TCP CONNECTION ANALYSER THREE WAY HANDSHAKE

PROJECT REPORT

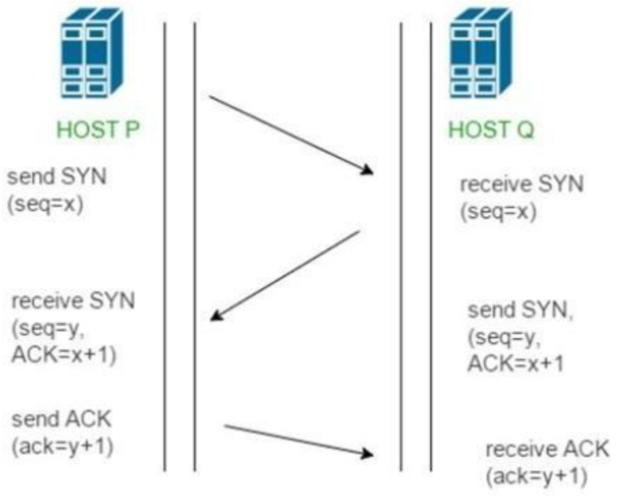
SUBMITTED BY

AKSHAYA SRIKRISHNA
2022103065
ANAGHA SRIKRISHNA
2022103066
KRISHNENDU M R
2022103081

TABLE OF CONTENTS

INTRODUCTION

The TCP 3-Way Handshake is the foundational process that enables a reliable connection between a client and a server before actual data transmission occurs. This handshake sequence ensures both parties are synchronized and ready for secure communication, making it a crucial step in TCP/IP networking. When monitoring network connections, analyzing the TCP 3-Way Handshake can help us verify connection integrity and troubleshoot issues.
We can use Wireshark to capture and inspect the TCP 3-Way Handshake process:

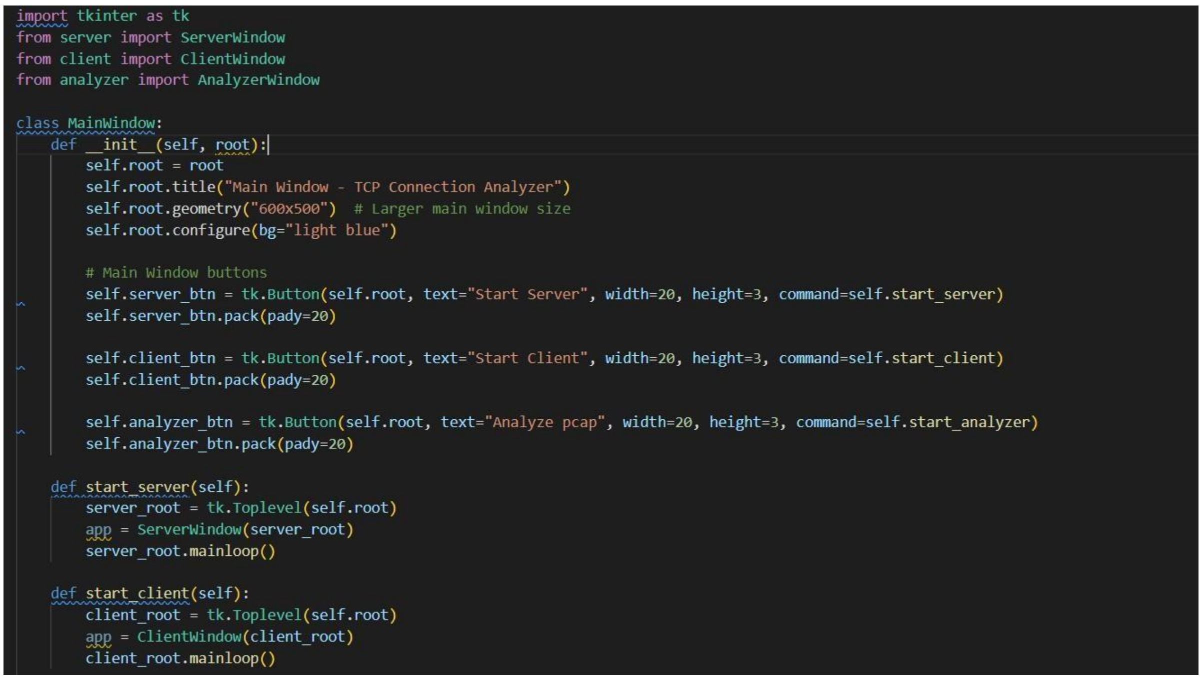


1. SYN (Synchronize): The client initiates a connection request by sending a segment with the SYN (Synchronize Sequence Number) flag, indicating that it wants to start communication and specifies the starting sequence number.
2. SYN-ACK (Synchronize + Acknowledgment): The server responds to the client's SYN request with a SYN-ACK segment. The ACK acknowledges the client's SYN, and the SYN indicates the server's starting sequence number.
3. ACK (Acknowledgment): The client sends an ACK in response, completing the handshake and establishing a reliable connection. Both parties are now ready to start transmitting data.

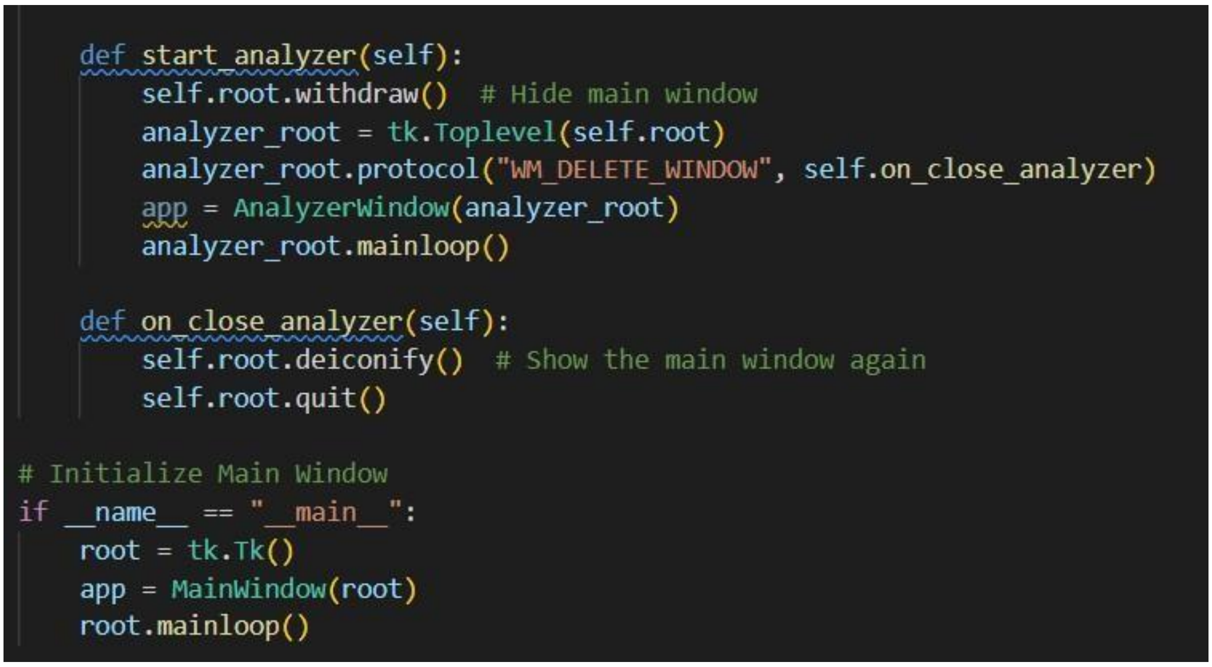
Using Wireshark, we'll capture these handshake packets, allowing us to validate the correct establishment of TCP connections. This tool is invaluable for network engineers and security analysts, as it helps ensure connection reliability and identify potential connection issues in real time.

