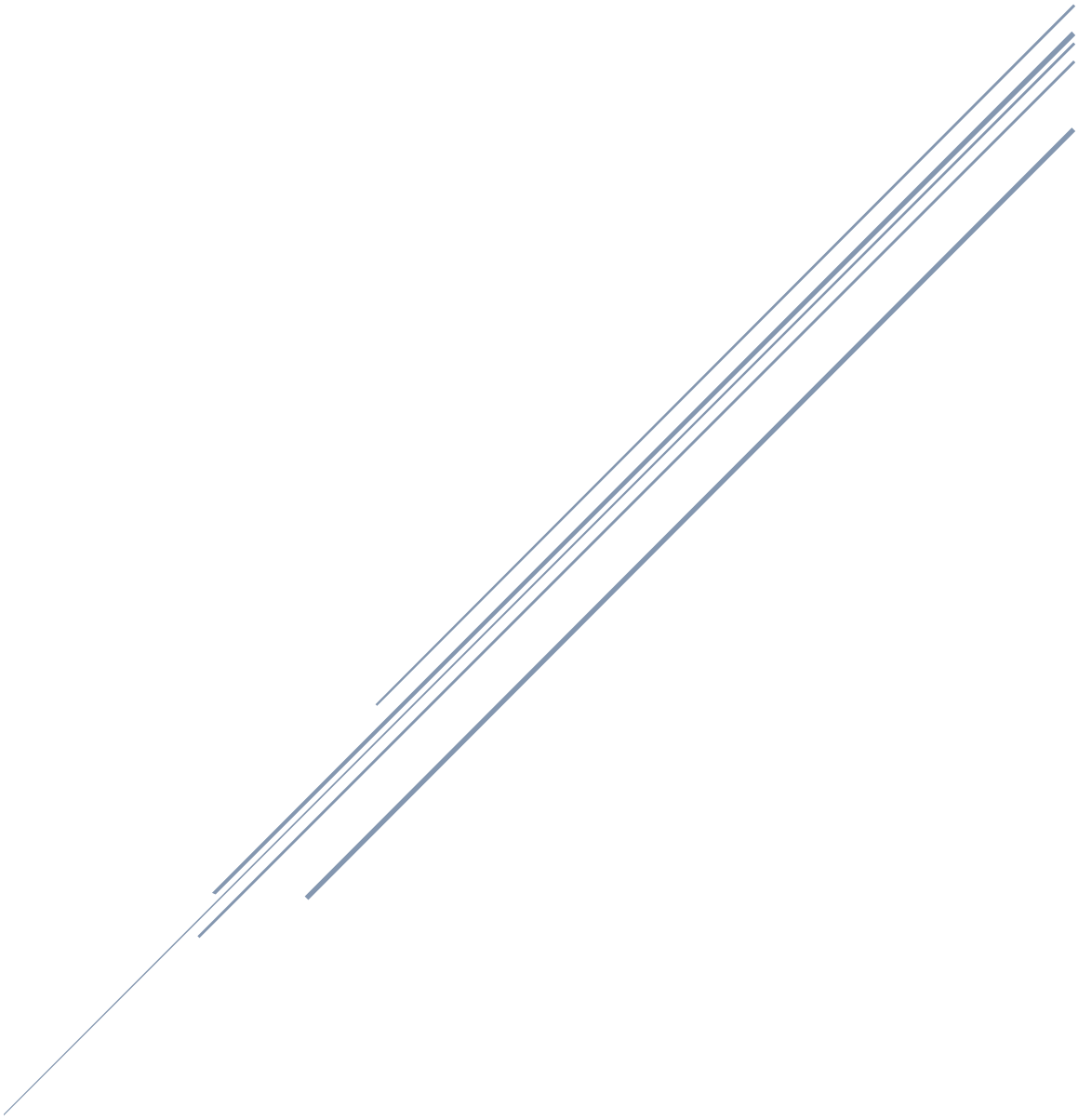
# COMPUTER SYSTEMS AND NETWORKS COURSE-WORK-1

