A Project Report

on

Packet Header Analysis

Submitted to

Dr. Durga Toshniwal

Submitted by

Gaurav Kumar (19911014), **Asmita Mahajan** (20911002)

Srishti Sharma (20911009), **Kanhu Charan Gouda** (19911006)

Hem Chandra Joshi (20911005)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
ROORKEE- 247667, UTTARAKHAND(INDIA)

Nov - 2020

Contribution

Project Number – 1 Group Number - 7

Contribution of members:

Gaurav Kumar: Gaurav Kumar has contributed to the collection of PCAP files from various data sources. He has also implemented the python code to read the different parameter values of ARP and ICMP packets and store the fetched values in the excel file, which would be required to visualize and summarize the packet data. At last, he has prepared the part of the presentation and project report. (P.N. – 3, 21-28, 38-40)

Asmita Mahajan: Asmita Mahajan has contributed to this project to analyze and capture packets from Wireshark. She has implemented python code to extract and display information regarding Ethernet header and Ethernet payload, IP header and IP payload, TCP/UDP header, and payload. Further, she implemented the code to convert the required information to a more presentable format, i.e., extracted the header fields into a data frame for further analysis. At last, she has prepared the part of the presentation and project report. (P.N. – 19-21, 37)

Srishti Sharma: Sristi Sharma has prepared in documentation part, read different existing research paper based on packet capture, and analyzed various fields of packet structure parameters such as source address, destination address, Protocol using different layers such as TCP/IP protocol, IPv4 header format of PCAP file using Wireshark software tool. (P.N. – 7-12, 19-21)

Kanhu Charan Gouda: Kanhu Charan Gouda has prepared in documentation part, analyzed different packet structure parameters such as source address, a destination address, source port address destination on port address, protocols using different layers such as ARP protocol of PCAP file using Wireshark software tool. (P.N. – 13-18, 21-28)

Hem Chandra Joshi: Hem Chandra Joshi has contributed to coding parts. He used the processed excel file and extracted information from it to visualize and summarize data. He read about the protocols and types of graphs. At last, he has prepared the part of the presentation and project report. (P.N. – 23-36)

Abstract

Packet capture (PCAP) is a C array format file having various network packets which need to be read to diagnose and solve network problems such as identify security threats, troubleshooting undesirable network behaviors, determine network congestion, identifying data or packet loss, and forensic network analysis. We have analyzed various packets to determine all the fields in the packet header like source/destination MAC and IP address, port numbers, different protocols used in different layers etc. A packet reader has been implemented in the python language which takes PCAP file as an input and gives all the parameters as an output with visulization of data. At last, based on the output we have prepared a summary for captured packets.

Table of contents

List of figures

List of tables

1. Introduction

Packet capture:

Packet capture is a networking term for intercepting a data packet that is crossing a specific data network. Once a data packet is capturing in real-time, it is stored for a period of time so that it can be analyzed, and then either be downloaded, achieved or discarded. Packets are capture and examined to help diagnose and solve network problems such as identify security threats, troubleshooting undesirable network behaviors, identify network congestion, identifying data or packet loss, and forensic network analysis. Packet capture can be performed in-line or using a copy of the traffic that is sent by network switching devices to a packet capture device [1].

Full packet capture:

Entire packets or specific portions of a packet can be captured. A full packet includes two things i.e. a payload and a header. The payload is the actual contents of the packet, while the header contains metadata, including the packet's source and destination address.

Packet capture analysis:

Analysis of packet capture data typically requires significant technical skills, and often is performed with tools such as Wireshark.

Wireshark:

Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communication protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006. In other words, Wireshark is a packet sniffer and analysis tool. It captures network traffic on the local network and stores that data for offline analysis. Wireshark captures network traffic from Ethernet, Bluetooth and wireless [1].

2. Literature Survey

TCP/IP Protocol Suite:

The TCP/IP protocol stack provides layers for networks and systems each having its own protocols that allows communications between any types of devices. This stack consists of five separate layers. These layers are although separate but related to each other for any communication to take place. The Internet protocol suite defines these five layers. TCP/IP says most about the network and transport layers' protocols.

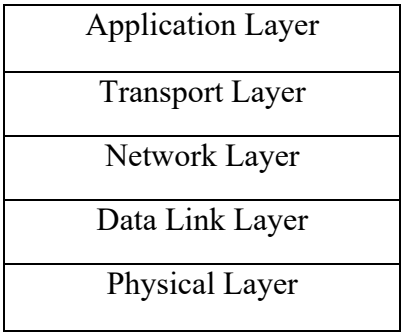


Fig. 1 Layers of TCP/IP Protocol Suite

In this project, we are identifying all the packets for all the different protocols they are using while passing through the network. Each layer has its own header which gives the important information about the packet and allows it to pass through the network. The various packets passing through the network are analyzed for the Protocol they are using in each layer.

Transport Layer Protocols (TCP): Transport layer protocols used by the packets are TCP and User Datagram Protocol (UDP). Each Protocol has its own header and containing the information about the packet. TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening for connection requests from clients before a connection is established. Three-way handshake, retransmission, and error-detection adds to reliability but lengthens latency. The header of a TCP segment can range from 20 - 60 bytes where 40 bytes are for options [2].

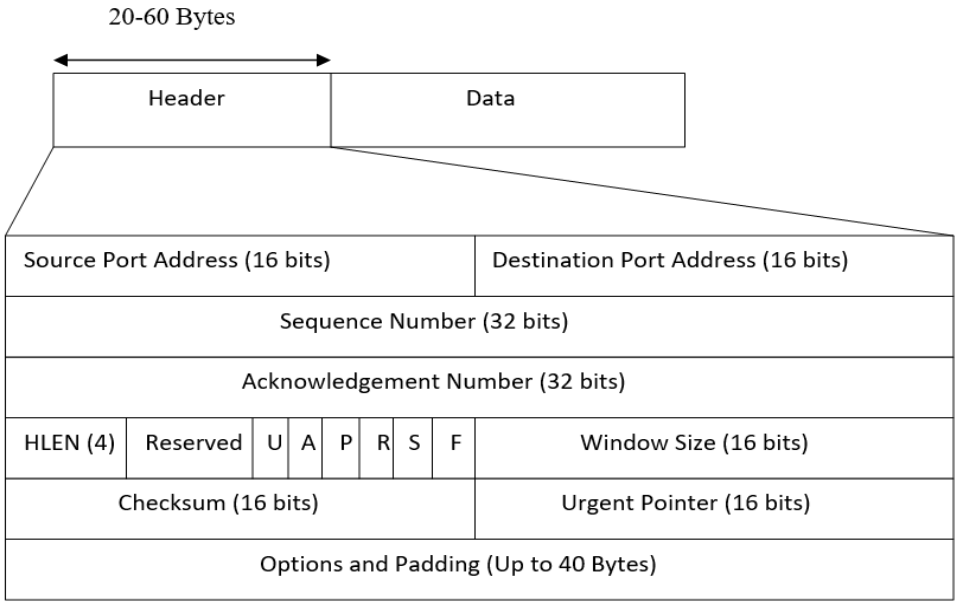


Fig.2 TCP Header

Source Port Address: It is 16-bit field that holds the port address of the application that is sending the data segment.

Destination Port Address: It is a 16-bit field that holds the port address of the application in the host that is receiving the data segment.

Sequence Number: It is a 32-bit field that holds the sequence number, i.e. the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end if the segments are received out of order.

Acknowledgement Number: It is a 32-bit field that holds the acknowledgement number, i.e., the byte number that the receiver expects to receive next. It is an acknowledgment for the previous bytes being received successfully.

Header Length (HLEN): This is a 4-bit field that indicates the length of the TCP header by number of 4-byte words in the header, i.e., if the header is of 20 bytes (min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.

Control flags: These are 6, 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:

- URG: Urgent pointer is valid
- ACK: Acknowledgement number is valid (used in case of cumulative acknowledgement)
- PSH: Request for push
- RST: Reset the connection
- SYN: Synchronize sequence numbers
- FIN: Terminate the connection

Window size: This field tells the window size of the sending TCP in bytes.

Checksum: This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.

Urgent Pointer: This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

User Datagram Protocol (UDP): It is a Transport Layer protocol. it is unreliable and connectionless Protocol. So, there is no need to establish connection prior to data transfer. UDP header is 8-bytes fixed and simple header. First 8 Bytes contains all necessary header information and remaining part consist of data.

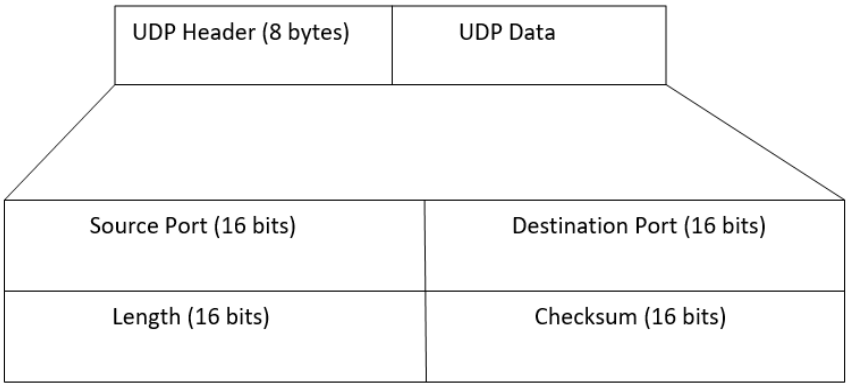


Fig.3 UDP Header

Source Port: Source Port is 2 Byte long field used to identify port number of sources.

Destination Port: It is 2 Byte long field, used to identify the port of destined packet.