PROGRAMS IN PYTHON

Main.py

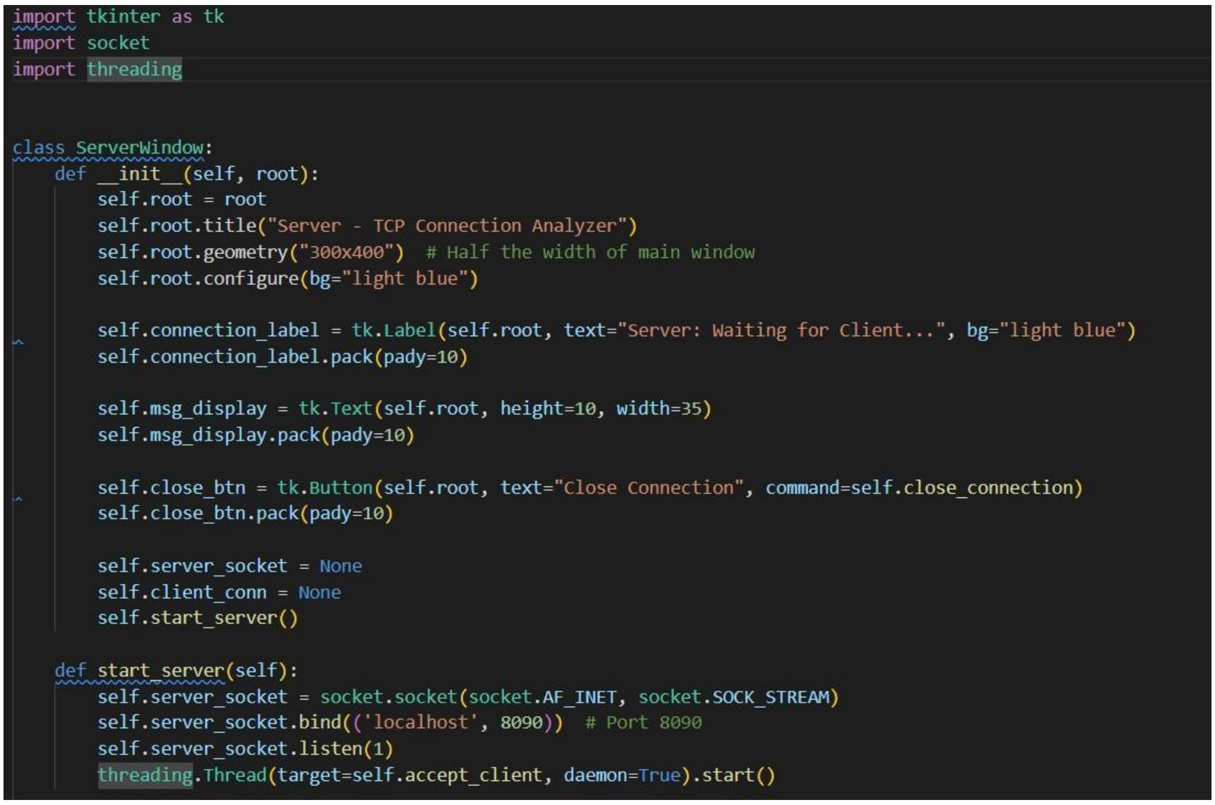


```python
import tkinter as tk
from server import ServerWindow
from client import ClientWindow
from analyzer import AnalyzerWindow

class MainWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("Main Window - TCP Connection Analyzer")
        self.root.geometry("600x500")  # Larger main window size
        self.root.configure(bg="light blue")

        # Main Window buttons
        self.server_btn = tk.Button(self.root, text="Start Server", width=20, height=3, command=self.start_server)
        self.server_btn.pack(pady=20)

        self.client_btn = tk.Button(self.root, text="Start Client", width=20, height=3, command=self.start_client)
        self.client_btn.pack(pady=20)

        self.analyzer_btn = tk.Button(self.root, text="Analyze pcap", width=20, height=3, command=self.start_analyzer)
        self.analyzer_btn.pack(pady=20)

    def start_server(self):
        server_root = tk.Toplevel(self.root)
        app = ServerWindow(server_root)
        server_root.mainloop()

    def start_client(self):
        client_root = tk.Toplevel(self.root)
        app = ClientWindow(client_root)
        client_root.mainloop()
```

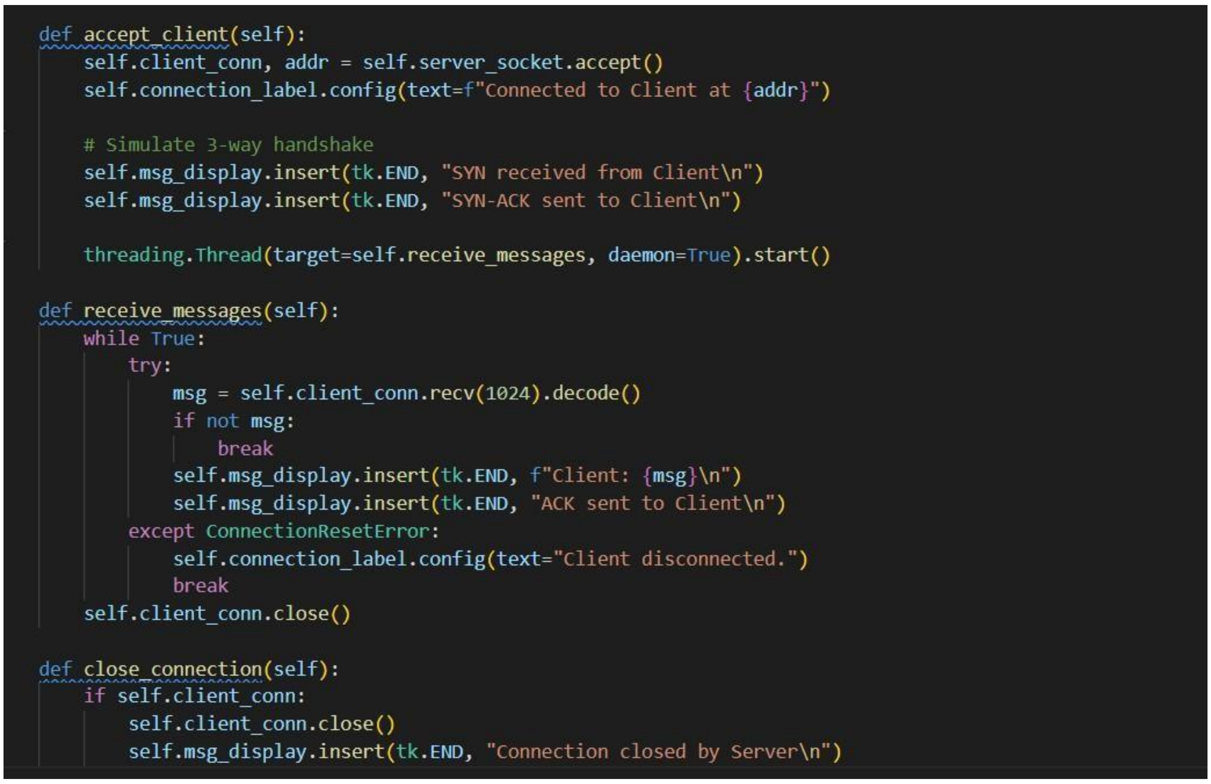


```python
    def start_analyzer(self):
        self.root.withdraw()  # Hide main window
        analyzer_root = tk.Toplevel(self.root)
        analyzer_root.protocol("WM_DELETE_WINDOW", self.on_close_analyzer)
        app = AnalyzerWindow(analyzer_root)
        analyzer_root.mainloop()

    def on_close_analyzer(self):
        self.root.deiconify()  # Show the main window again
        self.root.quit()

# Initialize Main Window
if __name__ == "__main__":
    root = tk.Tk()
    app = MainWindow(root)
    root.mainloop()
```