Made BY: Begad Hatem Diab ID# 202101453



# Table of Contents

## *Introduction:*

This report talks about how Filbert Company, an IT consulting company that specializes in network topology insights for networking security, developed an application solution. The deep packet inspection software, network modulators, English to network translators, and other tools will all be part of the application tool that will be created using the Python programming language. The main objective of the tool is to offer the clients of the Filbert Company thorough and effective network security solutions that are tailored to their individual requirements. The application tool's development process, including each solution's implementation and integration into the overall tool architecture, will be covered in depth in this report. The paper will also go over the advantages of the tool and how it may help clients of Filbert Company's clients increase network security.

## *Part 1: Libraries Used*

- **Prettytable**: A Python package called the prettytable library enables programmers to produce aesthetically pleasing ASCII tables in their programming. In order to build tables that are simple to read and comprehend, it offers a variety of formatting choices, including alignment, borders, and colour.
- **Socket**: A built-in Python module called the socket library makes it possible for programmers to construct and use network sockets. It offers a collection of classes and methods for generating, modifying, and delivering data across TCP/IP networks. When creating network applications that need to communicate with many devices, the socket library is crucial.
- **Ipaddress**: A Python module called the ipaddress library offers classes and methods for working with IP addresses and networks. It offers a number of capabilities that make working with IP addresses and networks in Python simpler, including IP address parsing, subnetting, and network address manipulation.
- **Time**: A built-in Python module called the time library offers a number of methods for dealing with time-related issues. It has tools for managing timezones, formatting dates and times, and more. It also has tools for measuring time intervals.
- **scapy.all**: The Python module scapy.all library offers a potent interactive packet manipulation tool and library. Python code may be used by developers to produce, modify, send, and receive network packets. Scapy.all has several sophisticated functions, including the ability to sniff packets from a network interface, analyse and decode packets, and create custom packets. For testing and debugging network protocols and security, network engineers and security experts use it frequently.

# Part 2: Functions Used

## 1- def Ethernet_packet(packet)

This "ethernet packet" function extracts the Ethernet header data from a packet object as an input argument.

By confirming that the packet contains an Ethernet layer, it first determines if it is a legitimate Ethernet packet. If not, it returns after printing a notice stating that the packet is invalid.

The function retrieves the source and destination MAC addresses, the Ethernet protocol type, and the packet size if the packet is valid. The next step is to identify whether the packet is broadcast, multicast, or unicast.

The function determines the broadcast and network addresses if the packet is broadcast. The function sets the broadcast address to the destination MAC address if the packet is multicast.

Finally, the function prints the packet information, including the packet type, source and destination MAC addresses, Ethernet protocol type, packet size, broadcast address, and network ID. The function returns a list of the source and destination MAC addresses.