Length: Length is the length of UDP including header and the data. It is 16-bits field.

Checksum: Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, pseudo header of information from the IP header and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

Network Layer Protocols: There are 2 network layer protocols that we have identified in our packets. IP (Internet Protocol) and ICMP (Internet Control Message Protocol).

IPv4 protocol: IPv4 is a connectionless protocol used for packet-switched networks. It operates on a best effort delivery model, in which neither delivery is guaranteed, nor proper sequencing or avoidance of duplicate delivery is assured. Internet Protocol Version 4 (IPv4) is the fourth revision of the Internet Protocol and a widely used protocol in data communication over different kinds of networks. IPv4 uses 32-bit (4 byte) addressing, which gives 2^{32} addresses. IPv4 addresses are written in the dot-decimal notation, which comprises of four octets of the address expressed individually in decimal and separated by periods, for instance, 192.168.1.5.

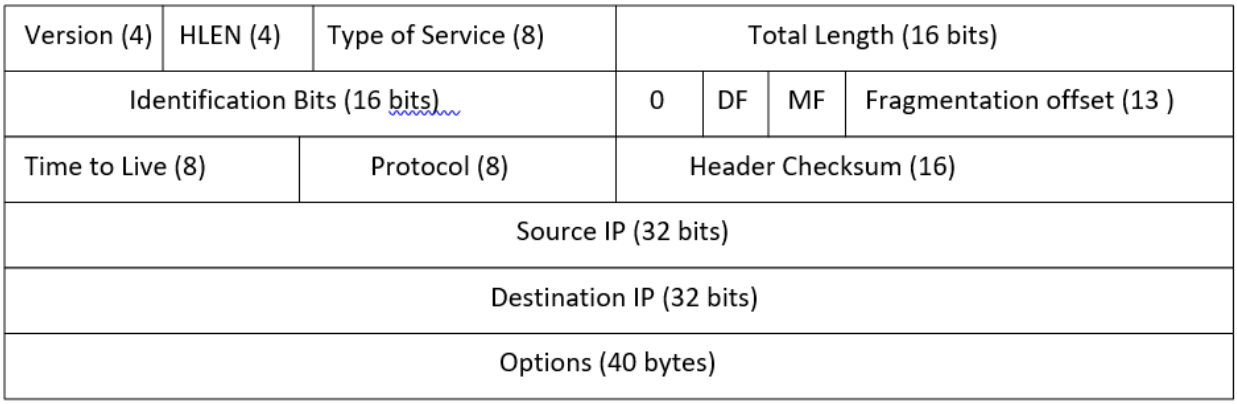


Fig.4 IPv4 Header

VERSION: Version of the IP protocol (4 bits), which is 4 for IPv4.

HLEN: IP header length (4 bits), which is the number of 32-bit words in the header. The minimum value for this field is 5 and the maximum is 15.

Type of service: Low Delay, High Throughput, Reliability (8 bits).

Total Length: Length of header + Data (16 bits), which has a minimum value 20 bytes and the maximum is 65,535 bytes.

Identification: Unique Packet Id for identifying the group of fragments of a single IP datagram (16 bits).

Flags: 3 flags of 1 bit each: reserved bit (must be zero), do not fragment flag, more fragments flag.

Fragment Offset: Represents the number of Data Bytes ahead of the particular fragment in the particular Datagram. Specified in terms of number of 8 bytes, which has the maximum value of 65,528 bytes.

Time to live: Datagram's lifetime (8 bits), It prevents the datagram to loop through the network by restricting the number of Hops taken by a Packet before delivering to thDestinationon.

Protocol: Name of the Protocol to which the data is to be passed (8 bits).

Header Checksum: 16 bits header checksum for checking errors in the datagram header.

S**ource IP address:** 32 bits IP address of the sender.

Destination IP address: 32 bits IP address of the receiver.

Option: Optional information such as source route, record route. Used by the Network administrator to check whether a path is working or not.

Internet Control Message Protocol (ICMP): IP does not have an inbuilt mechanism for sending error and control messages. It depends on ICMP to provide an error control. It is used for reporting errors and management queries. It is a supporting protocol and used by networks devices like routers for sending the error messages and operations information [2].

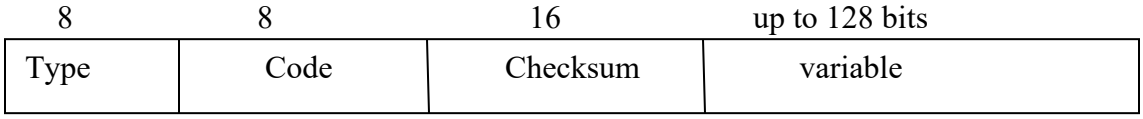


Fig 5. ICMP message format

ICMP Messages:

- **Type 0 - Echo Reply**: This is the Echo reply from the end station which is sent as a result of the Type 8 Echo.
- **Type 3 - Destination Unreachable**: The source is told that a problem has occurred when delivering a packet. There are 5 codes and these are as follows:

    - **Code 0 - Net Unreachable**: Sent by a router to a host if the router does not know a route to a requested network.
    - **Code 1 - Host Unreachable:** Sent by a router to a host if the router can see the requested network but not the destination node.
    - **Code 2 - Protocol Unreachable**: This would only occur if the destination host was reached but was not running UDP or TCP.
    - **Code 3 - Port Unreachable**: This can happen if the destination host was up and the TCP/IP was running but a particular service such as a web server that uses a specific port was not running.
    - **Code 4 - Cannot Fragment**: Sent by a router if the router needed to fragment a packet but **Do not fragment (DF)** bit was set in the IP header.
    - **Code 5 - Source Route Failed**: IP Source Routing is one of the IP Options.
- **Type 4 - Source Quench**: The source is sending data too fast for the receiver (Code 0), the buffer has filled up, slow down!
- **Type 5 – Redirect**: The source is told that there is another router with a better route for a particular packet i.e. this gateway checks its routing table and sees that another router exists on the same network with a more direct route. The Codes are assigned as follows:
    - **Code 0** - Redirect datagrams for the network.
    - **Code 1** - Redirect datagrams for the host.
    - **Code 2** - Redirect datagrams for the Type of Service and the network.
    - **Code 3** - Redirect datagrams for the Type of Service and the host.

All 4 octets of the Variable Field are used for the gateway IP address where this better router resides and packets should therefore be sent.

- **Type 8 - Echo Request**: This is sent by Ping (Packet Internet Groper) to a destination in order to check connectivity.
- **Type 11 - Time Exceeded**: The packet has been discarded as it has taken too long to be delivered.
- **Type 12 - Parameter Problem**: Identifies an incorrect parameter on the datagram (Code 0).
- **Type 13 - Timestamp request**: This gives the round-trip time to a particular destination. The Variable Field is made up of two 16-bit fields and three 32-bit fields:

- **Identifier** - as with the Echo/Echo Reply.
- **Sequence Number** - as with the Echo/Echo Reply.
- **Originate Timestamp** - Time in milliseconds since midnight within the request as it was sent out.
- **Receive Timestamp** - Time in milliseconds since midnight as the receiver receives the message.
- **Transmit Timestamp** - Time in milliseconds since midnight within the reply as it was sent out.

The Identifier and Sequence Number field are used to match timestamp requests with replies.

- **Type 14 - Timestamp reply**: This gives the round-trip time to a particular destination.
- **Type 15 - Information Request**: This allows a host to learn the network part of an IP address on its subnet by sending a message with the source address in the IP header filled and all zeros in the destination address field. Uses the two 16-bit Identifier and Sequence Number fields.
- **Type 16 - Information Reply**: This is the reply containing the network portion. These two are an alternative to RARP. Uses the two 16-bit Identifier and Sequence Number fields.
- **Type 17 - Address mask request**: Request for the correct subnet mask to be used.
- **Type 18 - Address mask response:** Reply with the correct subnet mask to be used.

Address Resolution Protocol (ARP):

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. ARP has been implemented with many combinations of network and data link layer technologies, such as IPv4, Chaosnet, DECnet and Xerox PARC Universal Packet (PUP) using IEEE 802 standards, FDDI, X.25, Frame Relay and Asynchronous Transfer Mode (ATM). The Address Resolution Protocol is a request-response protocol whose messages are encapsulated by a link layer protocol. It is communicated within the boundaries of a single network, never routed across internetworking nodes. This property places ARP into the link layer of the Internet protocol suite [3].

Packet Structure: The Address Resolution Protocol uses a simple message format containing one address resolution request or response. The size of the ARP message depends on the link layer and network layer address sizes. The message header specifies the types of network in use at each layer as well as the size of addresses of each. The message header is completed with the operation code for request (1) and reply (2). The principal packet structure of ARP packets is shown in the following table which illustrates the case of IPv4 networks running on Ethernet. In this scenario, the packet has 48-bit fields for the sender hardware address (SHA) and target hardware address (THA), and 32-bit fields for the corresponding sender and target protocol addresses (SPA and TPA). The ARP packet size in this case is 28 bytes.

Table 1: Internet Protocol (IPv4) over Ethernet ARP packet

Octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

Hardware type (HTYPE): This field specifies the network link protocol type.
Example: Ethernet is 1.
Protocol type (PTYPE): This field specifies the internetwork protocol for which the ARP request is intended. For IPv4, this has the value 0x0800.

Hardware length (HLEN): Length (in octets) of a hardware address. Ethernet address length is 6.

Protocol length (PLEN): Length (in octets) of internetwork addresses. The internetwork protocol is specified in PTYPE.

Example: IPv4 address length is 4.

Operation: Specifies the operation that the sender is performing: 1 for request, 2 for reply.

