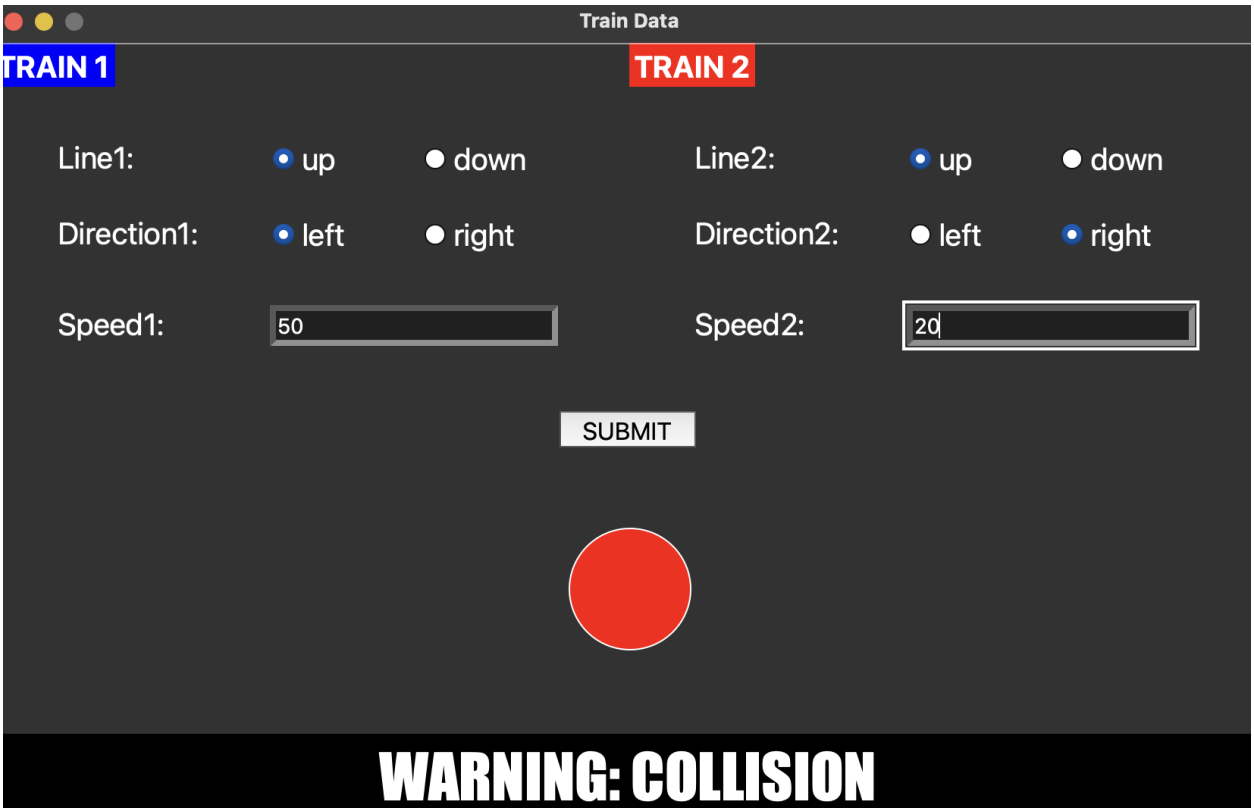
Server-side



CLIENT-SIDE GUI INITIALLY

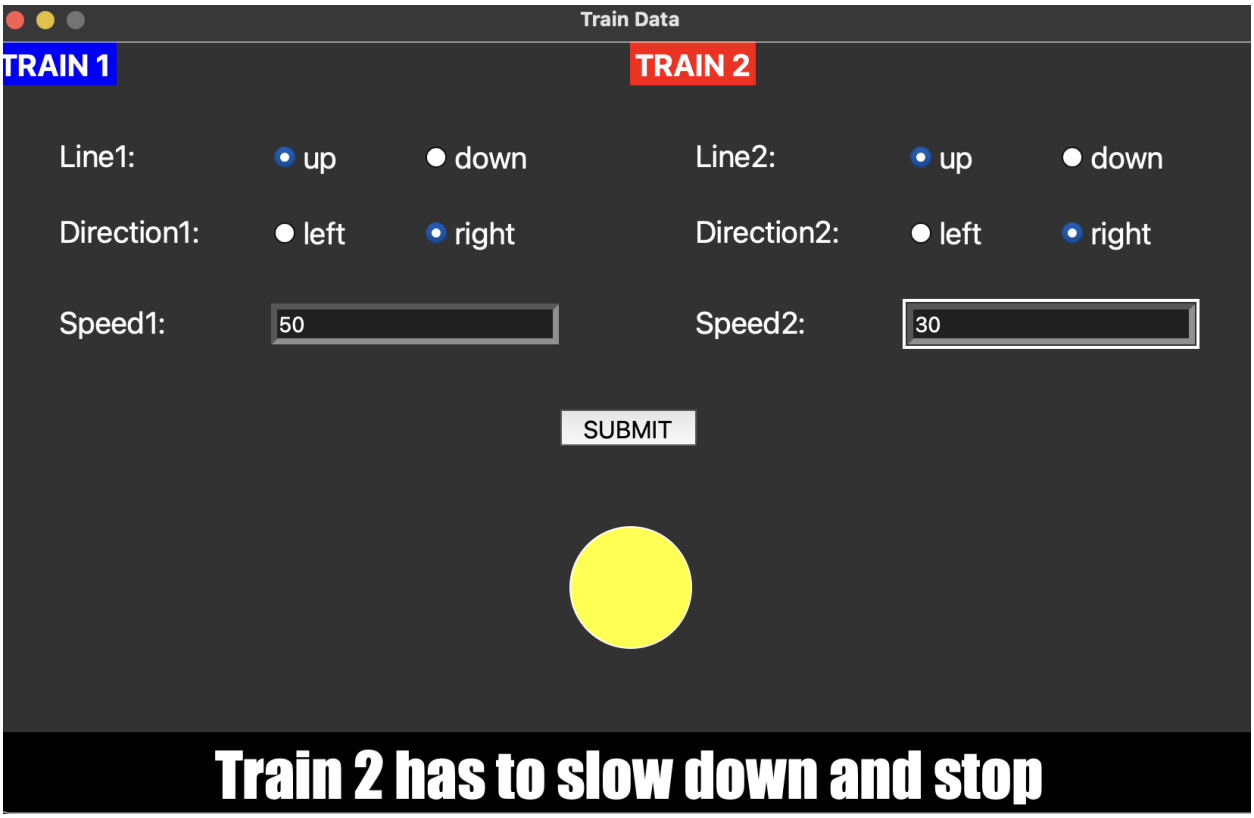


COLLISION WARNING

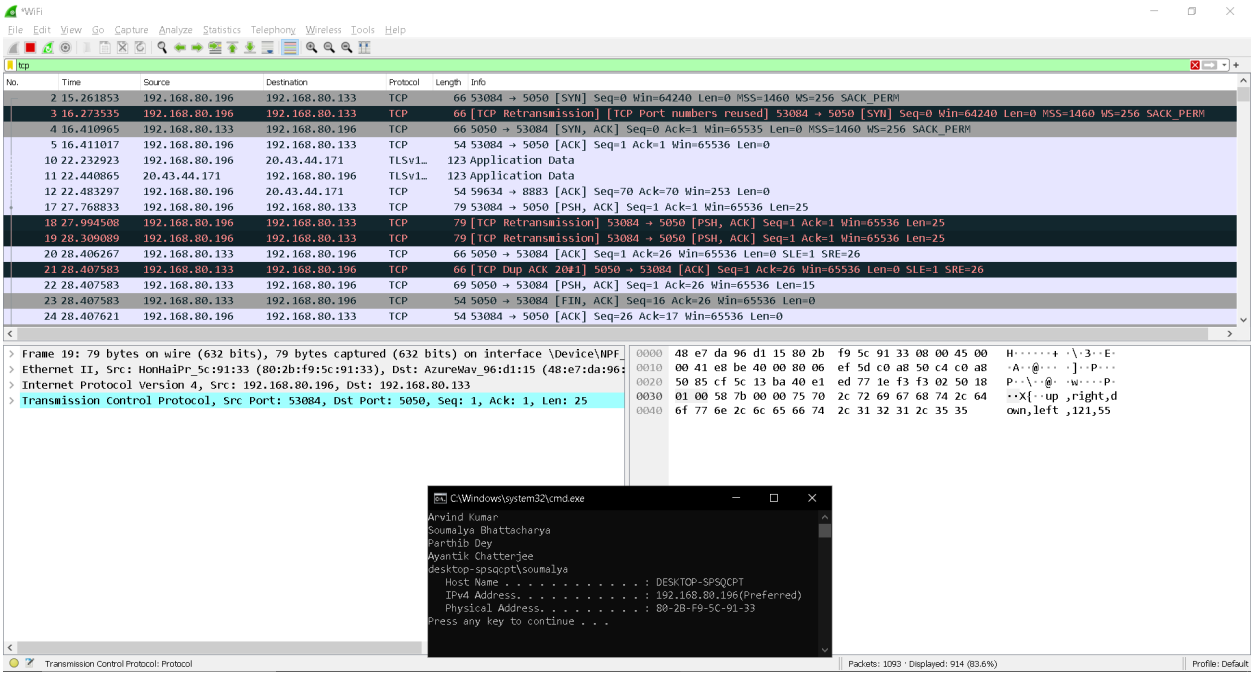


ALL CLEAR SIGNAL

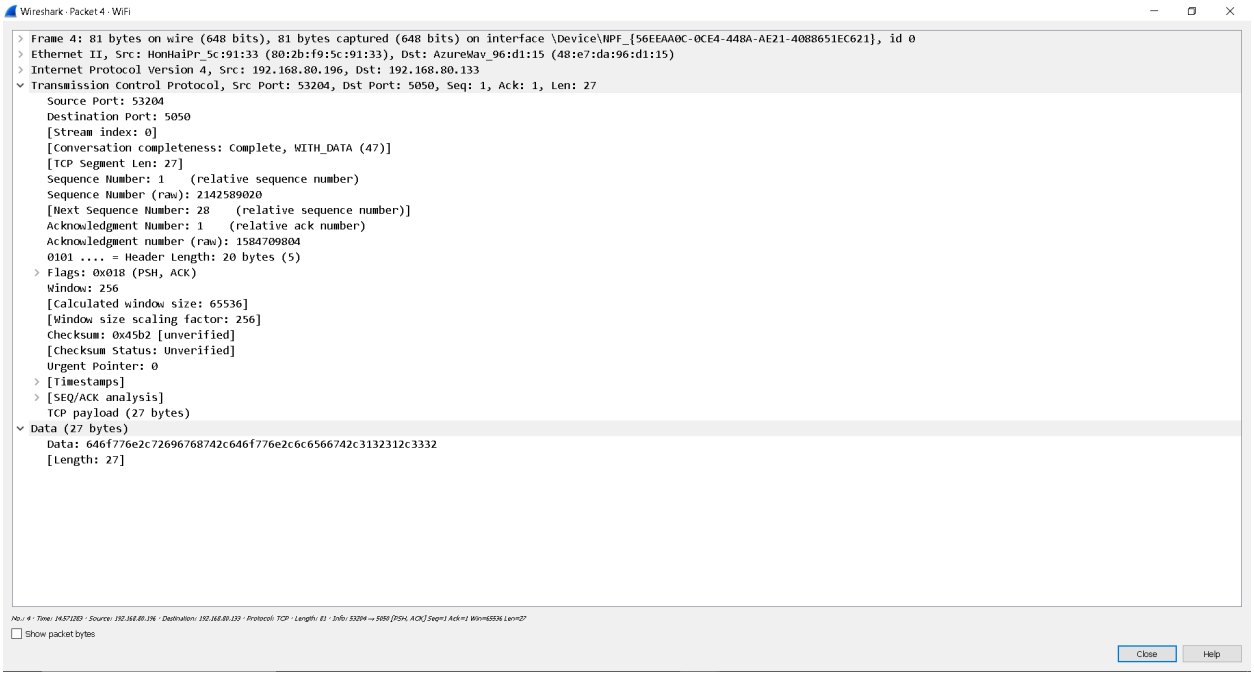
WARNING FOR SLOW DOWN



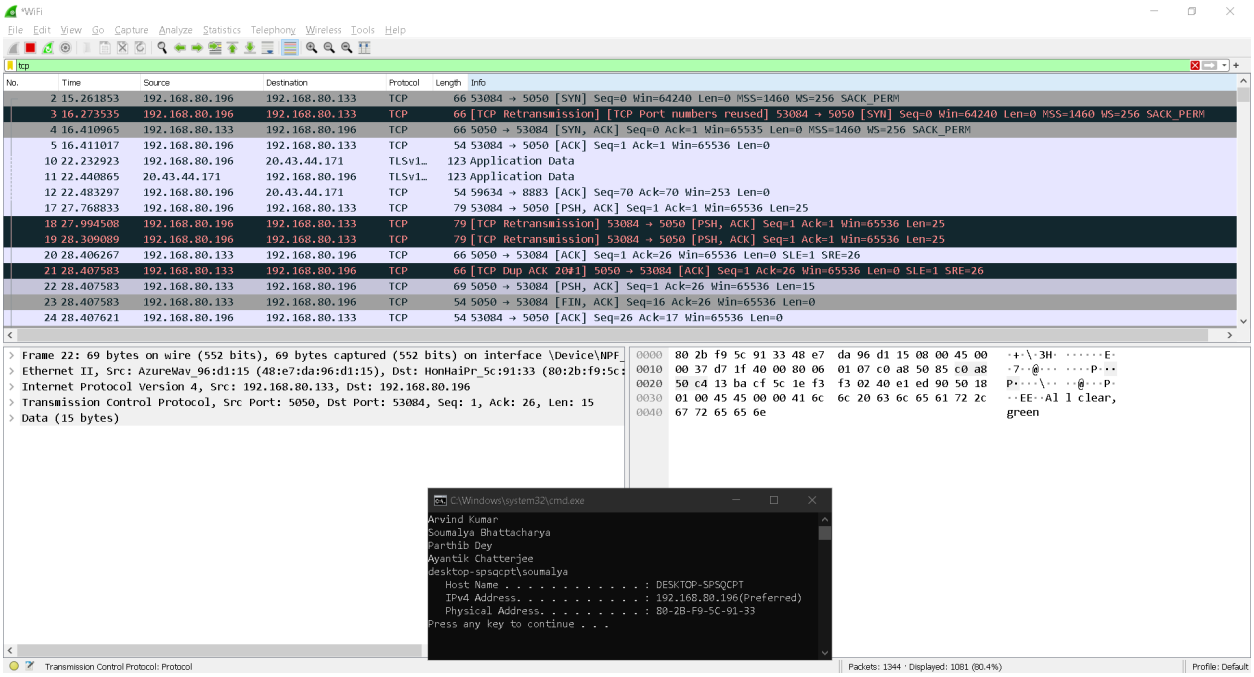
7) Wireshark Verification



TCP PACKET FROM CLIENT TO SERVER



TCP PACKET FROM SERVER TO CLIENT



THEY ALSO POP UP WHEN CLIENT SIDE CODE IS RUN

8)Advertisement

Introducing the Future of Rail Safety: Our Client-Server-Based Train Collision Detection System

In the wake of recent incidents, such as the tragic Orissa train accident, it has become evident that prioritizing rail safety is not just a choice but an absolute necessity. We understand the gravity of