## 2- def ip_packet(packet)

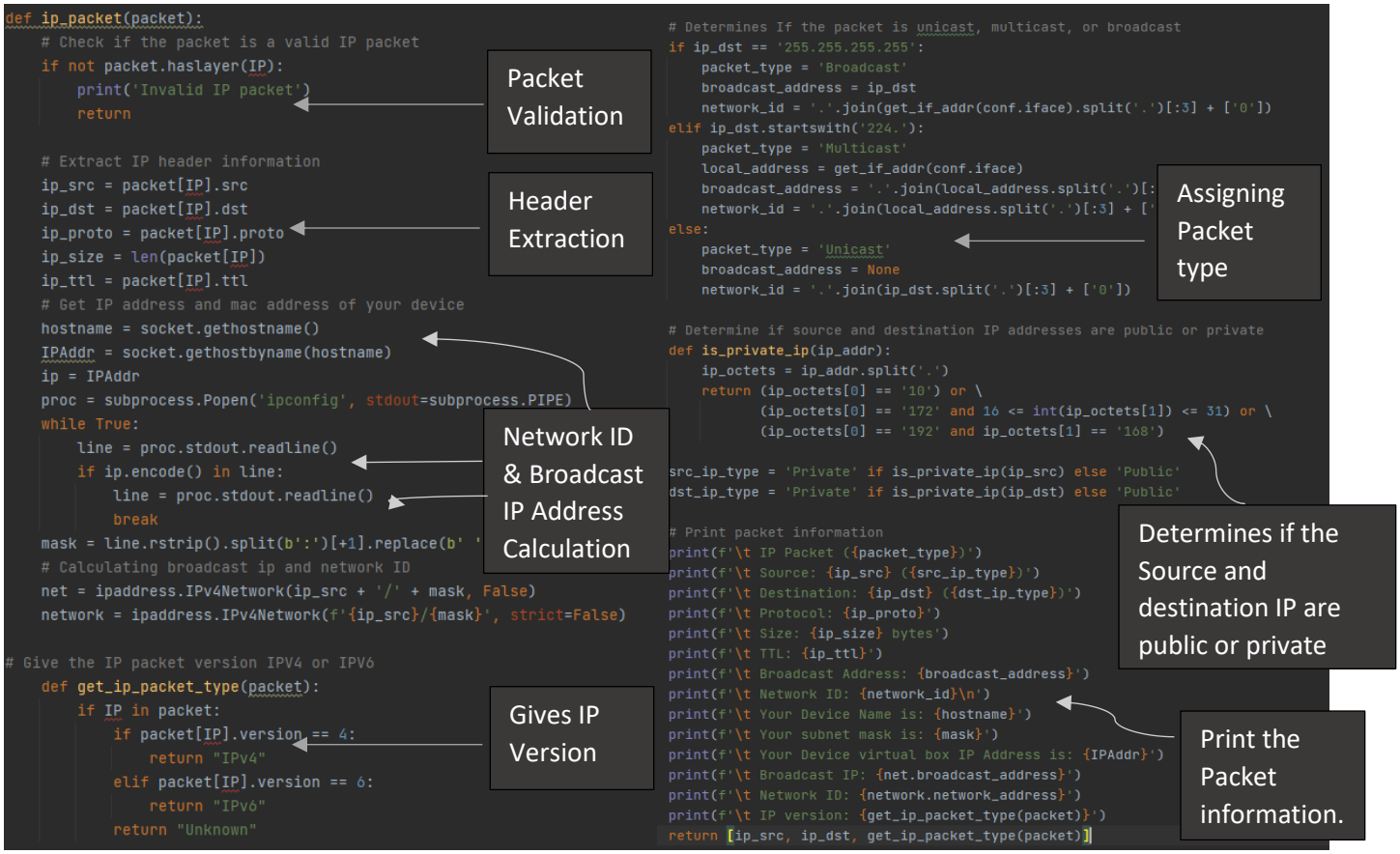
The ip_packet function takes a packet as input and extracts various pieces of information from it related to the IP Internet Protocol layer of the network stack. It first checks if the packet is a valid IP packet and returns an error message if it is not.

If the packet is valid, the function extracts the following information from the IP header of the packet: source IP address, destination IP address, protocol used TCP or UDP, size of the IP packet in bytes, and Time-To-Live (TTL) value.

Then, the function gets the IP address and MAC address of the local device and extracts the subnet mask of the device's network interface. It also calculates the broadcast IP and network ID for the packet's source IP address using the subnet mask.

The function then determines the type of the IP packet IPv4, IPv6, or unknown and whether the packet is unicast, multicast, or broadcast. For broadcast and multicast packets, it also calculates the broadcast address and network ID of the local device's network interface.

Finally, the function determines if the source and destination IP addresses are public or private, prints out all the extracted information, and returns a list containing the source IP address, destination IP address, and IP packet type.