server.py

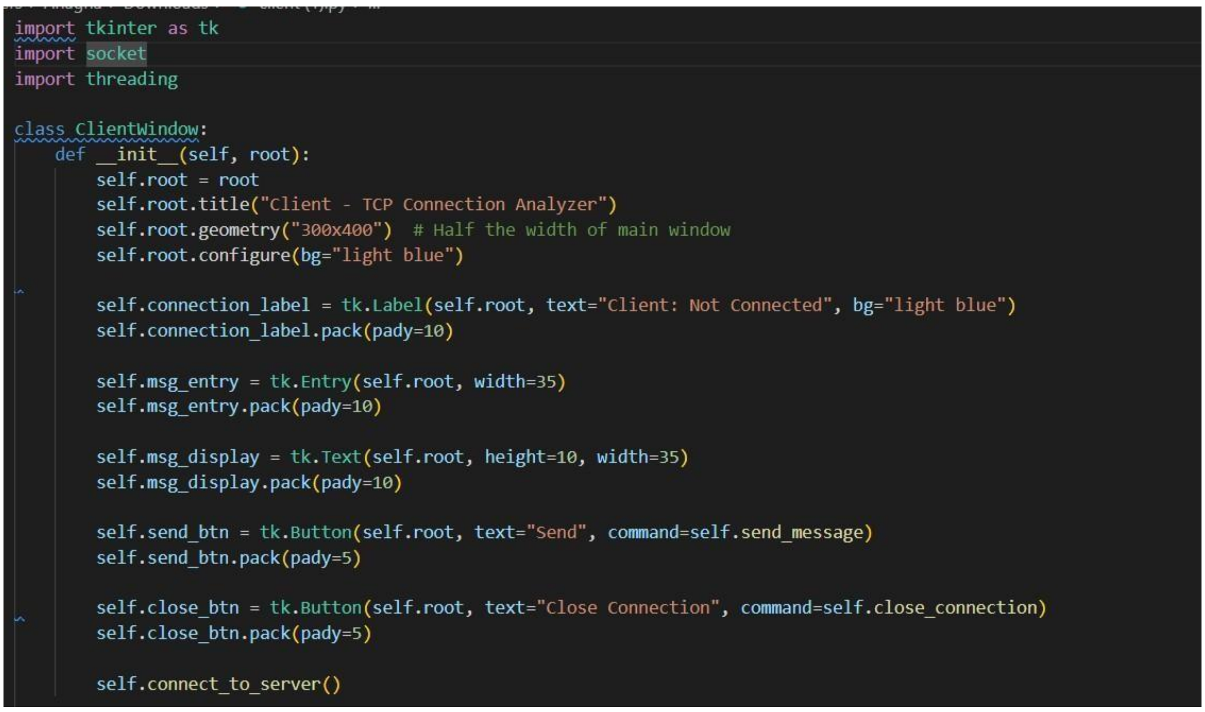


```python
import tkinter as tk
import socket
import threading


class ServerWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("Server - TCP Connection Analyzer")
        self.root.geometry("300x400")  # Half the width of main window
        self.root.configure(bg="light blue")

        self.connection_label = tk.Label(self.root, text="Server: Waiting for Client...", bg="light blue")
        self.connection_label.pack(pady=10)

        self.msg_display = tk.Text(self.root, height=10, width=35)
        self.msg_display.pack(pady=10)

        self.close_btn = tk.Button(self.root, text="Close Connection", command=self.close_connection)
        self.close_btn.pack(pady=10)

        self.server_socket = None
        self.client_conn = None
        self.start_server()

    def start_server(self):
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_socket.bind(('localhost', 8090))  # Port 8090
        self.server_socket.listen(1)
        threading.Thread(target=self.accept_client, daemon=True).start()
```

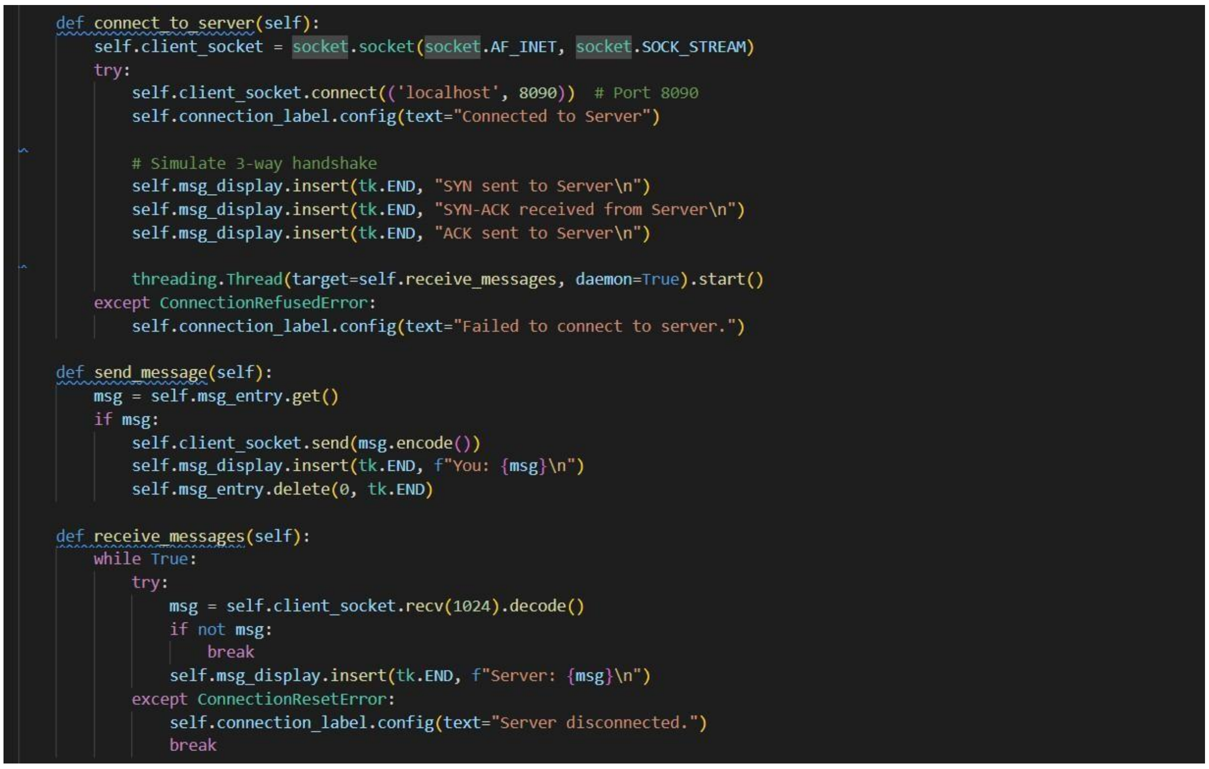


```python
    def accept_client(self):
        self.client_conn, addr = self.server_socket.accept()
        self.connection_label.config(text=f"Connected to Client at {addr}")

        # Simulate 3-way handshake
        self.msg_display.insert(tk.END, "SYN received from Client\n")
        self.msg_display.insert(tk.END, "SYN-ACK sent to Client\n")

        threading.Thread(target=self.receive_messages, daemon=True).start()

    def receive_messages(self):
        while True:
            try:
                msg = self.client_conn.recv(1024).decode()
                if not msg:
                    break
                self.msg_display.insert(tk.END, f"Client: {msg}\n")
                self.msg_display.insert(tk.END, "ACK sent to Client\n")
            except ConnectionResetError:
                self.connection_label.config(text="Client disconnected.")
                break
        self.client_conn.close()

    def close_connection(self):
        if self.client_conn:
            self.client_conn.close()
            self.msg_display.insert(tk.END, "Connection closed by Server\n")
```