the situation, and that's why we bring you the ultimate solution: our state-of-the-art Client-Server-Based Train Collision Detection System.

Learn from the past, act in the present, and safeguard the future. We proudly present our Client-Server-Based Train Collision Detection System, revolutionizing rail safety like never before. With our technology, we prioritize the well-being of passengers and crew while ensuring smooth, uninterrupted train operations.

Key Features and Benefits:

1. Real-time Alerts and Emergency Response: Time is of the essence when it comes to preventing accidents. Our system instantly alerts train operators, control centers, and onboard personnel when it detects a possible collision threat. Rapid response protocols can be activated immediately, ensuring the safety of passengers, crew, and valuable assets.
2. Seamless Integration, Enhanced Efficiency: Our client-server architecture seamlessly integrates into your existing infrastructure, ensuring a smooth transition and minimal disruptions (due to usage of Python). Experience streamlined data sharing, real-time updates, and centralized monitoring capabilities, empowering you with comprehensive control over your rail network's safety.

Do not wait for the next tragedy to strike. Act now, and secure the future of rail safety with our *Client-Server-Based Train Collision Detection System*. Join the growing number of forward-thinking railway operators who prioritize the well-being of their passengers and crew.

Remember, safety is not an expense—it's an investment. Together, let's build a safer and more reliable railway system for generations to come.

9) Team Contribution

ARVIND KUMAR - PROJECT IDEATION AND PROGRAMMING

SOUMALYA BHATTACHARYA - PROJECT IDEATION AND PROGRAMMING

AYANTIK CHATTERJEE - PROJECT IDEATION AND DOCUMENTATION

PARTHIB DEY - PROJECT IDEATION AND DOCUMENTATION

10) References

1. SOCKET PROGRAMMING - CLIENT AND SERVER SIDE BY DR. SHANMUGA PERUMAL

2. Tech With Tim. (2023, March 16). Python Socket Programming Tutorial. [Video]. YouTube. https://www.youtube.com/watch?v=3QiPPX-KeSc

3. OpenAI. (2021). ChatGPT [Computer software]. Retrieved June 3, 2023, from https://openai.com/research/chatgpt

4. Tutorialspoint Python tkinter programming https://www.tutorialspoint.com/python/python_gui_programming.htm