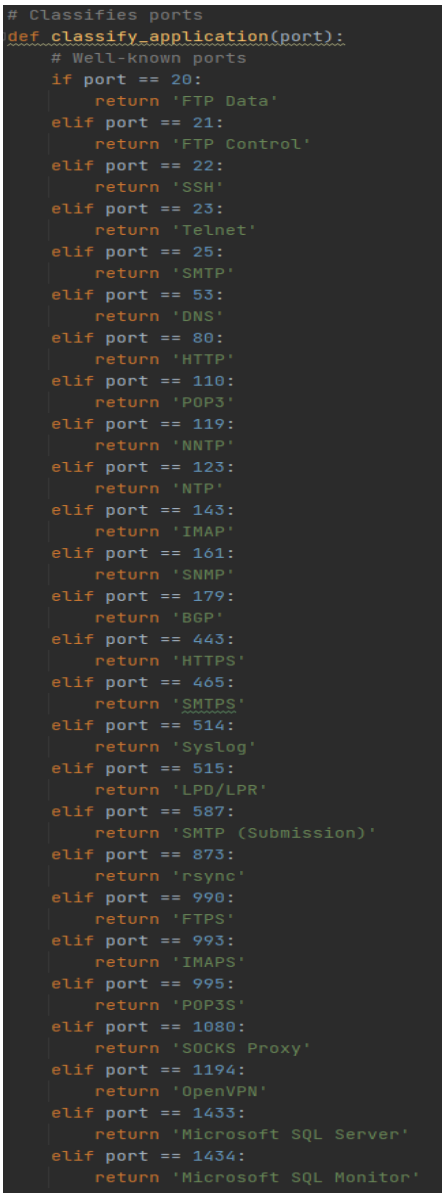
## 3- def classify_application(port)

The "classify application" function accepts an integer "port" as input and outputs a string describing the application or protocol connected to that port number.

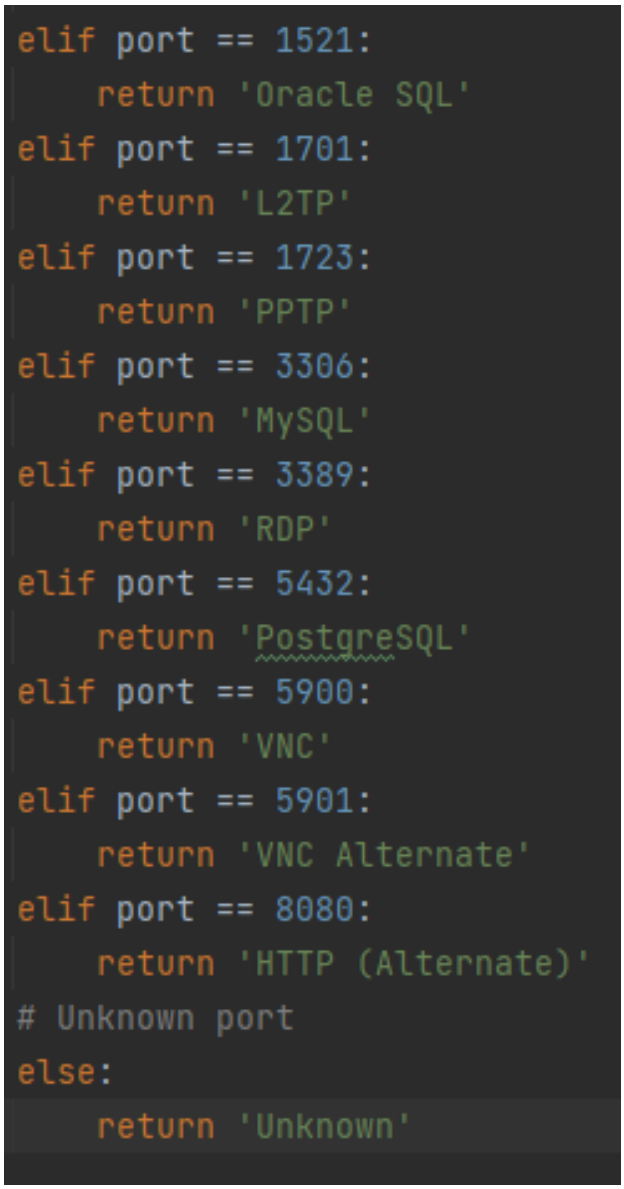
Before doing anything else, the function determines whether the input port number corresponds to any of the well-known port numbers, such as port 80 for HTTP or port 443 for HTTPS. The method returns the appropriate application or protocol name if the port number matches a well-known port.