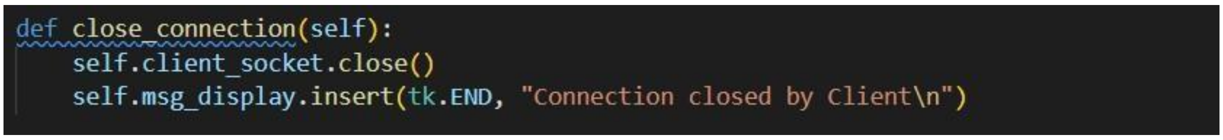
client.py



```
import tkinter as tk
import socket
import threading

class ClientWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("Client - TCP Connection Analyzer")
        self.root.geometry("300x400")  # Half the width of main window
        self.root.configure(bg="light blue")

        self.connection_label = tk.Label(self.root, text="Client: Not Connected", bg="light blue")
        self.connection_label.pack(pady=10)

        self.msg_entry = tk.Entry(self.root, width=35)
        self.msg_entry.pack(pady=10)

        self.msg_display = tk.Text(self.root, height=10, width=35)
        self.msg_display.pack(pady=10)

        self.send_btn = tk.Button(self.root, text="Send", command=self.send_message)
        self.send_btn.pack(pady=5)

        self.close_btn = tk.Button(self.root, text="Close Connection", command=self.close_connection)
        self.close_btn.pack(pady=5)

        self.connect_to_server()
```

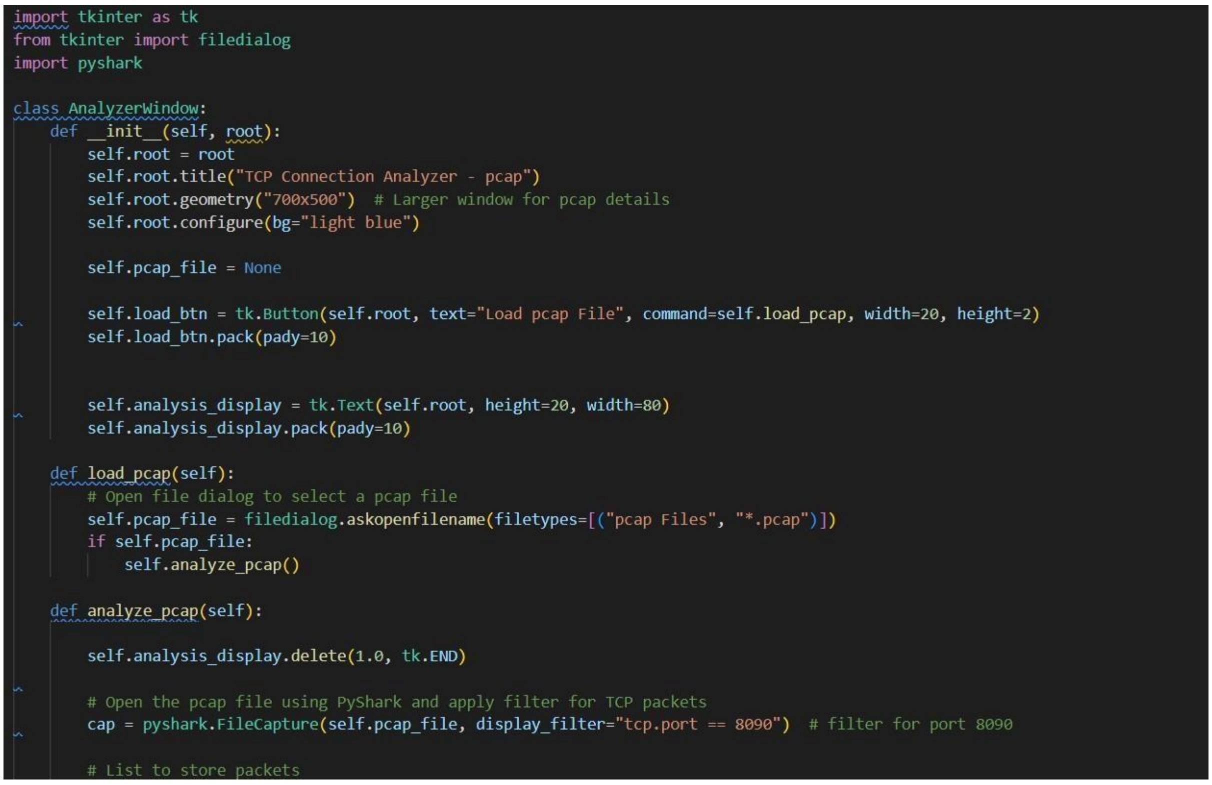


```
    def connect_to_server(self):
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.client_socket.connect(('localhost', 8090))  # Port 8090
            self.connection_label.config(text="Connected to Server")

            # Simulate 3-way handshake
            self.msg_display.insert(tk.END, "SYN sent to Server\n")
            self.msg_display.insert(tk.END, "SYN-ACK received from Server\n")
            self.msg_display.insert(tk.END, "ACK sent to Server\n")

            threading.Thread(target=self.receive_messages, daemon=True).start()
        except ConnectionRefusedError:
            self.connection_label.config(text="Failed to connect to server.")

    def send_message(self):
        msg = self.msg_entry.get()
        if msg:
            self.client_socket.send(msg.encode())
            self.msg_display.insert(tk.END, f"You: {msg}\n")
            self.msg_entry.delete(0, tk.END)

    def receive_messages(self):
        while True:
            try:
                msg = self.client_socket.recv(1024).decode()
                if not msg:
                    break
                self.msg_display.insert(tk.END, f"Server: {msg}\n")
            except ConnectionResetError:
                self.connection_label.config(text="Server disconnected.")
                break
```

