A REPORT ON

TOPIC: **NETWORK TRAFFIC ANALYSER USING PYSHARK**

OBJECTIVE :

The primary objective of this project was to develop a robust and intuitive Network Traffic Analyzer, a Python-based application designed to monitor and analyze network traffic in real-time utilizing the PyShark library.Initially we were to develop a tool which captures live network packets and displays key information such as source IP, destination IP, and protocol in a user-friendly graphical interface built with Tkinter.

But, along with these additionally, we were also able to develop an email alert system which triggers alerts when traffic exceeds predefined thresholds and plotted graphs.

TOOLS USED:

1. Python : Used for scripting and developing the entire application
2. PyShark : Python wrapper for capturing live network packets
3. Tkinter : GUI library for creating user interface
4. MatplotLib : Plotting library used for visualizing real-time data
5. smtplib and email.message: Standard Python libraries for sending email alerts

WORK FLOW :

1. Project Initialization

- Project is setup using Python with necessary libraries: *tkinter, pyshark, threading, smtplib, email, matplotlib,* and *asyncio.*
- A global stop event is initialized using threading.Event() for controlling the packet capture process

2. Email Alert Configuration

- Constants for the email account credentials and recipient address are defined.
- The *send_email_alert* function is implemented to send email notifications using the *smtplib* and *email* libraries.

3. Packet Capture Function

- The *capture_packets* function is created to handle packet capturing using *pyshark.*
- The *asyncio* event loop is initialized to allow asynchronous packet capture.
- A LiveCapture is setup on the specified network interface and the captured packets are saved to a PCAP file.
- The number of TCP and UDP packetsare counted , a live plot of packet count over time is updated using matplotlib, and packet information is inserted into the tkinter text widget.
- An alert is triggered and an email is sent if packet count exceeds the threshold.

4. Submit Capture Details

- The *submit_capture_details* function is defined to handle user inputs and start the packet capture process.
- User inputs for the network interface, PCAP file path, and capture duration is retrieved.
- A separate thread is created and started for capturing packets to ensure the GUI remains responsive.
- For the specified capture duration it sleeps and the stop flag is set to stop the .

5. File Browser Dialog

- The *browse_pcap_file* function is implemented to allow users to select a PCAP file using a file dialog.

6. Stop Capture Function

- The *stop_capture* function is defined to set the stop event, stopping the packet capture process.

7. GUI Setup

- The main GUI window is setup using tkinter.
- Input fields and labels for network interface, PCAP file path, and capture duration are created.
- Buttons arre added for starting and stopping the capture, and browsing for a PCAP file.

- A text widget is added to display captured packet information, with tags for coloring text based on the protocol.

8. Live Packet Display and Plotting

- As packets are captured, their information is displayed in the text widget with color-coded tags for TCP and UDP packets.
- A live plot of packet count over time is updated using matplotlib.

9. Alerts and Notifications

- If the packet count exceeds the defined threshold, a warning message box is displayed and an email alert is sent

CODE :

IMPORTING LIBRARIES AND INITIALIZING VARIABLES

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pyshark
import threading
import time
import smtplib
from email.message import EmailMessage
import matplotlib.pyplot as plt
import asyncio

stop_event = threading.Event()

EMAIL_ADDRESS = "your mail"
EMAIL_PASSWORD = "your password"
RECIPIENT_ADDRESS = "recepient mail"
ALERT_THRESHOLD = 100        // Email alert is sent when this threshold is reached

EMAIL ALERT FUNCTION
def send_email_alert(subject, body, to):
    msg = EmailMessage()
    msg.set_content(body)
    msg['From'] = EMAIL_ADDRESS
    msg['To'] = RECIPIENT_ADDRESS
    msg['Subject'] = subject

    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
```

```
    server.send_message(msg)
    server.quit()