The function returns "Unknown" if any of the well-known port numbers in the provided port number do not match.

In general, this function may be used to determine the programme or protocol connected to a specific port number, which can be helpful for network security and troubleshooting.



```
# Classifies ports
def classify_application(port):
    # Well-known ports
    if port == 20:
        return 'FTP Data'
    elif port == 21:
        return 'FTP Control'
    elif port == 22:
        return 'SSH'
    elif port == 23:
        return 'Telnet'
    elif port == 25:
        return 'SMTP'
    elif port == 53:
        return 'DNS'
    elif port == 80:
        return 'HTTP'
    elif port == 110:
        return 'POP3'
    elif port == 119:
        return 'NNTP'
    elif port == 123:
        return 'NTP'
    elif port == 143:
        return 'IMAP'
    elif port == 161:
        return 'SNMP'
    elif port == 179:
        return 'BGP'
    elif port == 443:
        return 'HTTPS'
    elif port == 465:
        return 'SMTPS'
    elif port == 514:
        return 'Syslog'
    elif port == 515:
        return 'LPD/LPR'
    elif port == 587:
        return 'SMTP (Submission)'
    elif port == 873:
        return 'rsync'
    elif port == 990:
        return 'FTPS'
    elif port == 993:
        return 'IMAPS'
    elif port == 995:
        return 'POP3S'
    elif port == 1080:
        return 'SOCKS Proxy'
    elif port == 1194:
        return 'OpenVPN'
    elif port == 1433:
        return 'Microsoft SQL Server'
    elif port == 1434:
        return 'Microsoft SQL Monitor'
```



```
elif port == 1521:
    return 'Oracle SQL'
elif port == 1701:
    return 'L2TP'
elif port == 1723:
    return 'PPTP'
elif port == 3306:
    return 'MySQL'
elif port == 3389:
    return 'RDP'
elif port == 5432:
    return 'PostgreSQL'
elif port == 5900:
    return 'VNC'
elif port == 5901:
    return 'VNC Alternate'
elif port == 8080:
    return 'HTTP (Alternate)'
# Unknown port
else:
    return 'Unknown'
```

## 4- def transport_layer_packet(packet)

The transport_layer_packet function takes a network packet as input and performs the following tasks:

Verify that the transport layer in the packet is active (either TCP or UDP). If the packet's transport layer is invalid, it outputs an error message and exits.

Extract the packet's source and destination port numbers.