Sender hardware address (SHA): Media address of the sender. In an ARP request this field is used to indicate the address of the host sending the request. In an ARP reply this field is used to indicate the address of the host that the request was looking for.

Sender protocol address (SPA): Internetwork address of the sender.

Target hardware address (THA): Media address of the intended receiver. In an ARP request this field is ignored. In an ARP reply this field is used to indicate the address of the host that originated the ARP request.

Target protocol address (TPA): Internetwork address of the intended receiver.

Note: ARP protocol parameter values have been standardized and are maintained by the Internet Assigned Numbers Authority (IANA).

3. Problem statement

In this project, we have focus on a given file containing some packets captured (in C array format) we want to analyze those packets to determine all the fields in the packet header like source/destination MAC and IP address, port numbers, different protocols used in different layers etc. Develop an application which takes this file as an input and gives all the parameters as an output. And based on the output make a summary of which are the most used protocols in different layers.

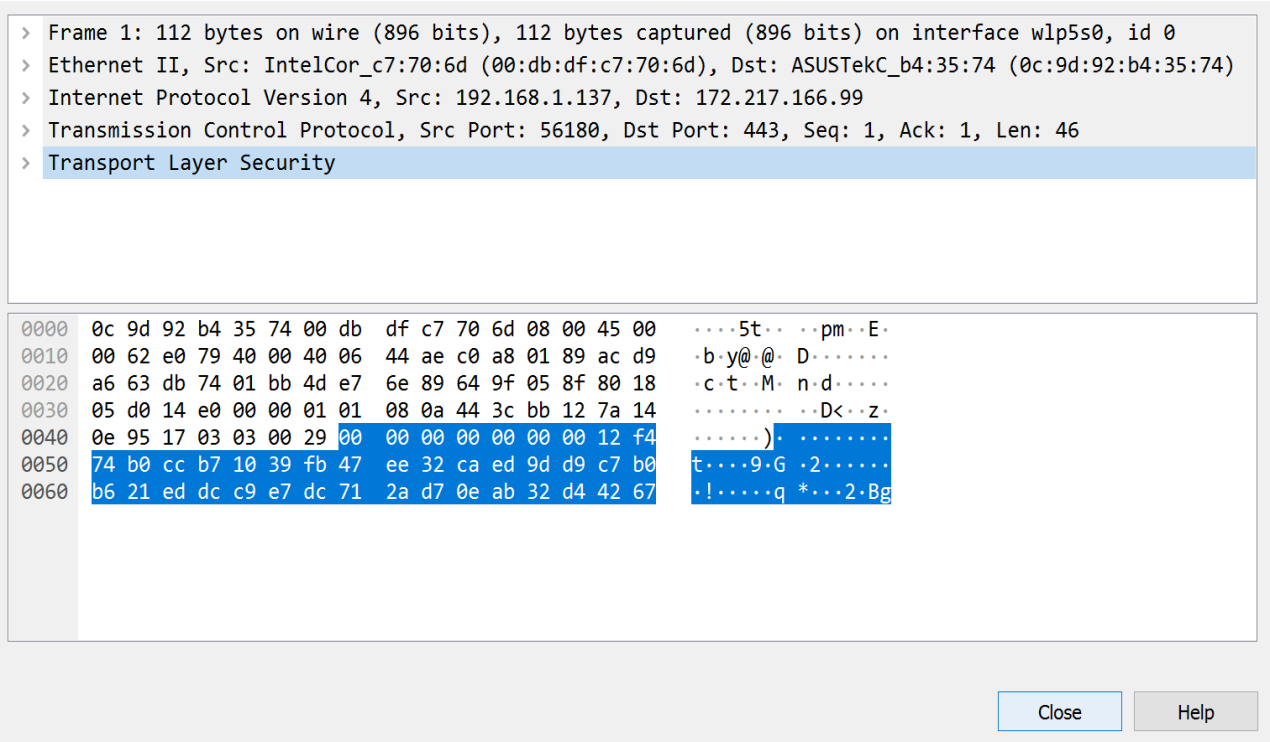


Fig 6. Packet analyzed of first frame

In this project we used Wireshark software tool which is used for network troubleshooting, analysis, software and communication protocol development, and education. It captures network traffic on the local network and stores that data for offline analysis. In figure 6, analyzed first packet consisting of source address, destination address, source port number, destination port number, and protocols used in different layer (such as IPv4 protocol, TCP protocol).

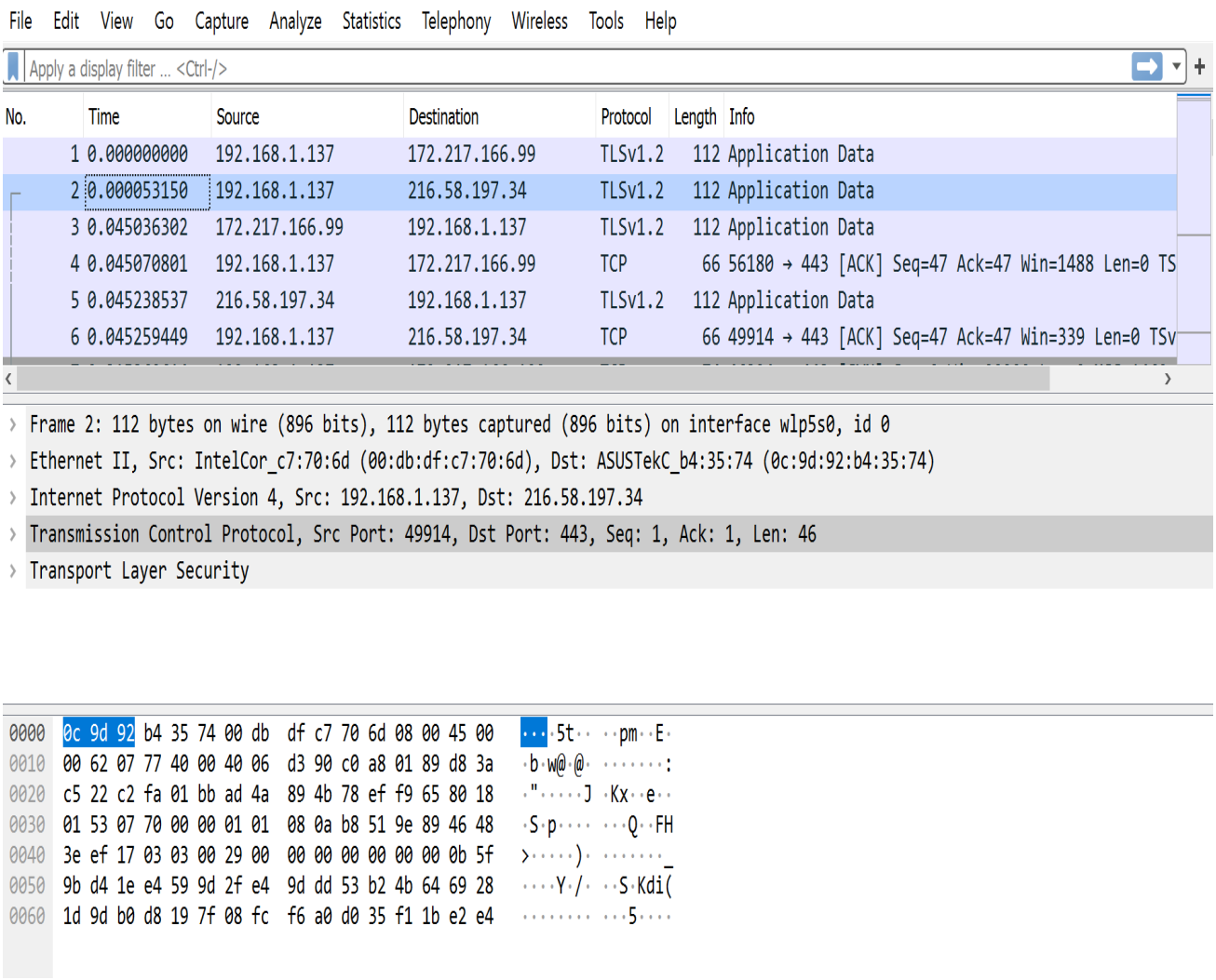


Fig 7. Packet analyzed of second frame

Similarly, in figure 7, analyzed second packet consisting of source address, destination address, source port number, destination port number, and protocols used in different layer (such as IPv4 protocol, TCP protocol).