PACKET CAPTURE FUNCTION
def capture_packets(interface_name, pcap_file_path, capture_duration):
    asyncio.set_event_loop(asyncio.new_event_loop())
    loop = asyncio.get_event_loop()

    async def run_capture():
        # Initialize live capture on the specified interface and save to pcap file
        capture = pyshark.LiveCapture(interface=interface_name, output_file=pcap_file_path)
        print(f"Starting capture on interface {interface_name} for {capture_duration} seconds...")
        tcp_count = 0
        udp_count = 0
        alarm_triggered = False

        # Capture packets for the specified duration
        capture.sniff(timeout=capture_duration)
        start_time = time.time()
        packet_count = 0

        # Optionally, print the captured packets
        for packet in capture.sniff_continuously():
            if hasattr(packet, 'ip'):
                protocol = packet.transport_layer
                src_ip = packet.ip.src
                dst_ip = packet.ip.dst
                packet_info = f"Protocol: {protocol}, Source: {src_ip}, Destination: {dst_ip}\n"
                text_widget.insert(tk.END, packet_info)

                # Apply color tags based on protocol
                if protocol == 'TCP':
                    text_widget.tag_add('tcp', f"{packet_count+1}.0", f"{packet_count+1}.end")
                elif protocol == 'UDP':
                    text_widget.tag_add('udp', f"{packet_count+1}.0", f"{packet_count+1}.end")
                text_widget.see(tk.END)
                packet_count += 1
                elapsed_time = time.time() - start_time
                if protocol == 'TCP':
                    tcp_count += 1
                elif protocol == 'UDP':
                    udp_count += 1
                plt.clf()
                plt.plot(elapsed_time, packet_count, 'bo-')
                plt.xlabel('Time (seconds)')
                plt.ylabel('Packet Count')
                plt.title('Packet Count over Time')
                plt.grid(True)
                plt.pause(0.1)
                if packet_count > ALERT_THRESHOLD and not alarm_triggered:
```

```
                print("Network traffic is too much!")
                alarm_triggered = True
                messagebox.showwarning("Network Alarm", "Network traffic has exceeded the
threshold!")
                send_email_alert("NETWORK ALARM", "Traffic exceeded the threshold",
"6362382724@jio.ril.com")

            if elapsed_time >= capture_duration:
                break
        print(f"Capture stopped. Packets saved to {pcap_file_path}")
        labels = ['TCP', 'UDP']
        sizes = [tcp_count, udp_count]
        colors = ['blue', 'orange']
        plt.figure()
        plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
        plt.title('Packet Distribution')
        plt.show()
        plt.figure()

    loop.run_until_complete(run_capture())

SUBMIT CAPTURE DETAILS FUNCTION
def submit_capture_details():
    global stop_flag
    stop_flag = False
    interface_name = 'Wi-Fi'
    pcap_file_path = entry_pcap_file_path.get()
    capture_duration = int(entry_capture_duration.get())

    # Create a separate thread for capturing packets
    capture_thread = threading.Thread(target=capture_packets, args=(interface_name,
pcap_file_path, capture_duration))
    capture_thread.start()

    # Wait for the capture duration
    time.sleep(capture_duration)

    # Set the stop flag to true to stop the capture
    stop_flag = True
    capture_packets(interface_name, pcap_file_path, capture_duration)

FILE BROWSER DIALOG
def browse_pcap_file():
    filepath = filedialog.askopenfilename(filetypes=[("PCAP Files", "*.pcap")])
    if filepath:
        entry_pcap_file_path.delete(0, tk.END)
        entry_pcap_file_path.insert(0, filepath)

STOP CAPTURE FUNCTION
def stop_capture():
```

```
    stop_event.set()

# GUI Setup
root = tk.Tk()
root.title("Packet Capture Display")

frame_inputs = ttk.Frame(root, padding="10")
frame_inputs.grid(row=0, column=0, sticky=(tk.W, tk.E))

ttk.Label(frame_inputs, text="Network Interface:").grid(row=0, column=0, padx=5, pady=5,
sticky=tk.W)
ttk.Label(frame_inputs, text="Wi-Fi").grid(row=0, column=1, padx=5, pady=5, sticky=tk.W)