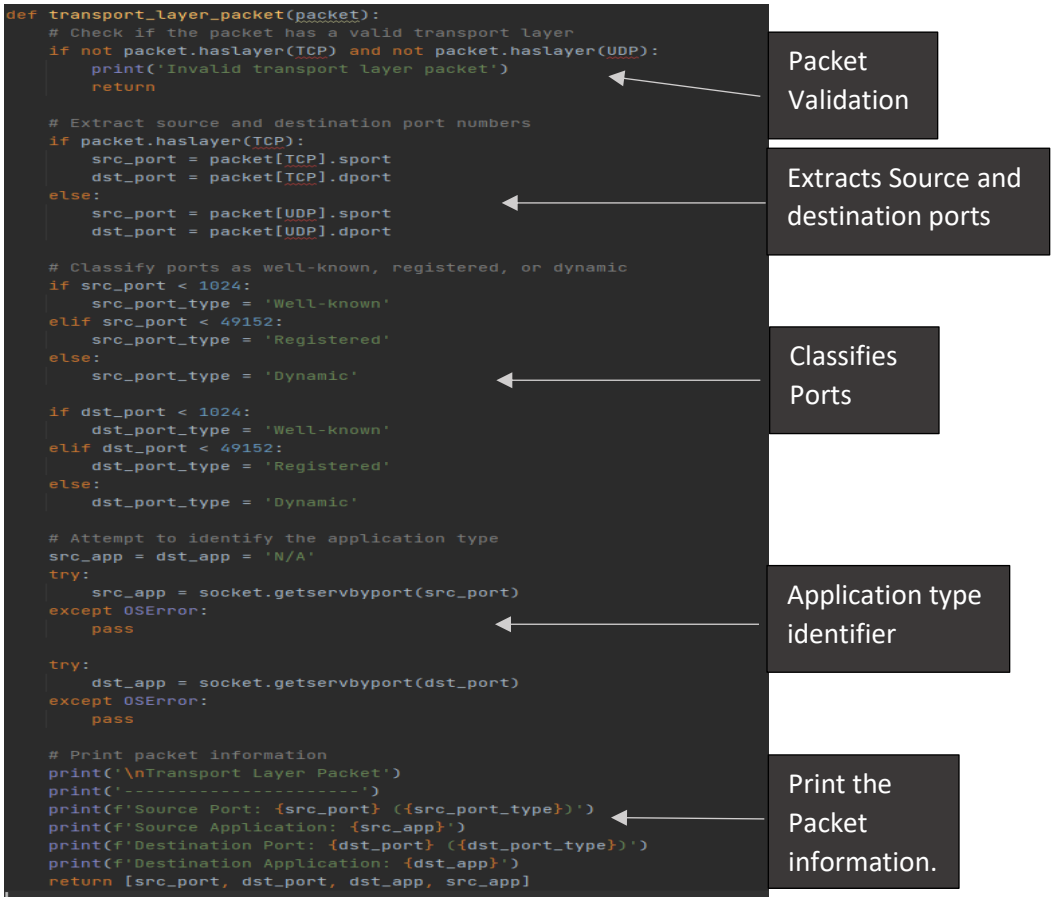
Depending on their values, categorise the source and destination port numbers as either well-known, registered, or dynamic.

Use the socket library's getservbyport method to determine the source and destination application types. This function provides the name of the service connected to the specified port number.

In addition to the source and destination port numbers, their kinds, and the known source and destination application types, print packet details.

The source port number, destination port number, source application type, and destination application type should all be listed in the response.

Overall, by identifying a transport layer packet's source and destination ports, categorising the ports based on their values, and attempting to determine the related application types, the transport layer packet function performs a rudimentary analysis of the transport layer packet.



## Part 3: Printing the outputs

This code continuously sniffs the network for packets, analyzes each packet in a step-by-step manner, and prints out information about the packet at each layer of the OSI model.

The code starts by using the sniff() function from the Scapy library to capture a single packet from the network. It then passes this packet to a series of functions that analyze it at each layer of the OSI model.

The first function called is ethernet_packet(), which analyzes the Ethernet frame and prints out information such as the source and destination MAC addresses.

Next, the code waits for 5 seconds and then calls the ip_packet() function to analyze the IP packet and print out information such as the source and destination IP addresses.

After another 5 second delay, the code calls the transport_layer_packet() function to classify the source and destination ports as well-known, registered, or dynamic, and attempt to identify the application type.

The classify_application() function is then called to further classify the application based on its network behavior.

Finally, the code waits for another 5 seconds and prints out a table that summarizes the information collected at each layer of the OSI model for the captured packet.