4. Implementation

We have used the scapy package in python language to read the PCAP file then summarize and visualize the captured data using pandas and matplotlib.

```
import pandas as pd
from scapy.all import *
from scapy.utils import RawPcapReader
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP, TCP
import time
import numpy as np
import binascii
import seaborn as sns
import openpyxl
sns.set(color_codes=True)
%matplotlib inline

file2 = rdpcap('03-13 los altos.pcap')
print(file2)

file_name = file2
def display_param(n,ifile):
    ifile[n].display()
    return 0

print('Total Number of Packets is', len(file_name))
while(1):
    print('\nDo you want to continue?\n')
    c = input('Press c to continue or q to quit:')
    if(c == 'c'):
        p = int(input('\nEnter the packet whose details you want to
display:'))
```

```
        display_param(p,file_name)
    if(c == 'q'):
        break
ethernet_frame = file_name[2]
ip_packet = ethernet_frame.payload
segment = ip_packet.payload
data = segment.payload

print(ethernet_frame.summary())
print(ip_packet.summary())
print(segment.summary())
print(data.summary())

ether_fields = [field.name for field in Ether().fields_desc]
ip_fields = [field.name for field in IP().fields_desc]
tcp_fields = [field.name for field in TCP().fields_desc]
udp_fields = [field.name for field in UDP().fields_desc]
arp_fields = [field.name for field in ARP().fields_desc]
icmp_fields = [field.name for field in ICMP().fields_desc]

tcp_col = ether_fields+ip_fields + ['time'] + tcp_fields

tcp_df = pd.DataFrame(columns=tcp_col)

print('Reading TCP/UDP packets......')
for packet in file_name[IP]:
    tcp_field_values = []
    tcp_field_values.append(packet.src)
    tcp_field_values.append(packet.dst)
    tcp_field_values.append(packet.type)
    for field in ip_fields:
        if field == 'options':
```

```
tcp_field_values.append(len(packet[IP].fields[field]))

#udp_field_values.append(len(packet[IP].fields[field]))
        else:
            tcp_field_values.append(packet[IP].fields[field])
            #udp_field_values.append(packet[IP].fields[field])

    tcp_field_values.append(packet.time)
    #udp_field_values.append(packet.time)

    ip_payload_type = type(packet[IP].payload)

    for field in tcp_fields:
        try:
            if field == 'options':

tcp_field_values.append(len(packet[ip_payload_type].fields[field]
))
            else:

tcp_field_values.append(packet[ip_payload_type].fields[field])
        except:
            tcp_field_values.append(None)


#     field_values.append(len(packet[layer_type].payload))
#     field_values.append(packet[layer_type].payload.original)
#
field_values.append(binascii.hexlify(packet[layer_type].payload.o
riginal))
```

```
#                   tcp_df    =    tcp_df.append([tcp_field_values],
ignore_index=False)
#                   udp_df    =    tcp_df.append([udp_field_values],
ignore_index=False)

    temp1_df = pd.DataFrame([tcp_field_values],columns = tcp_col)
    tcp_df = pd.concat([tcp_df,temp1_df],axis=0)

print('Reading Completed.')
tcp_df = tcp_df.reset_index()
tcp_df = tcp_df.drop(columns="index")

tcp_df.head()

#Fetch ARP Data
arp_df = pd.DataFrame(columns=arp_fields)

print("Reading ARP Packets.....")
for packet in file_name[ARP]:
    field_values = []
    for field in arp_fields:
        field_values.append(packet[ARP].fields[field])

    df_append = pd.DataFrame([field_values], columns=arp_fields)
    arp_df = pd.concat([arp_df, df_append], axis=0)
print("Completed")

arp_df = arp_df.reset_index()
arp_df = arp_df.drop(columns="index")
arp_df.head()

#Fetch ICMP Data
```

```
field = icmp_fields[0:3]
icmp_df = pd.DataFrame(columns=field)
print("Reading ARP Packets.....")
for packet in file_name[ICMP]:
    field_values = []
    field_values.append(packet.type)
    field_values.append(packet.code)
    field_values.append(packet.chksum)

    df_append = pd.DataFrame([field_values], columns=field)
    icmp_df = pd.concat([icmp_df, df_append], axis=0)
print("Completed")
icmp_df = icmp_df.reset_index()
icmp_df = icmp_df.drop(columns="index")
icmp_df.head()

writer = pd.ExcelWriter('Pcap_Data.xlsx', engine ='xlsxwriter')
tcp_df.to_excel(writer, sheet_name ='TCP_UDP')
arp_df.to_excel(writer, sheet_name ='ARP')
icmp_df.to_excel(writer, sheet_name ='ICMP')
writer.save()
```

#Summarization and Visualization

```
tcp_udp_df=pd.read_excel("Pcap_Data.xlsx",sheet_name="TCP_UDP")
arp_df=pd.read_excel("Pcap_Data.xlsx",sheet_name="ARP")
icmp_df=pd.read_excel("Pcap_Data.xlsx",sheet_name="ICMP")

tcp_udp_df.columns
arp_df.columns
icmp_df.columns

tcp_udp_val_cnt = tcp_udp_df['proto'].value_counts()
```

```
tcp_udp_val_cnt.index=['TCP','UDP','ICMP']
arp_icmp_val_cnt = pd.Series([len(arp_df),len(icmp_df)], index =
['ARP','ICMP'])
proto_val_cnt = tcp_udp_val_cnt.append(arp_icmp_val_cnt)
#print(proto_val_cnt)

ax = proto_val_cnt.plot(kind='bar', figsize=(10,8),color="green",
fontsize=13)
plt.xlabel('Protocol')
plt.ylabel('Count')


for i in ax.patches:
    ax.text(i.get_x(),       i.get_height(),       i.get_height(),
fontsize=20,color='purple')

# Destination Ethernet Request in TCP_UDP
labels = list(tcp_udp_df['dst'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,9))
ax1.pie(tcp_udp_df['dst'].value_counts(), autopct='%.1f%%',shadow
= True,pctdistance=1.1, labeldistance=1)
plt.title("Different Destination Ethernet Address")
ax1.legend(labels, loc = "upper right")
plt.tight_layout()
plt.show()

labels = list(tcp_udp_df['type'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,6))
ax1.pie(tcp_udp_df['type'].value_counts(), autopct='%.1f%%',shadow
= True,pctdistance=1.1, labeldistance=1)
plt.title("Type")
ax1.legend(labels, loc = "upper right")
```

```
plt.tight_layout()
plt.show()

labels = list(tcp_udp_df['ihl'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,6))
ax1.pie(tcp_udp_df['ihl'].value_counts(),  autopct='%.1f%%',shadow
= True,pctdistance=1.1, labeldistance=1)
plt.title("Internet Header Length")
ax1.legend(labels, loc = "upper right")
plt.tight_layout()
plt.show()

labels=list(tcp_udp_df['ttl'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,6))
ax1.pie(tcp_udp_df['ttl'].value_counts(),  autopct='%.1f%%',shadow
= True,pctdistance=1.1, labeldistance=1)
plt.title("TTL")
ax1.legend(labels, loc = "upper right")
plt.tight_layout()
plt.show()
labels = list(tcp_udp_df['src.1'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,9))
ax1.pie(tcp_udp_df['src.1'].value_counts(),
autopct='%.1f%%',shadow = True,pctdistance=1.1, labeldistance=1)
plt.title("Source IP")
ax1.legend(labels, loc = "upper right")
plt.tight_layout()
plt.show()

labels = list(tcp_udp_df['dst.1'].value_counts().keys())
fig, ax1 = plt.subplots(figsize = (20,9))
```

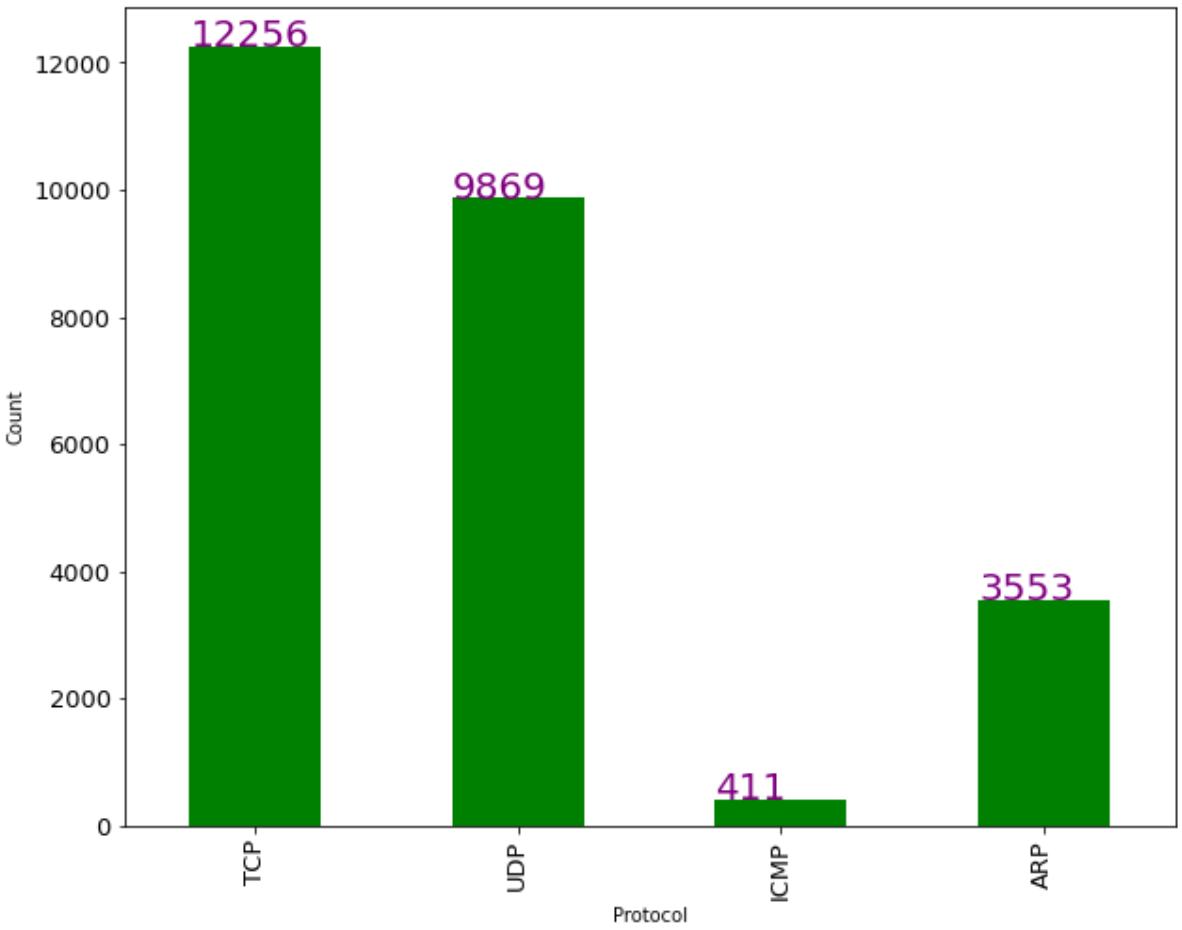
```
ax1.pie(tcp_udp_df['dst.1'].value_counts(),
autopct='%.1f%%',shadow = True,pctdistance=1.1, labeldistance=1)
plt.title("Destination IP")
ax1.legend(labels, loc = "upper right")
plt.tight_layout()
plt.show()

src_ethernet_address="01:00:5e:7f:ff:fa"
extract_data=tcp_udp_df[:][tcp_udp_df['src']==src_ethernet_addres
s]
speed = extract_data['dst'].value_counts()
index = list(extract_data['dst'].value_counts().keys())
df   =   pd.DataFrame({'Destination   Ethernet   Address':   speed},
index=index)
ax = df.plot.bar(rot=0)
speed = extract_data['src.1'].value_counts()
index = list(extract_data['src.1'].value_counts().keys())
df = pd.DataFrame({'Number Of Requests': speed}, index=index)
ax = df.plot.bar(rot=0)

speed = extract_data['dst.1'].value_counts()
index = list(extract_data['dst.1'].value_counts().keys())
df = pd.DataFrame({'Number Of Requests': speed}, index=index)
ax = df.plot.bar(rot=0)

tcp_udp_df[['src.1','sport','dst.1','dport']]
connections                                                    =
pd.DataFrame(tcp_udp_df['src.1'].map(str)+':'+tcp_udp_df['sport']
.map(str)+','+
                          tcp_udp_df['dst.1'].map(str)+':'

+tcp_udp_df['dport'].map(str),columns=['Conn-pair'])
```

```
unique_conn_pair = connections['Conn-pair'].value_counts()
unique_conn_pair.head(10)

unique_conn = len(unique_conn_pair)
total_conn =len(connections['Conn-pair'])
unique_conn_pair.index = np.arange(1,unique_conn+1)
ax            =            unique_conn_pair[0:20].plot(kind='bar',
figsize=(10,8),color="green", fontsize=13)
plt.xlabel('Connection Id')
plt.ylabel('Count of Packet Transferred')

for i in ax.patches:
    ax.text(i.get_x(),       i.get_height(),       i.get_height(),
fontsize=10,color='purple')
```