ttk.Label(frame_inputs, text="PCAP File Path:").grid(row=1, column=0, padx=5, pady=5,
sticky=tk.W)
entry_pcap_file_path = ttk.Entry(frame_inputs)
entry_pcap_file_path.grid(row=1, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))
browse_button = ttk.Button(frame_inputs, text="Browse", command=browse_pcap_file)
browse_button.grid(row=1, column=2, padx=5, pady=5, sticky=(tk.W, tk.E))

ttk.Label(frame_inputs, text="Capture Duration (seconds):").grid(row=2, column=0, padx=5,
pady=5, sticky=tk.W)
entry_capture_duration = ttk.Entry(frame_inputs)
entry_capture_duration.grid(row=2, column=1, padx=5, pady=5, sticky=(tk.W, tk.E))

start_button = ttk.Button(frame_inputs, text="Start Capture",
command=submit_capture_details)
start_button.grid(row=2, column=2, padx=5, pady=5, sticky=(tk.W, tk.E))

stop_button = ttk.Button(frame_inputs, text="Stop Capture", command=stop_capture)
stop_button.grid(row=3, column=2, padx=5, pady=5, sticky=(tk.W, tk.E))

text_widget = tk.Text(frame_inputs, height=20, width=100)
text_widget.grid(row=3, column=0, columnspan=3, padx=5, pady=5, sticky=(tk.W, tk.E))

# Add tags for coloring text
text_widget.tag_configure('tcp', foreground='blue')
text_widget.tag_configure('udp', foreground='orange')