The code then waits for 1 minute and repeats the process of sniffing and analyzing packets indefinitely.

```
while True:
    print("Ethernet packets and analysis:\n")
    pkt = sniff(count=1) # Sniffs 1 time for each packet
    x = ethernet_packet(pkt[0]) # Prints the ethernet analysis
    print("------------------------------------------------------------")
    print("Waiting for 5 sec... to print IP analysis \n")
    time.sleep(5) # Delays for 5 seconds
    print("IP packets and analysis: \n")
    y = ip_packet(pkt[0]) # Prints the IP analysis
    print("------------------------------------------------------------")
    print("Waiting for 5 sec... to print Port analysis \n")
    time.sleep(5)
    z = transport_layer_packet(pkt[0]) # Prints Port classifications
    classify_application(pkt[0]) # Prints Application classification
    print("------------------------------------------------------------")
    print("Waiting for 5 sec... to print OSI Model \n")
    time.sleep(5)
    # OSI Model
    table = PrettyTable(['Layer', 'PDU', 'Address Type', 'Source', 'Destination', 'Protocols'])
    table.add_row(['Application', 'Data', 'Port Number', f'Source Port: {z[0]}', f'Destination port number: {z[1]}', f'Protocols: {z[3]} - {z[2]}'])
    table.add_row(['Transport', 'Segment', 'Port Number', f'Source Port: {z[0]}', f'Destination port number: {z[1]}', f'Protocols: {z[3]} - {z[2]}'])
    table.add_row(['Network', 'Packet', 'IP Address', f'Sender IP: {y[0]}', f'Receiver IP {y[1]}', f'IP-Version: {y[2]}'])
    table.add_row(['Data Link', 'Frame', 'MAC Address', f'Sender MAC: {x[0]}', f'Receiver MAC: {x[1]}', 'Protocols: Ethernet 2'])
    table.add_row(['Physical', 'Bit', 'N/A', 'N/A', 'N/A', 'N/A'])
    print(table)
    print("Waiting for 1 minute to get next packet info... \n")
    print('\n')
    time.sleep(60)
```

Sniffs The packets and outputs only 1 packet type information at a time and loops it so that it prints every 1 minute.