5. Result Analysis

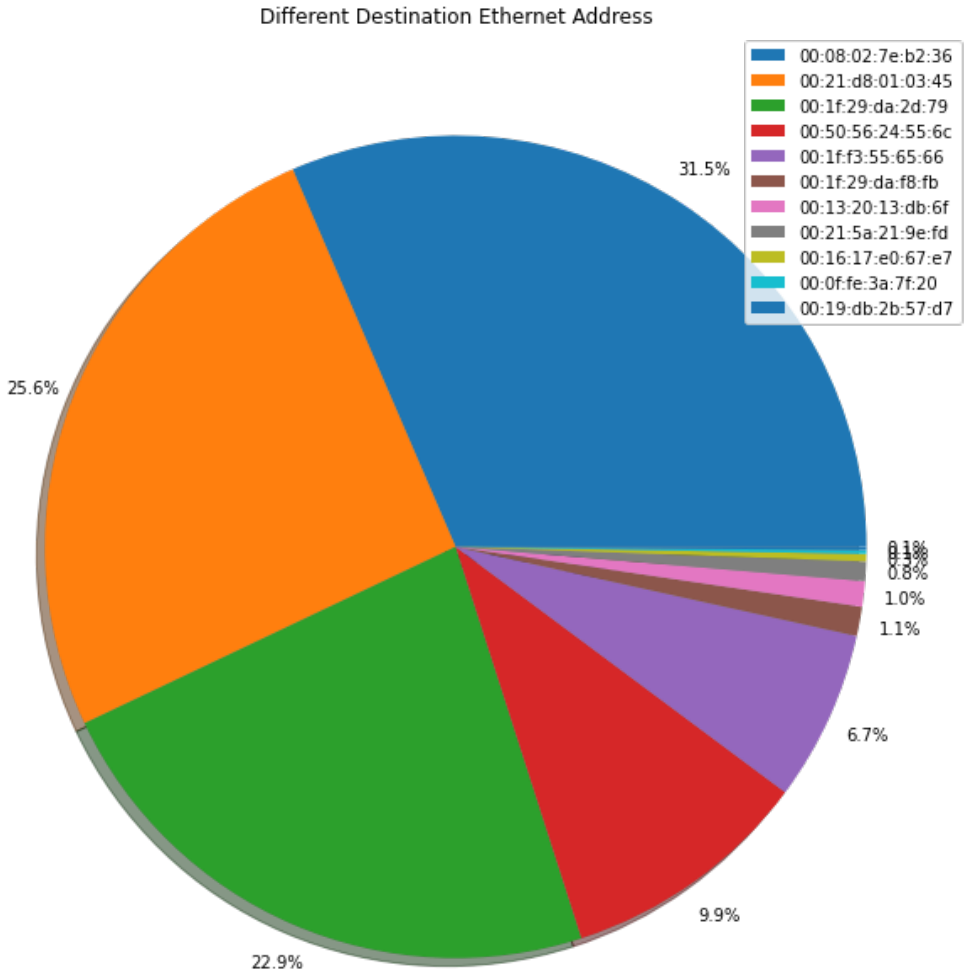
The visualization of the captured packets header information is shown in this part. Here, we have displayed few important values of IP, TCP, UDP, ARP and ICMP headers. The analysis of the captured data can be done in various ways. The followings diagram shows the some relevant information.

i.



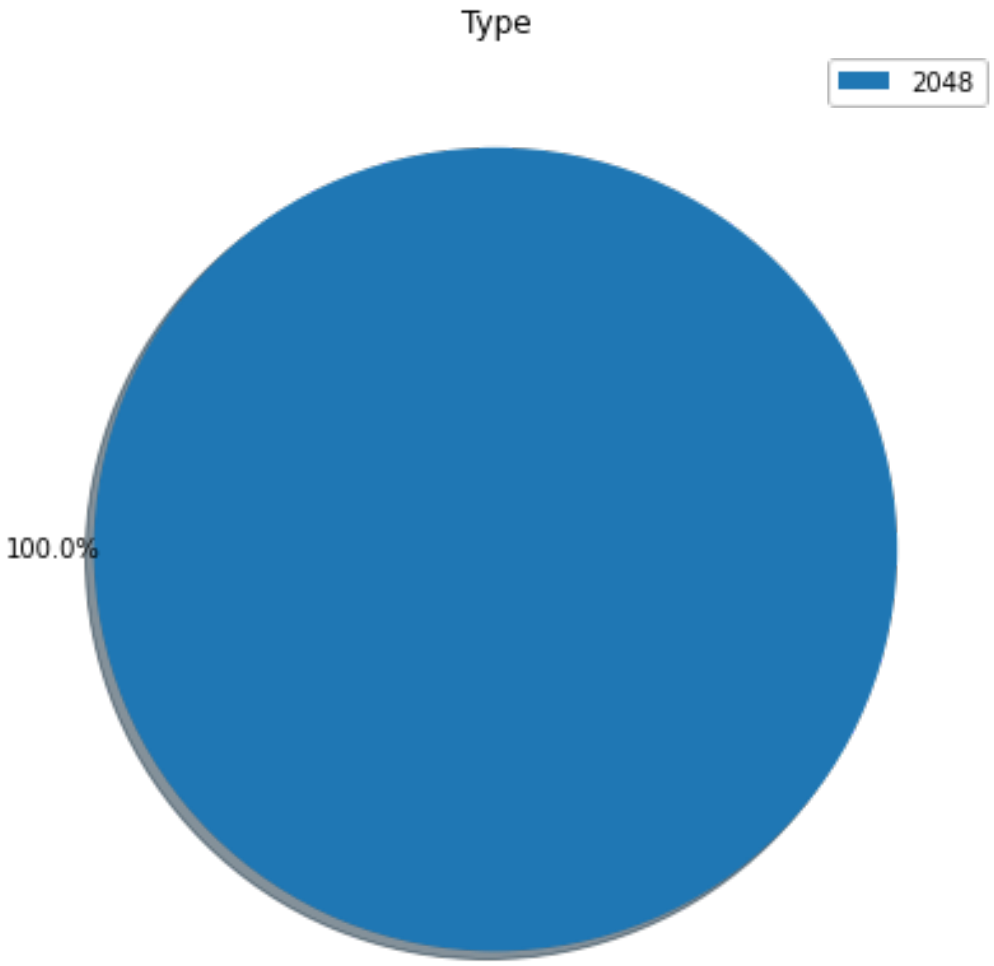
The above bar graph shows the number of packets each protocol has in the network. The test file is having total 26776 packets there are 12256 TCP packets, 9869 UDP packets, 411 ICMP packets and 3553 ARP packets. It can be clearly depicted from the chart that most of the communication was happened using TCP protocol. Therefore, TCP/UDP is the most used protocols in all captured packets.

ii.



