MINI-PROJECT

NETWORK TRAFFIC ANALYSER

**Ex** : **Name**: KEERTHICK V

**Date:**   **Register No.** : 3122 22 5002 058

Problem statement:

Network administrators and security professionals need to capture and analyze network packets to manage traffic, optimize performance, and detect threats. Manual packet analysis is complex and time-consuming, requiring automated solutions for efficiency. This project aims to develop a Python-based tool to capture, analyze, and visualize network traffic, providing insights into packet sizes, protocols, and traffic patterns. This will simplify network monitoring and enhance security and performance management.

Problem Description:

Network traffic analysis is essential for maintaining network health, security, and performance optimization. However, the manual process of capturing and analyzing network packets is intricate and time-consuming, requiring specialized knowledge to interpret raw data and identify meaningful patterns. Network administrators and security professionals often face challenges in efficiently monitoring network traffic, diagnosing issues, and detecting malicious activities due to the complexity of handling various network protocols and the sheer volume of data.

There is a pressing need for an automated, user-friendly solution that can seamlessly capture network packets, analyze their contents, and present the findings in an intuitive and visual format. Such a tool would greatly aid in the identification of network anomalies, traffic bottlenecks, and potential security threats, thereby enhancing the overall management and security of network infrastructures.

This project aims to develop a Python-based tool that addresses these challenges by capturing network packets from a specified interface, analyzing the packets to extract key information such as packet size, source and destination IP addresses, and protocols, and visualizing the results. The tool will help users, regardless of their expertise level, to quickly and accurately understand network traffic characteristics and make informed decisions for network management and security.

Code:

import pyshark

import pandas as pd

import matplotlib.pyplot as plt

def capture\_packets(interface, count=100):

print(f"Capturing {count} packets from interface {interface}...")

capture = pyshark.LiveCapture(interface=interface)

packets = []

packets\_captured=0

for packet in capture.sniff\_continuously(packet\_count=count):

packets.append(packet)

packets\_captured += 1

print(f"Packets captured: {packets\_captured}/{count}")

return packets

def analyze\_packet(packet):

try:

packet\_info = {'packet\_size': int(packet.length)}

# Check if the packet has an IP layer

if hasattr(packet, 'ip'):

packet\_info.update({

'src\_ip': packet.ip.src,

'dst\_ip': packet.ip.dst

})

if 'TCP' in packet:

packet\_info.update({

'protocol': 'TCP',

'src\_port': packet.tcp.srcport,

'dst\_port': packet.tcp.dstport

})

if packet.tcp.srcport == '21' or packet.tcp.dstport == '21':

packet\_info['protocol'] = 'FTP'

elif packet.tcp.srcport == '25' or packet.tcp.dstport == '25':

packet\_info['protocol'] = 'SMTP'

elif 'UDP' in packet:

packet\_info.update({

'protocol': 'UDP',

'src\_port': packet.udp.srcport,

'dst\_port': packet.udp.dstport

})

elif 'IGMP' in packet:

packet\_info.update({

'protocol': 'IGMP'

})

elif 'ICMP' in packet:

packet\_info.update({

'protocol': 'ICMP'

})

else:

packet\_info.update({

'protocol': 'Other IP'

})

elif hasattr(packet, 'arp'):

packet\_info.update({

'protocol': 'ARP',

'src\_ip': packet.arp.src\_proto\_ipv4,

'dst\_ip': packet.arp.dst\_proto\_ipv4

})

elif hasattr(packet, 'rarp'):

packet\_info.update({

'protocol': 'RARP'

})

elif hasattr(packet, 'dns'):

packet\_info.update({

'protocol': 'DNS'

})

elif hasattr(packet, 'dhcp'):

packet\_info.update({

'protocol': 'DHCP'

})

elif hasattr(packet, 'http'):

packet\_info.update({

'protocol': 'HTTP'

})

else:

pass

print(f"Packet: {packet\_info}")

return packet\_info

except Exception as e:

print(f"Error analyzing packet: {e}")

return None

def visualize\_packet\_size\_distribution(data):

try:

packet\_sizes = [packet['packet\_size'] for packet in data if packet]

if not packet\_sizes:

print("No valid packet sizes to display.")