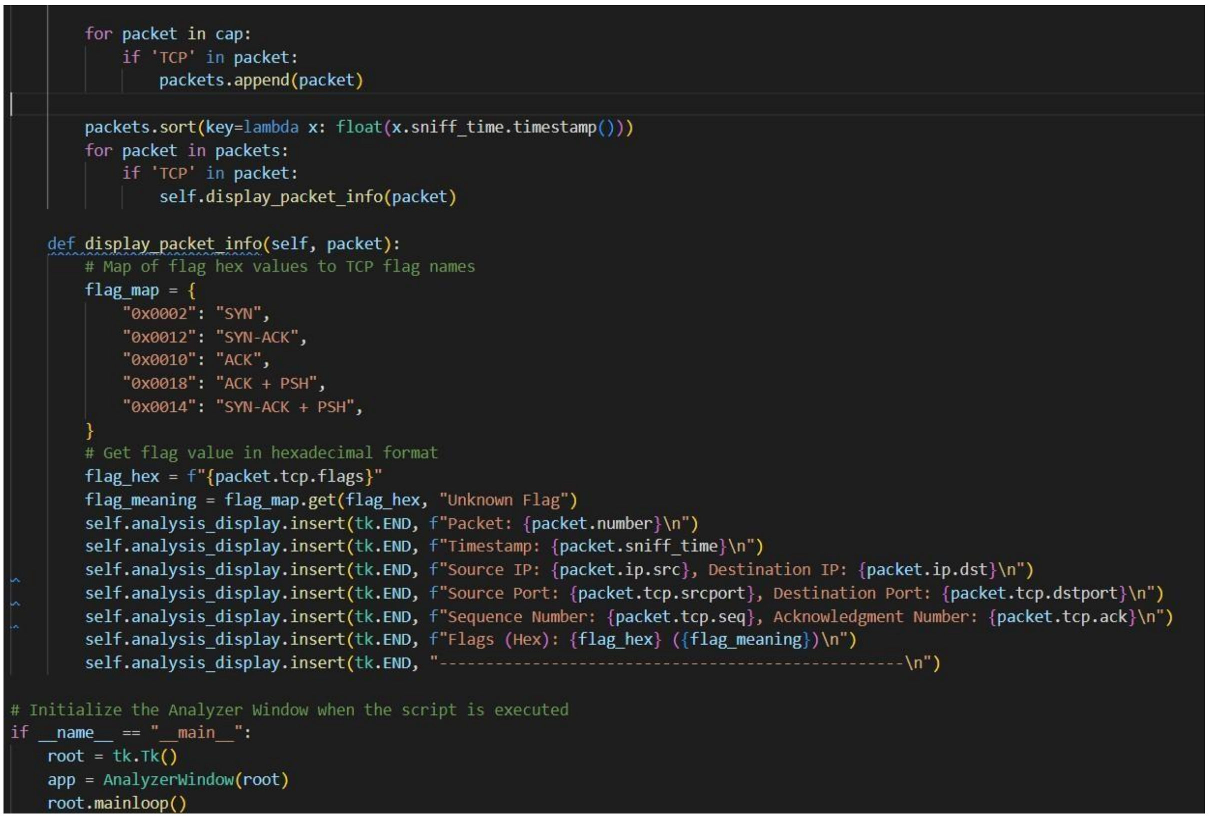


```
def close_connection(self):
    self.client_socket.close()
    self.msg_display.insert(tk.END, "Connection closed by Client\n")
```

analyzer.py



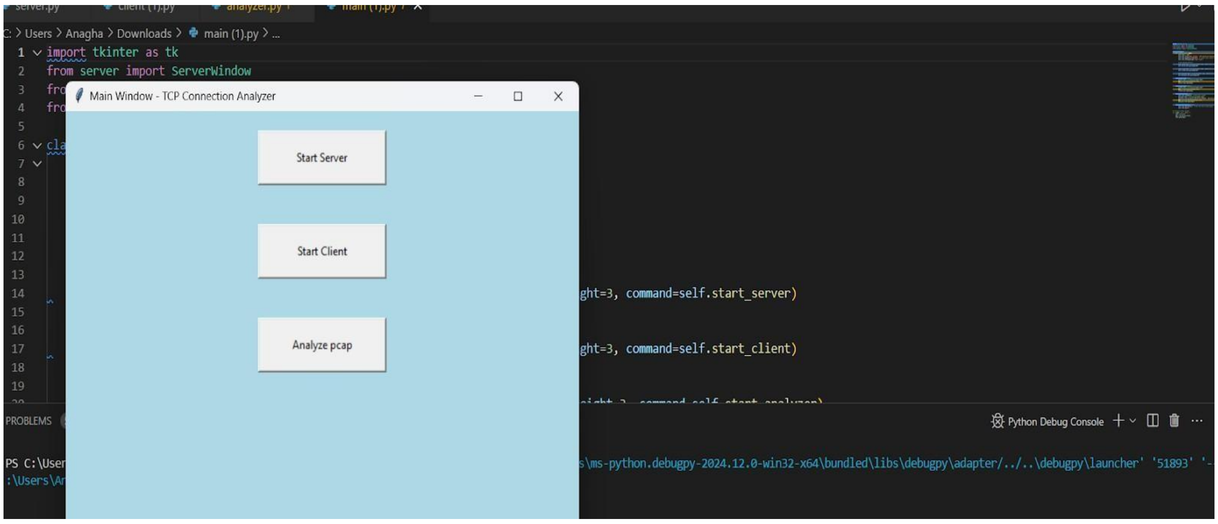
```python
import tkinter as tk
from tkinter import filedialog
import pyshark

class AnalyzerWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("TCP Connection Analyzer - pcap")
        self.root.geometry("700x500")  # Larger window for pcap details
        self.root.configure(bg="light blue")

        self.pcap_file = None

        self.load_btn = tk.Button(self.root, text="Load pcap File", command=self.load_pcap, width=20, height=2)
        self.load_btn.pack(pady=10)


        self.analysis_display = tk.Text(self.root, height=20, width=80)
        self.analysis_display.pack(pady=10)

    def load_pcap(self):
        # Open file dialog to select a pcap file
        self.pcap_file = filedialog.askopenfilename(filetypes=[("pcap Files", "*.pcap")])
        if self.pcap_file:
            self.analyze_pcap()

    def analyze_pcap(self):

        self.analysis_display.delete(1.0, tk.END)

        # Open the pcap file using PyShark and apply filter for TCP packets
        cap = pyshark.FileCapture(self.pcap_file, display_filter="tcp.port == 8090")  # filter for port 8090

        # List to store packets
```



```python
        for packet in cap:
            if 'TCP' in packet:
                packets.append(packet)

        packets.sort(key=lambda x: float(x.sniff_time.timestamp()))
        for packet in packets:
            if 'TCP' in packet:
                self.display_packet_info(packet)

    def display_packet_info(self, packet):
        # Map of flag hex values to TCP flag names
        flag_map = {
            "0x0002": "SYN",
            "0x0012": "SYN-ACK",
            "0x0010": "ACK",
            "0x0018": "ACK + PSH",
            "0x0014": "SYN-ACK + PSH",
        }
        # Get flag value in hexadecimal format
        flag_hex = f"{packet.tcp.flags}"
        flag_meaning = flag_map.get(flag_hex, "Unknown Flag")
        self.analysis_display.insert(tk.END, f"Packet: {packet.number}\n")
        self.analysis_display.insert(tk.END, f"Timestamp: {packet.sniff_time}\n")
        self.analysis_display.insert(tk.END, f"Source IP: {packet.ip.src}, Destination IP: {packet.ip.dst}\n")
        self.analysis_display.insert(tk.END, f"Source Port: {packet.tcp.srcport}, Destination Port: {packet.tcp.dstport}\n")
        self.analysis_display.insert(tk.END, f"Sequence Number: {packet.tcp.seq}, Acknowledgment Number: {packet.tcp.ack}\n")
        self.analysis_display.insert(tk.END, f"Flags (Hex): {flag_hex} ({flag_meaning})\n")
        self.analysis_display.insert(tk.END, "--------------------------------------------------\n")

# Initialize the Analyzer Window when the script is executed
if __name__ == "__main__":
    root = tk.Tk()
    app = AnalyzerWindow(root)
    root.mainloop()
```