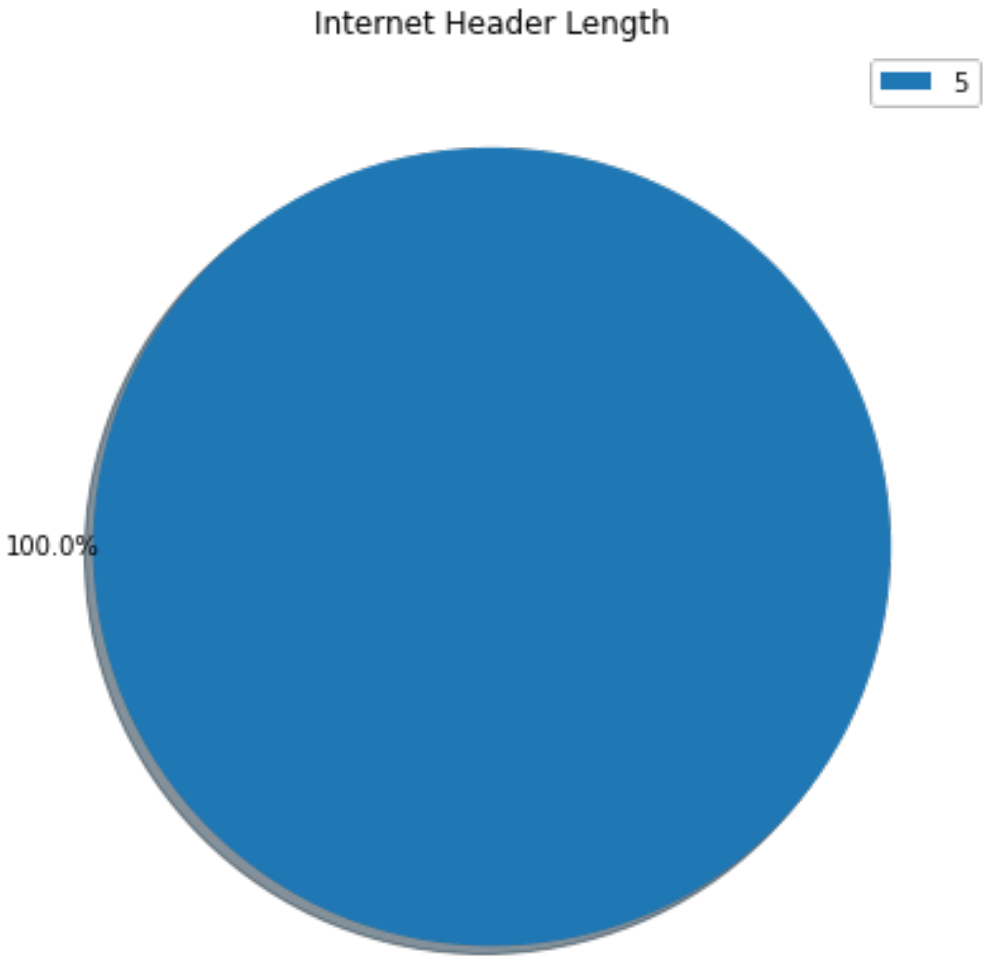
The above pie chart depicts the percentage of packets reaching the unique destination ethernet address. For example, 31.5% of packets (shown in blue) are having same destination ethernet address. The next destination address is having 21.5 % shares of packets.

iii.



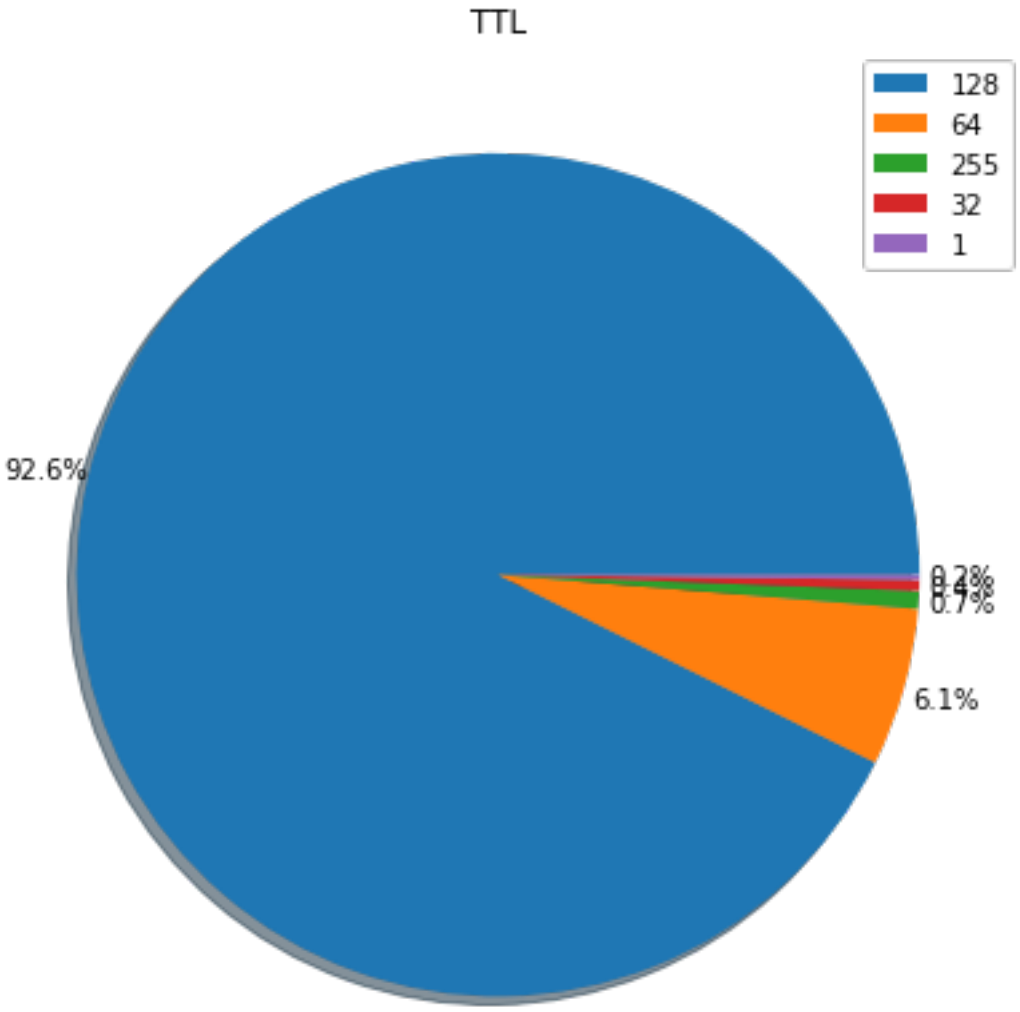
As we have used ethernet network during packet capture. Therefore, the above pie chart show that all the captured packet having its type value 2048 which is equivalent to 0x0800 in hexadecimal which is code of ethernet network.

iv.



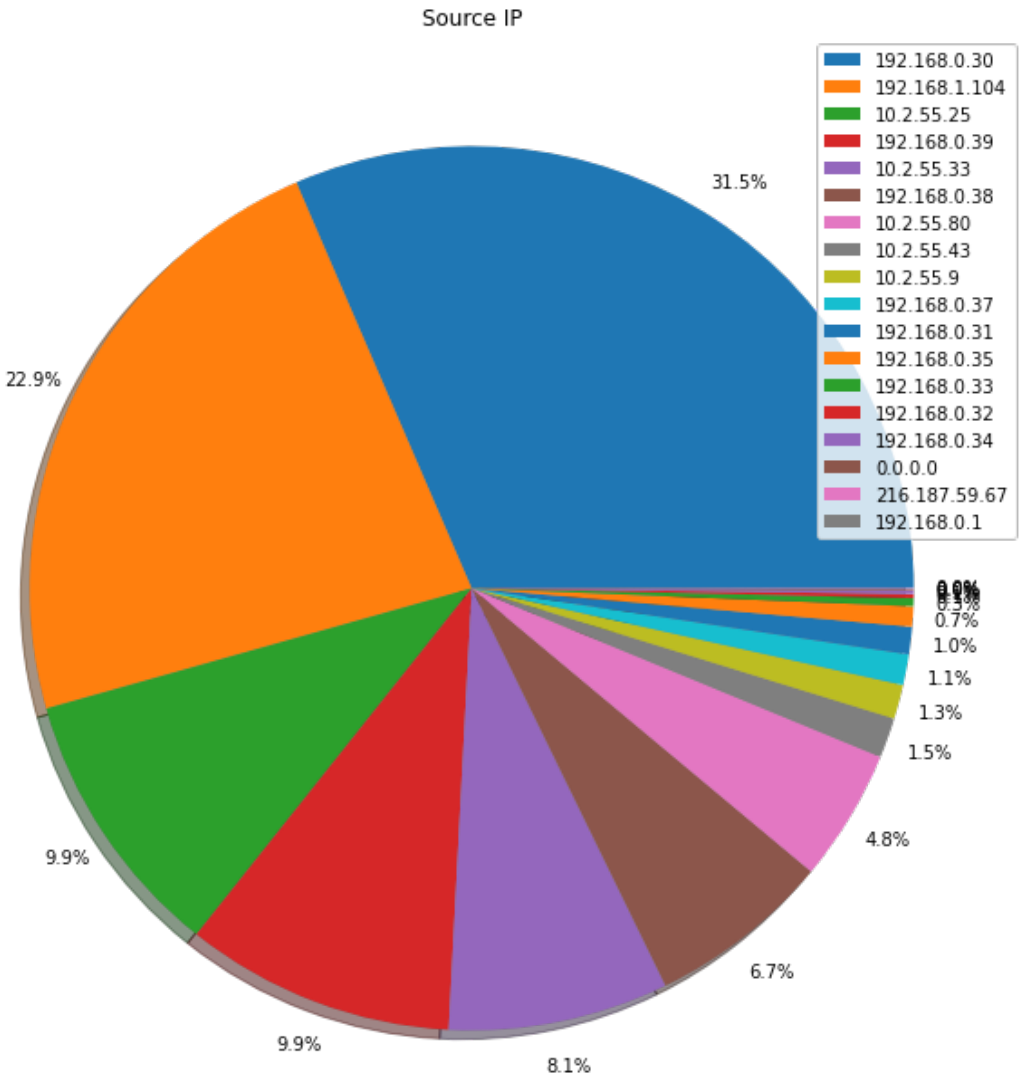
This graph shows that all our packets are using IP protocol with header length 5.

v.