root.mainloop()
```

RESULTS:

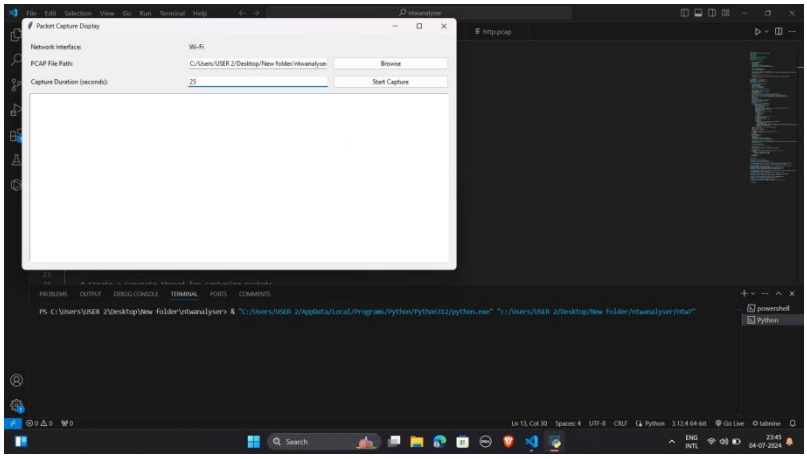

Fig 1. GUI for Start and Stop capture

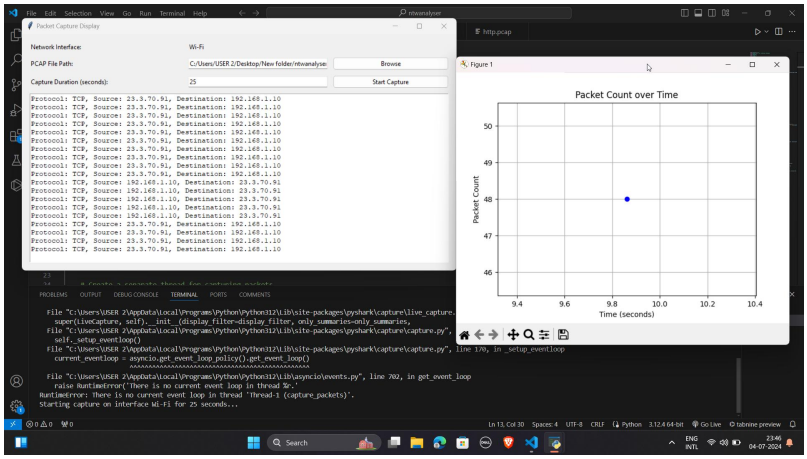

Fig 2. Source IP, Destination IP, Protocol and Packet count over Time

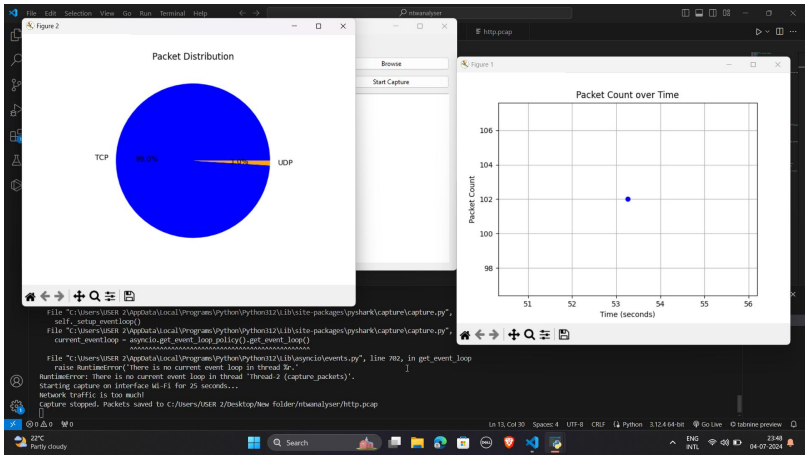

Fig 3.Packet Distribution(TCP vs UDP)

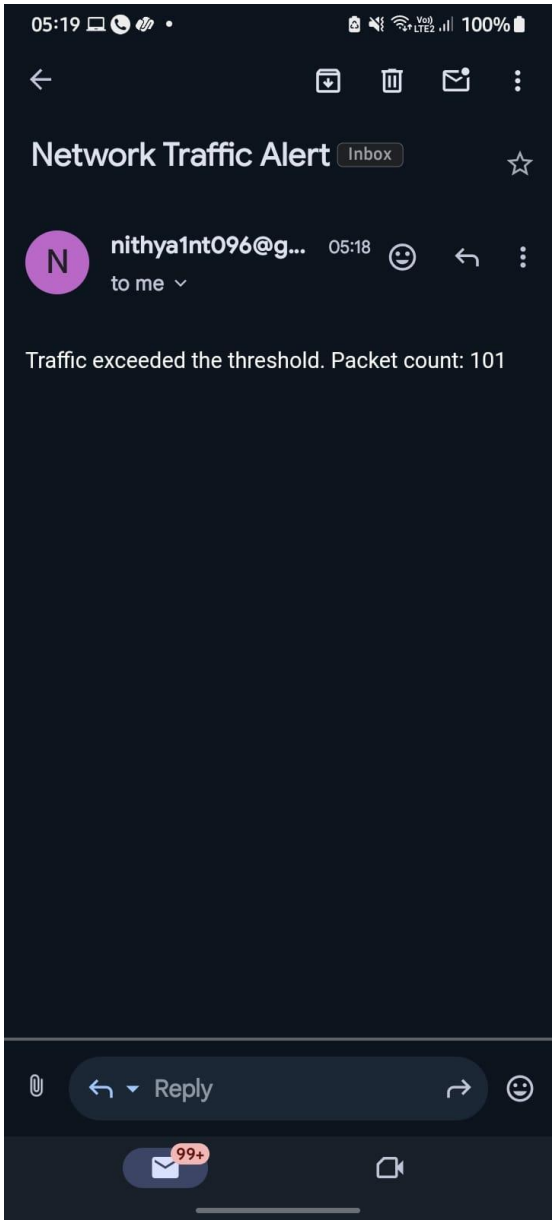

Fig 4. Email Alert

CONCLUSION:

This project provides an efficient and user-friendly solution for real-time network traffic monitoring and alerting. By combining powerful tools like `pyshark` for packet capturing and `tkinter` for the graphical interface, we offer users an intuitive way to monitor network activity and respond to potential traffic issues. The inclusion of automated email alerts ensures that users are promptly notified of any abnormal network behavior, enhancing security and responsiveness. The project demonstrates a seamless integration of various Python libraries to create a comprehensive network monitoring tool that is both accessible and effective.