OUTPUT

1) Run main.py

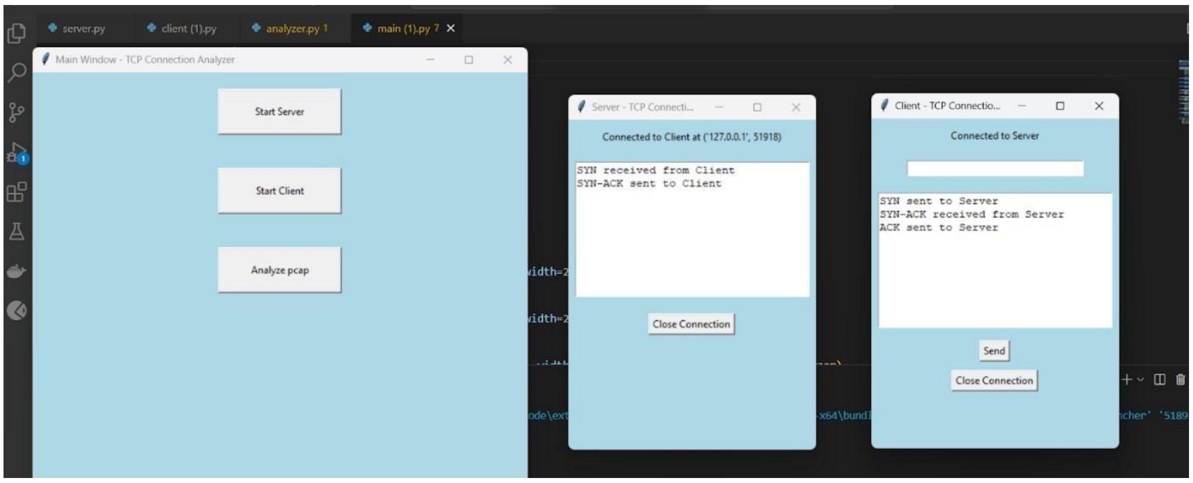

C: > Users > Anagha > Downloads > main (1).py > ...
Main Window - TCP Connection Analyzer
Start Server
Start Client
Analyze pcap
PROBLEMS
Python Debug Console

```
import tkinter as tk
from server import ServerWindow
fro
fro

cla
```

```
ght=3, command=self.start_server)
ght=3, command=self.start_client)
```

```
PS C:\User
:\Users\Ar
s\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '51893' '-
```

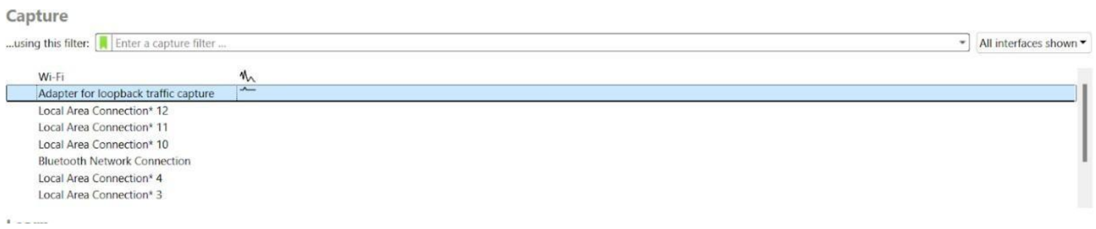
2) Start the server followed by the client



```
Connected to Client at ('127.0.0.1', 51918)

SYN received from Client
SYN-ACK sent to Client
```

```
Connected to Server

SYN sent to Server
SYN-ACK received from Server
ACK sent to Server
```

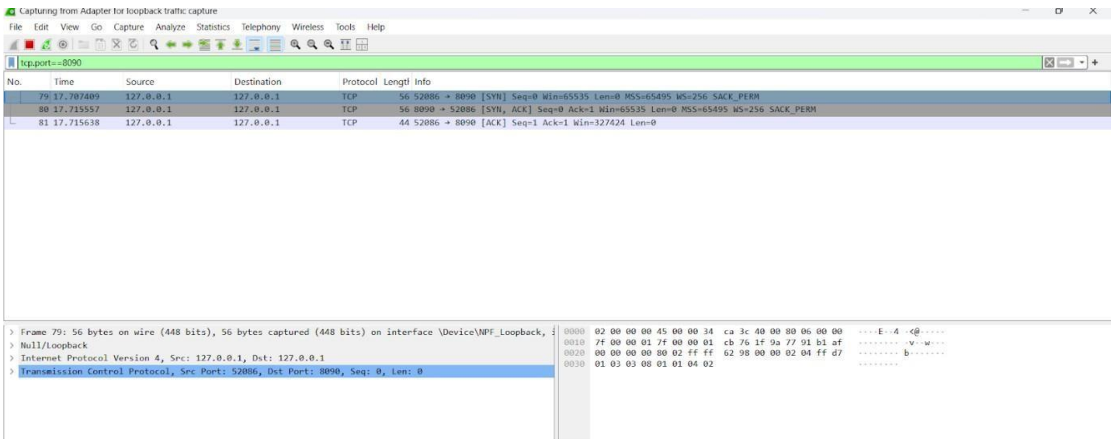
3) Select Adopter for loopback traffic capture


Capture

...using this filter: Enter a capture filter ... All interfaces shown

```
Wi-Fi
Adapter for loopback traffic capture
Local Area Connection* 12
Local Area Connection* 11
Local Area Connection* 10
Bluetooth Network Connection
Local Area Connection* 4
Local Area Connection* 3
```

4)Enter tcp.port==8090


Capturing from Adapter for loopback traffic capture

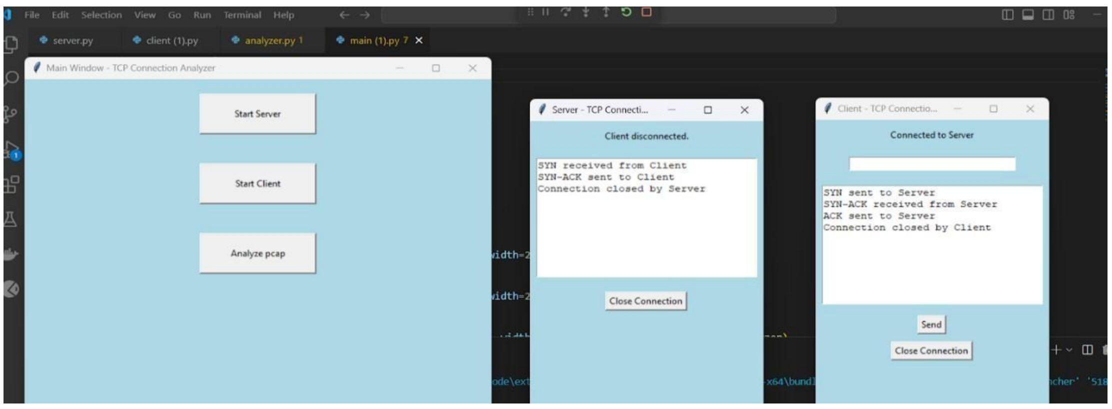
tcp.port==8090

```
No.   Time       Source     Destination  Protocol Length Info
   79 17.707409  127.0.0.1  127.0.0.1    TCP      56 52086 → 8090 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   80 17.715557  127.0.0.1  127.0.0.1    TCP      56 8090 → 52086 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
   81 17.715638  127.0.0.1  127.0.0.1    TCP      44 52086 → 8090 [ACK] Seq=1 Ack=1 Win=327424 Len=0
```

```
> Frame 79: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface \Device\NPF_Loopback, i
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 52086, Dst Port: 8090, Seq: 0, Len: 0
```

```
0000  02 00 00 00 45 00 00 34  ca 3c 40 00 80 06 00 00   ····E··4 ·<@·····
0010  7f 00 00 01 7f 00 00 01  cb 76 1f 9a 77 91 b1 af   ········ ·v··w···
0020  00 00 00 00 80 02 ff ff  62 98 00 00 02 04 ff d7   ········ b·······
0030  01 03 03 08 01 01 04 02                            ········
```