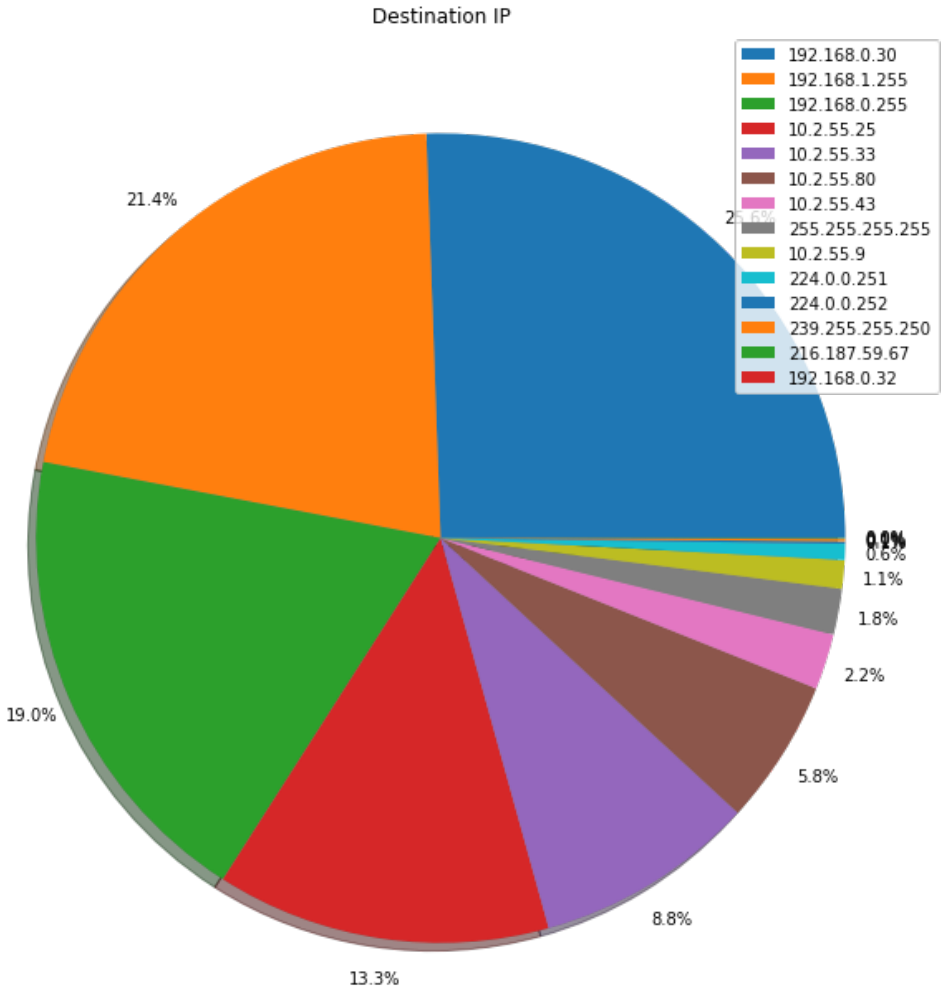
This graph shows the Time to live value for packets. The 92.6% of the packets have TTL value 128 which is the highest set value for any packet. Only a small portion (6.1%)of the packets is having TTL value 64.

vi.



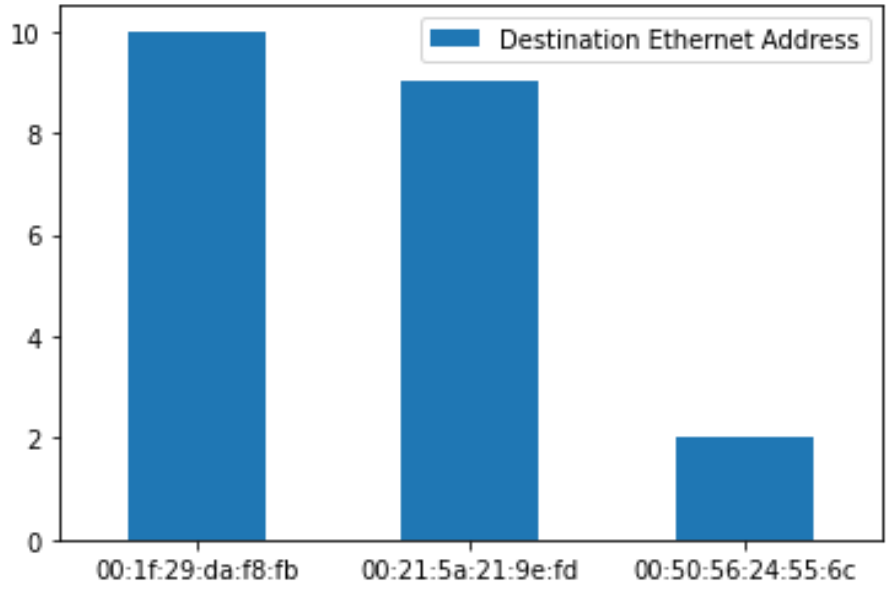
The above graph shows the packets coming from a particular source IP address. 31.5% of the packets are coming from IP address 192.168.0.30.

vii.



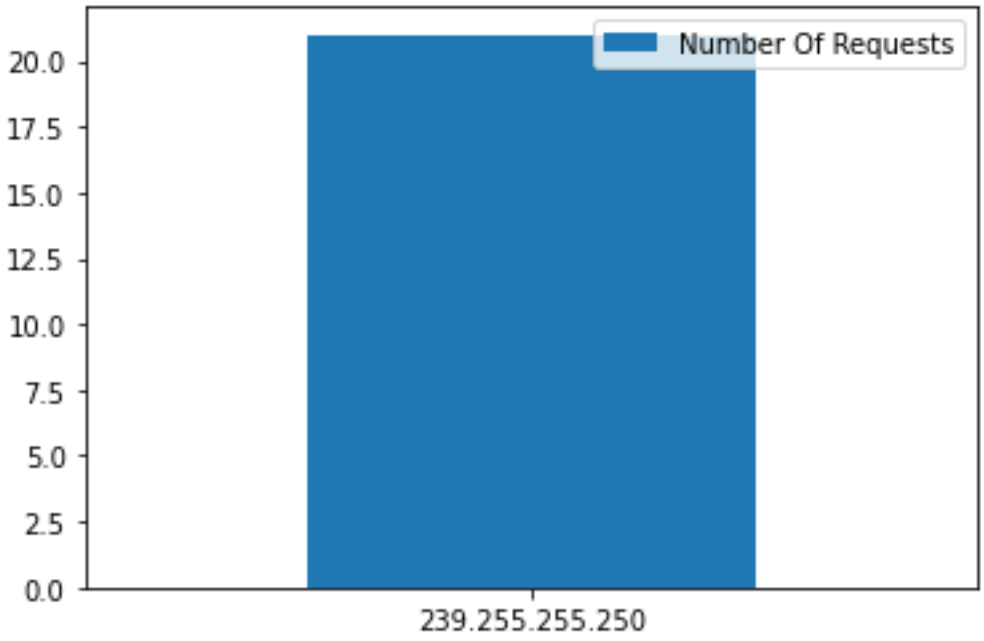
This graph shows the packets going to a particular destination IP address. 21.4% of packets are going to IP address 192.168.1.255.

viii.



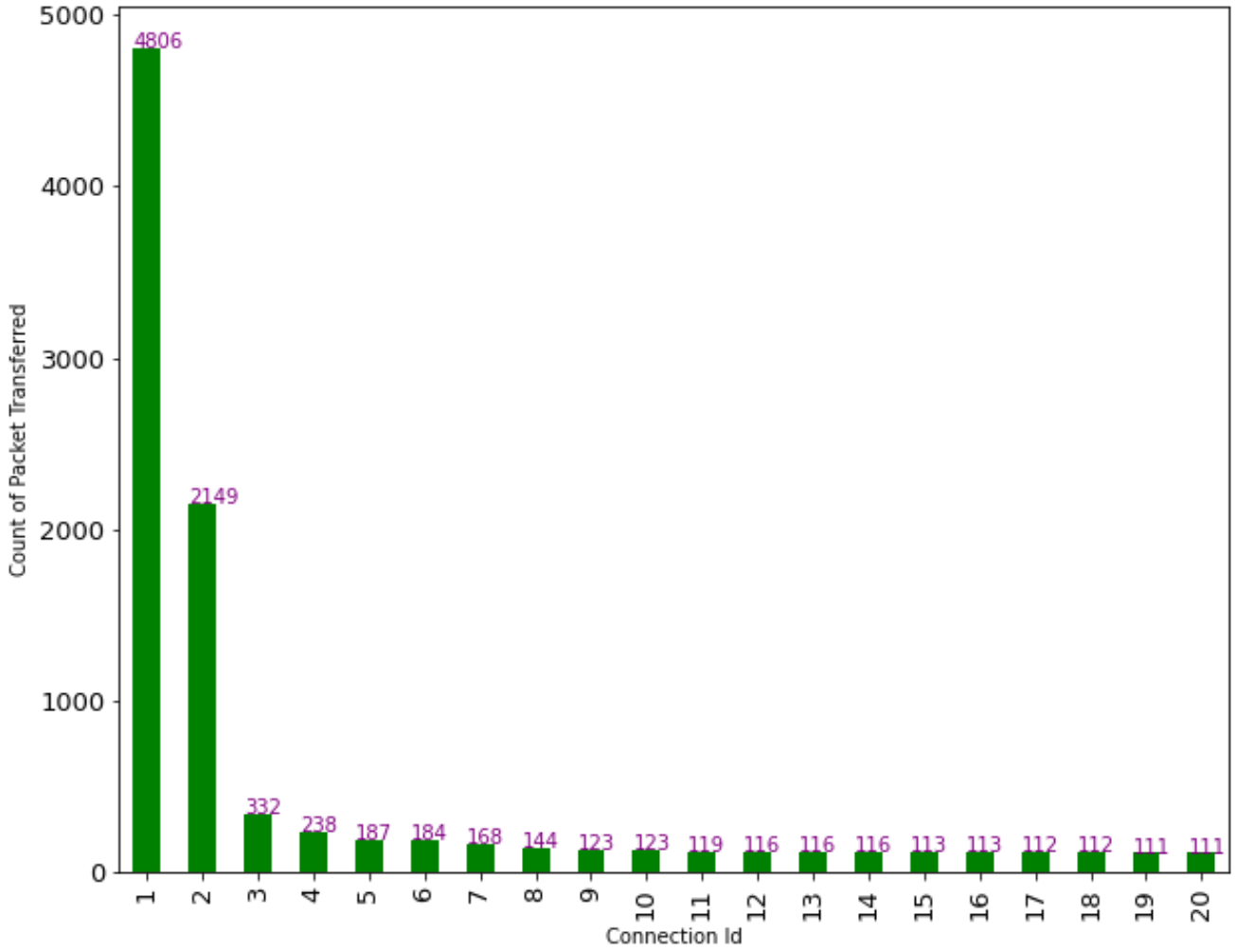
This graph shows the number of requests coming from each ethernet address.