return

plt.figure(figsize=(10, 6))

plt.hist(packet\_sizes, bins=20, color='skyblue', edgecolor='black')

plt.title('Packet Size Distribution')

plt.xlabel('Packet Size (bytes)')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

except Exception as e:

print(f"Error visualizing packet size distribution: {e}")

def visualize\_traffic\_by\_protocol(data):

try:

df = pd.DataFrame([packet for packet in data if packet])

if df.empty:

print("No valid packet data to display.")

return

protocol\_counts = df['protocol'].value\_counts()

plt.figure(figsize=(8, 6))

protocol\_counts.plot(kind='bar', color='salmon')

plt.title('Traffic by Protocol')

plt.xlabel('Protocol')

plt.ylabel('Packet Count')

plt.xticks(rotation=45)

plt.grid(True)

plt.show()

except Exception as e:

print(f"Error visualizing traffic by protocol: {e}")

def main():

interface = 'Wi-Fi' # Update with your network interface

packet\_count = 1000

# Capture packets

packets = capture\_packets(interface, count=packet\_count)

# Analyze packets

decoded\_packets = [analyze\_packet(packet) for packet in packets]

# Visualize packet size distribution

visualize\_packet\_size\_distribution(decoded\_packets)

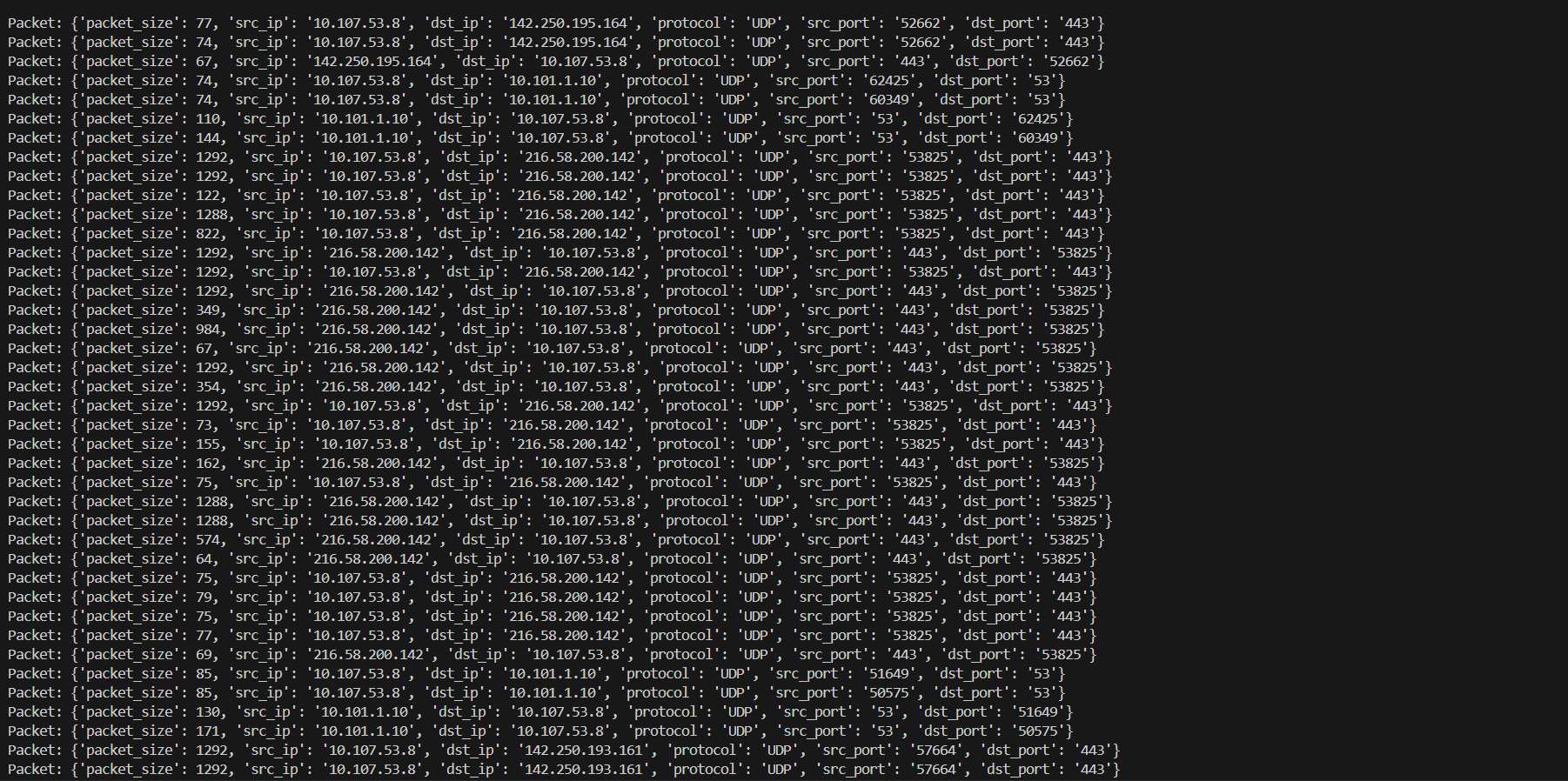
# Visualize traffic by protocol

visualize\_traffic\_by\_protocol(decoded\_packets)

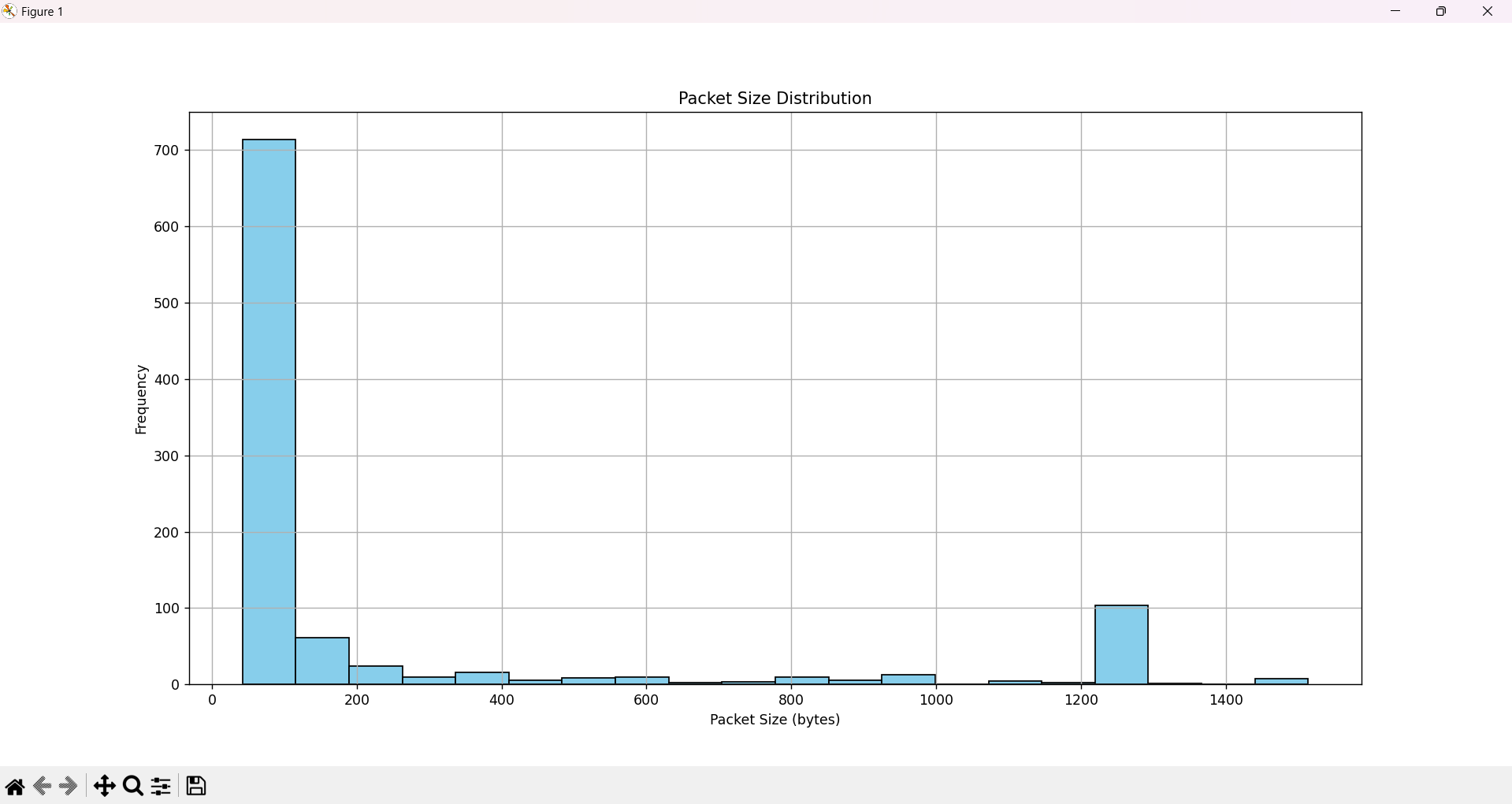
if \_\_name\_\_ == '\_\_main\_\_':