5)Close connections



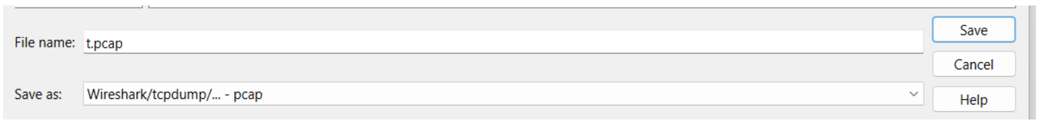
Client disconnected.

```
SYN received from Client
SYN-ACK sent to Client
Connection closed by Server
```

Connected to Server

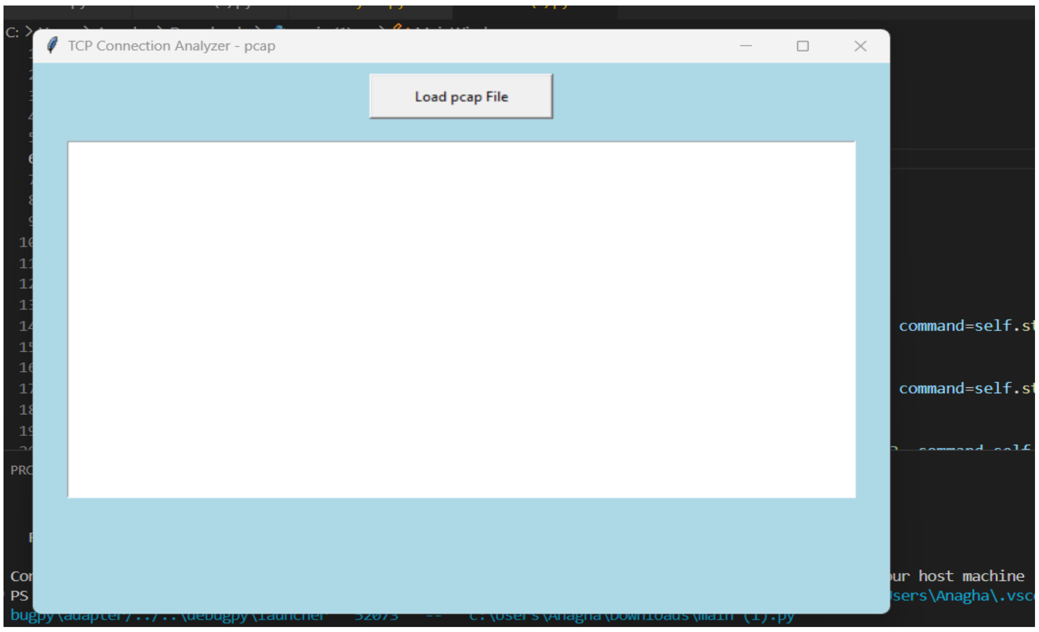
```
SYN sent to Server
SYN-ACK received from Server
ACK sent to Server
Connection closed by Client
```

1) Save wireshark

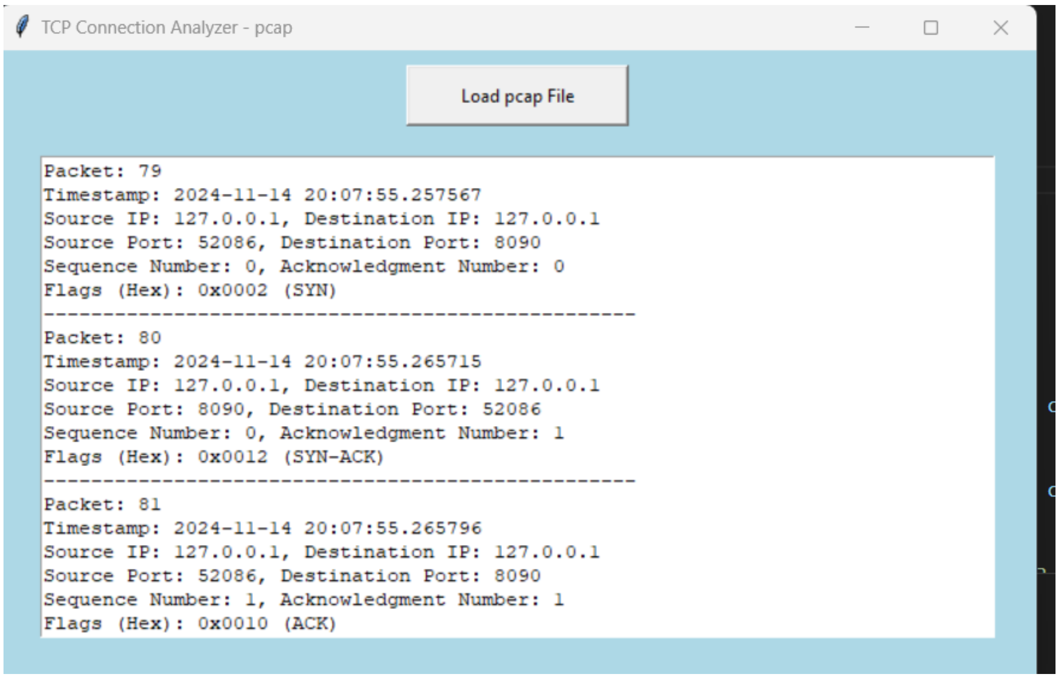


```
File name: t.pcap
Save as: Wireshark/tcpdump/... - pcap
```