ix.



This graph depicts the number of requests coming from a particular IP address.

x.



This graph shows only some of the socket connections of the network. There are total of 1549 connections as analyzed. Each connection has a number of packets transfer ranging from 1 to 4806. Mainly two connections have the maximum number of packets transferred between them. Connection 1 has a total of 4806 packets transmitted, and connection 2 having 2149 packets transferred.

Some Sample screenshots of the generated output

1.

```
In [2]:  1 file2 = rdpcap('03-13 los altos.pcap')
         2 print(file2)

<03-13 los altos.pcap: TCP:12256 UDP:9869 ICMP:312 Other:4339>
```

2

```
Total Number of Packets is 26776

Do you want to continue?

Press c to continue or q to quit:c

Enter the packet whose details you want to display:122
###[ Ethernet ]###
  dst       = 00:08:02:7e:b2:36
  src       = 00:21:d8:01:03:45
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 3878
     flags     = DF
     frag      = 0
     ttl       = 128
     proto     = tcp
     chksum    = 0xe9c0
     src       = 10.2.55.33
     dst       = 192.168.0.30
     \options   \
###[ TCP ]###
        sport     = microsoft_ds
        dport     = ssslog_mgr
        seq       = 1895285806
        ack       = 2246348315
        dataofs   = 5
```

```
      ttl       = 128
      proto     = tcp
      chksum    = 0xe9c0
      src       = 10.2.55.33
      dst       = 192.168.0.30
      \options   \
###[ TCP ]###
         sport     = microsoft_ds
         dport     = ssslog_mgr
         seq       = 1895285806
         ack       = 2246348315
         dataofs   = 5
         reserved  = 0
         flags     = FA
         window    = 64813
         chksum    = 0x6125
         urgptr    = 0
         options   = []
###[ Padding ]###
            load      = '\x00\x00\x00\x00\x00\x00'

Do you want to continue?

Press c to continue or q to quit:
```

3.

```
In [13]:
1 tcp_df.head()
```

Out[13]:

	dst	src	type	version	ihl	tos	len	id	flags	frag	...	dport	seq	ack	dataofs	reserved	flags	window	chksum	urgptr
0	00:50:56:24:55:6c	01:00:5e:7f:ff:fa	2048	4	5	0	157	1368		0	...	1900	None	None	None	None	None	None	50151	None
1	00:50:56:24:55:6c	ff:ff:ff:ff:ff:ff	2048	4	5	0	78	1380		0	...	137	None	None	None	None	None	None	32904	None
2	00:50:56:24:55:6c	ff:ff:ff:ff:ff:ff	2048	4	5	0	202	1394		0	...	138	None	None	None	None	None	None	61937	None
3	00:50:56:24:55:6c	ff:ff:ff:ff:ff:ff	2048	4	5	0	78	1395		0	...	137	None	None	None	None	None	None	23411	None
4	00:50:56:24:55:6c	ff:ff:ff:ff:ff:ff	2048	4	5	0	78	1403		0	...	137	None	None	None	None	None	None	23411	None

5 rows × 28 columns

4.

```
In [10]:
1 arp_df = arp_df.reset_index()
2 arp_df = arp_df.drop(columns="index")
3 arp_df.head()
```

Out[10]:

	hwtype	ptype	hwlen	plen	op	hwsrc	psrc	hwdst	pdst
0	1	2048	6	4	1	00:1f:29:da:2d:79	192.168.1.104	00:00:00:00:00:00	192.168.1.1
1	1	2048	6	4	1	00:1f:29:da:2d:79	192.168.1.104	00:00:00:00:00:00	192.168.1.1
2	1	2048	6	4	1	00:1f:29:da:2d:79	192.168.1.104	00:00:00:00:00:00	192.168.1.1
3	1	2048	6	4	1	00:1f:29:da:2d:79	192.168.1.104	00:00:00:00:00:00	192.168.1.1
4	1	2048	6	4	1	00:1f:29:da:2d:79	192.168.1.104	00:00:00:00:00:00	192.168.1.1

5.

```
In [15]:
1  #Fetch ICMP Data
2  field = icmp_fields[0:3]
3  icmp_df = pd.DataFrame(columns=field)
4  print("Reading ARP Packets.....")
5  for packet in file_name[ICMP]:
6      field_values = []
7      field_values.append(packet.type)
8      field_values.append(packet.code)
9      field_values.append(packet.chksum)
10
11     df_append = pd.DataFrame([field_values], columns=field)
12     icmp_df = pd.concat([icmp_df, df_append], axis=0)
13 print("Completed")
14 icmp_df = icmp_df.reset_index()
15 icmp_df = icmp_df.drop(columns="index")
16 icmp_df.head()
```

```
Reading ARP Packets.....
Completed
```

Out[15]:

	type	code	chksum
0	2048	0	34537
1	2048	0	11053
2	2048	0	34535
3	2048	0	11051
4	2048	0	34430

6.

```
In [19]: tcp_udp_df[['src.1','sport','dst.1','dport']]
```

click to scroll output; double click to hide

Out[19]:

	src.1	sport	dst.1	dport
0	192.168.0.39	55793.0	239.255.255.250	1900.0
1	192.168.0.39	137.0	192.168.0.255	137.0
2	192.168.0.39	138.0	192.168.0.255	138.0
3	192.168.0.39	137.0	192.168.0.255	137.0
4	192.168.0.39	137.0	192.168.0.255	137.0
...	...	...	...	...
22531	192.168.0.38	55024.0	192.168.0.255	137.0
22532	192.168.0.39	137.0	192.168.0.255	137.0
22533	192.168.0.39	137.0	192.168.0.255	137.0
22534	192.168.0.39	137.0	192.168.0.255	137.0
22535	192.168.0.39	138.0	192.168.0.255	138.0

22536 rows × 4 columns

7.

```
In [20]: connections = pd.DataFrame(tcp_udp_df['src.1'].map(str)+':'+tcp_udp_df['sport'].map(str)+','+
                                   tcp_udp_df['dst.1'].map(str)+':'
                                   +tcp_udp_df['dport'].map(str),columns=['Conn-pair'])
         unique_conn_pair = connections['Conn-pair'].value_counts()
         unique_conn_pair.head(10)
```

```
Out[20]: 192.168.1.104:137.0,192.168.1.255:137.0    4806
         192.168.0.39:137.0,192.168.0.255:137.0     2149
         192.168.1.104:68.0,255.255.255.255:67.0     332
         10.2.55.80:80.0,192.168.0.30:1614.0         238
         192.168.0.30:1614.0,10.2.55.80:80.0         187
         192.168.0.37:137.0,192.168.0.255:137.0      184
         192.168.0.31:137.0,192.168.0.255:137.0      168
         192.168.0.38:5353.0,224.0.0.251:5353.0      144
         10.2.55.33:nan,192.168.0.30:nan             123
         192.168.0.30:1324.0,10.2.55.25:135.0        123
         Name: Conn-pair, dtype: int64
```

Conclusion

We have analyzed a given PCAP file having different network packets captured in C array format. To read the content of these packets, we have used the scpay package in python language. The packets are analyzed to determine all the fields like source address, destination address, source port number, destination port number, and protocols used in a different layer (such as IPv4 protocol, TCP protocol). Finally, all this information is displayed using various graphs and pie charts to summarize the network packets.

References

[1]https://www.thousandeyes.com/learning/glossary/packet-capture

[2] http://www.exa.unicen.edu.ar/catedras/comdat1/material/TP1-Ejercicio5-ingles.

[3] https://en.wikipedia.org/wiki/Address_Resolution_Protocol.

[4] Veteikis, M. and Moriarty, M., BreakingPoint Systems Inc, 2013. *Packet capture for error tracking*. U.S. Patent Application 13/529,970.

[5] McCanne, S. and Jacobson, V., 1993, January. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX winter* (Vol. 46).

[6] Deri, L., 2004, September. Improving passive packet capture: Beyond device polling. In *Proceedings of SANE* (Vol. 2004, pp. 85-93).