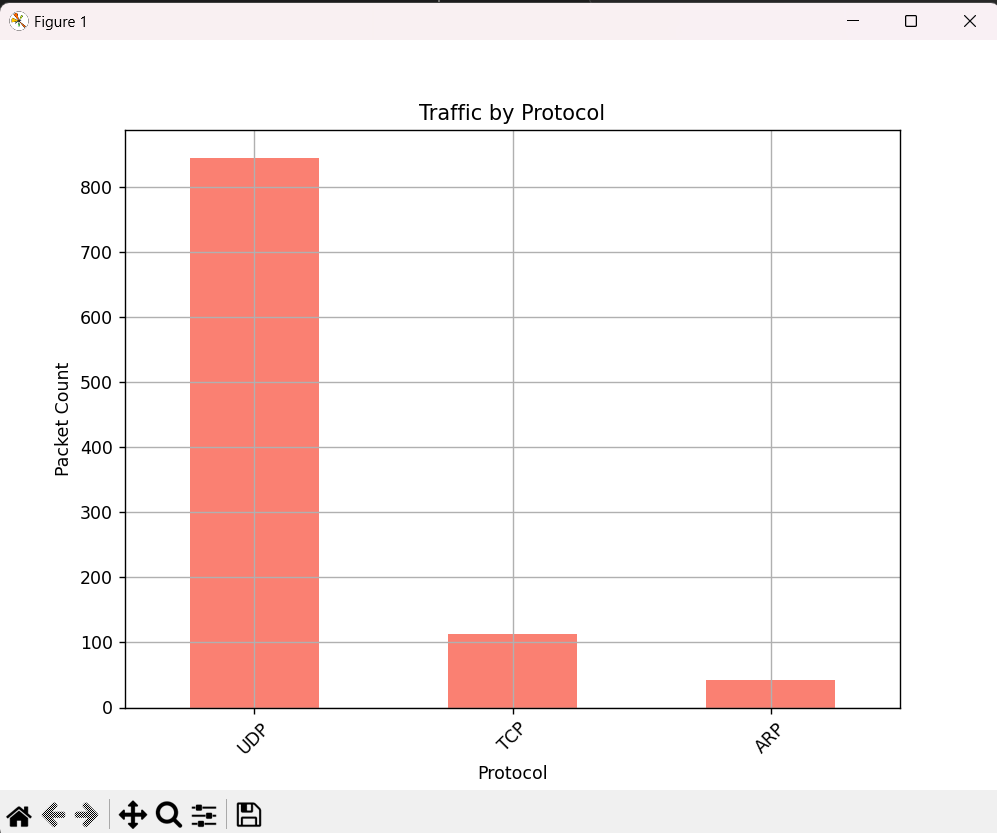
main()

Output:









Result:

The Python-based tool effectively captures and analyzes network packets, extracting key information such as packet size, IP addresses, and protocols. It provides visualizations of packet size distribution and traffic volume by protocol, simplifying network traffic analysis. This enhances the ability to identify anomalies, optimize performance, and detect security threats efficiently.