2) Load the pcap file to analyze the connection

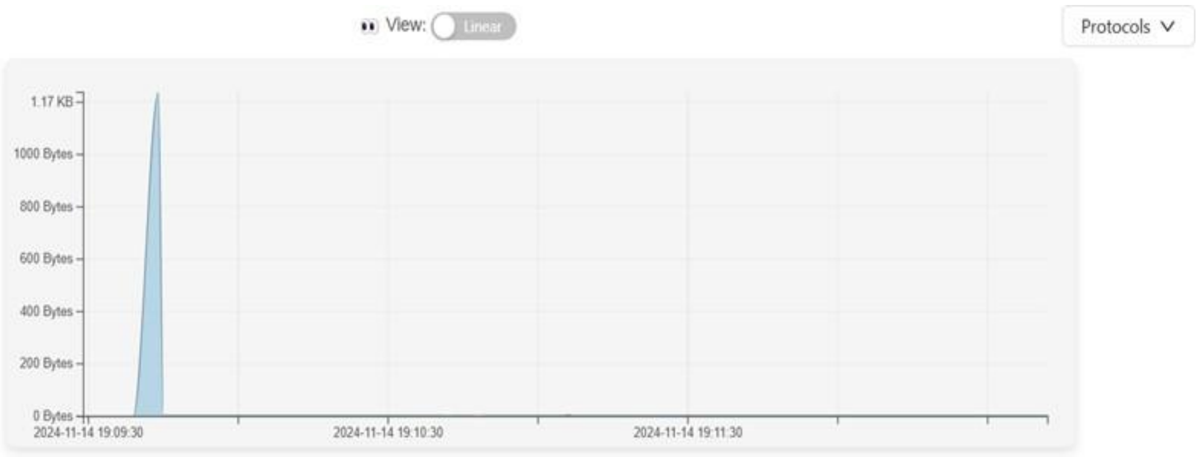


3) Analyze the connection



```
Packet: 79
Timestamp: 2024-11-14 20:07:55.257567
Source IP: 127.0.0.1, Destination IP: 127.0.0.1
Source Port: 52086, Destination Port: 8090
Sequence Number: 0, Acknowledgment Number: 0
Flags (Hex): 0x0002 (SYN)
--------------------------------------------------
Packet: 80
Timestamp: 2024-11-14 20:07:55.265715
Source IP: 127.0.0.1, Destination IP: 127.0.0.1
Source Port: 8090, Destination Port: 52086
Sequence Number: 0, Acknowledgment Number: 1
Flags (Hex): 0x0012 (SYN-ACK)
--------------------------------------------------
Packet: 81
Timestamp: 2024-11-14 20:07:55.265796
Source IP: 127.0.0.1, Destination IP: 127.0.0.1
Source Port: 52086, Destination Port: 8090
Sequence Number: 1, Acknowledgment Number: 1
Flags (Hex): 0x0010 (ACK)
```

4) Analyze pcap file using apackets.com



EXPLANATION

The code defines a GUI-based TCP Connection Analyzer application using Python’s tkinter library. It includes three primary modules: MainWindow, ServerWindow, and ClientWindow, alongside an AnalyzerWindow for handling .pcap file analysis. The MainWindow acts as the central dashboard, offering buttons to open separate windows for server, client, and analyzer functionalities. The ServerWindow initiates a TCP server on localhost at port 8090, waiting for a client connection. Once connected, it simulates a three-way handshake and allows the server to display received messages from the client. The ClientWindow connects to the server on the same port, simulating a handshake sequence and enabling message exchanges. Both server and client support message-sending and display connection status updates. The AnalyzerWindow lets users load a .pcap file (using pyshark), filter for TCP packets on port 8090, and display details such as packet numbers, timestamps, IP addresses, ports, sequence, and acknowledgment numbers, as well as TCP flags in a human-readable form. The application’s modular design facilitates TCP connection testing and basic network packet analysis through a user-friendly interface.

FUNCTIONALITY AND USE CASES

1. Capturing the Handshake Process

- The TCP handshake analyzer monitors the initial three-way handshake process essential for establishing a TCP connection. This involves capturing and analyzing the exchange of three specific messages:
    - SYN (synchronize): Initiated by the client to request a connection.
    - SYN-ACK (synchronize-acknowledge): Sent by the server to acknowledge the client's request and signal its readiness to connect.
    - ACK (acknowledge): Sent by the client to finalize the connection establishment.
- By capturing these packets, the analyzer verifies if the connection was established successfully, forming the basis for further diagnostics.

2. Verification of Connection Parameters

- The analyzer inspects various TCP header fields in each packet, such as sequence numbers, acknowledgment numbers, window size, and flags.
- It checks for anomalies or inconsistencies, such as mismatched sequence numbers or incorrect acknowledgment numbers, which may indicate network misconfigurations or potential security issues.

3. Diagnosing Connection Issues

- The analyzer identifies problems that occur during the handshake process, such as:
    - Connection Refusals: When the server actively rejects the connection request.
    - Timeouts: When one party fails to receive a timely response.
    - Resets (RST packets): Abrupt connection termination by one party.
    - This functionality aids network administrators and developers in diagnosing connection issues, allowing them to quickly address factors that may disrupt connectivity.

4. Monitoring for Security Threats

- The TCP handshake analyzer serves as a valuable tool for identifying specific types of attacks that exploit the connection setup phase, including:
  - SYN Flood Attacks: A denial-of-service attack in which an attacker sends a large number of SYN requests without completing the handshake, overloading the server.
  - Man-in-the-Middle Attacks: The analyzer can detect unusual retransmissions or packet manipulations that may suggest interception.
- By flagging these anomalies, the analyzer strengthens the security of theTCP connection establishment process.

5. Logging and Reporting Handshake Events

- The analyzer logs each step of the handshake process, recording time stamps, IP addresses, port numbers, and details of the handshake messages.
- These logs provide valuable insights for troubleshooting, auditing, and performance analysis, as they offer historical data on connection attempts, successes, and failures.

6. Performance Monitoring

- By measuring the time taken for each handshake, the analyzer can identify delays in the connection setup process. Extended handshake durations may indicate network congestion, server overload, or routinginefficiencies.
- Continuous monitoring of handshake performance helps in assessing overall network health and detecting potential bottlenecks.

7. Integration with Network Analysis Tools

- Many TCP handshake analyzers are designed to integrate with network analysis tools, such as Wireshark, enabling deep packet inspection and detailed visualization of handshake events within the broader context of network traffic.

- This integration provides a comprehensive view of network behavior, assisting in both detailed protocol analysis and general performance monitoring.

Common Use Cases

- Network Troubleshooting: Network administrators use a TCP handshake analyzer to pinpoint issues in connection setup, especially in cases where applications experience intermittent connectivity.
- Security Audits: Security teams rely on the analyzer to monitor for handshake-related anomalies that could indicate potential network attacks.
- Application Performance Assessment: Developers can use a handshake analyzer to optimize the efficiency of their applications' connection setup, especially in latency-sensitive environments.
- Compliance and Logging: Organizations use handshake logs for compliance, ensuring that all connections are properly documented andexhibit expected behavior.

CONCLUSION

The TCP 3-Way Handshake is a crucial process for establishing a reliable communication channel between a client and a server. This handshake ensures both sides are synchronized and prepared for secure data transmission. In this project, we developed an application that visualizes the TCP 3-Way Handshake, featuring both a server and client GUI interface, along with an analyzer that inspects pcap files and generates insightful graphs. By capturing the packet exchanges during the handshake (SYN, SYN-ACK, ACK), the application provides a clear understanding of how connections are established, validated, and troubleshooted. The integration of Wireshark-like packet capture functionality with real-time analysis makes the app a useful tool for network engineers, security analysts, and anyone working with TCP/IP networking. It aids in confirming the integrity of connections and detecting potential issues in a user-friendly format, allowing for easier network management and security monitoring. This project demonstrates not only the significance of the TCP 3-Way Handshake but also how modern tools like this application can simplify complex networking tasks.

ACKNOWLEDGEMENT

We would like to thank the developers and contributors of Wireshark and pcap libraries, which served as foundational resources for capturing and analyzing TCP packets. Special thanks to the community at apackets.com for providing a platform to generate visual packet graphs, which have been integrated into the analyzer. Additionally, we appreciate the insights from network engineers and security professionals who provided valuable feedback during the development and testing phases of this project. At last, we mention our gratitude to our teacher for providing us with this wonderful learning opportunity.

REFERENCES

1) RFC 793 - Transmission Control Protocol (TCP). (1981). IETF. RFC 793
2) Wireshark: The world's foremost network protocol analyzer. Wireshark Official Website
3) apackets.com: Network traffic analysis tool with packet visualization capabilities.
4) TCP 3-Way Handshake Explained. (2023). Retrieved from NetworkChuck.
5) PCAP (Packet Capture). (2023). Overview of packet capture formats. Retrieved from Wireshark Documentation