## Part 4: OSI Model

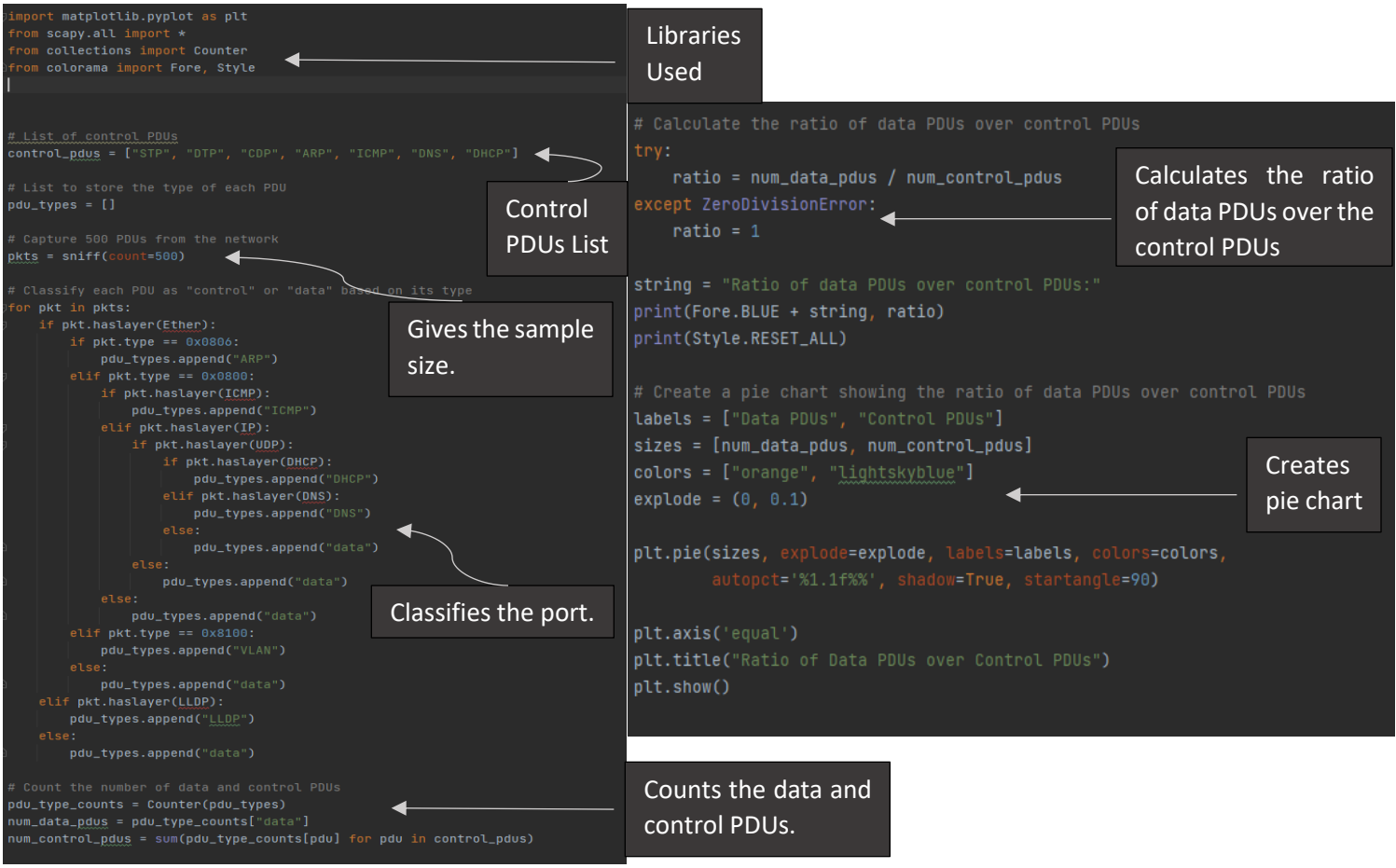
Using the PrettyTable library, this code constructs a table and adds rows to it. Six columns in the table are titled "Layer," "PDU," "Address Type," "Source," "Destination," and "Protocols." A layer's PDU, address type, source, destination, and protocols are all listed in each row, which corresponds to a layer in the OSI model. The variables x, y, and z, which were acquired through the examination of Ethernet, IP, and transport layer packets, respectively, provide the data. The print feature is then used to print the table to the console.

```
# OSI Model
table = PrettyTable(['Layer', 'PDU', 'Address Type', 'Source', 'Destination', 'Protocols'])
table.add_row(['Application', 'Data', 'Port Number', f'Source Port: {z[0]}', f'Destination port number: {z[1]}', f'Protocols: {z[3]} - {z[2]}'])
table.add_row(['Transport', 'Segment', 'Port Number', f'Source Port: {z[0]}', f'Destination port number: {z[1]}', f'Protocols: {z[3]} - {z[2]}'])
table.add_row(['Network', 'Packet', 'IP Address', f'Sender IP: {y[0]}', f'Receiver IP {y[1]}', f'IP-Version: {y[2]}'])
table.add_row(['Data Link', 'Frame', 'MAC Address', f'Sender MAC: {x[0]}', f'Receiver MAC: {x[1]}', 'Protocols: Ethernet 2'])
table.add_row(['Physical', 'Bit', 'N/A', 'N/A', 'N/A', 'N/A'])
print(table)
```

Prints the OSI model in details for each layer.

## Part 5: Statistical Model

This program takes 500 packets from a network using the Scapy library, categorises them as "control" or "data" depending on their nature, and then determines the proportion of data packets to control packets. A pie chart is then produced to show this ratio. ARP, ICMP, DNS, DHCP, STP, DTP, and CDP are the several categories of control packets that are considered. The control packets are divided into the several categories listed above, and the pie chart displays the percentage of both data packets and control packets. The Colorama library is also used by the code to output a message to the console containing the computed ratio. The pie chart was made using the Matplotlib software.



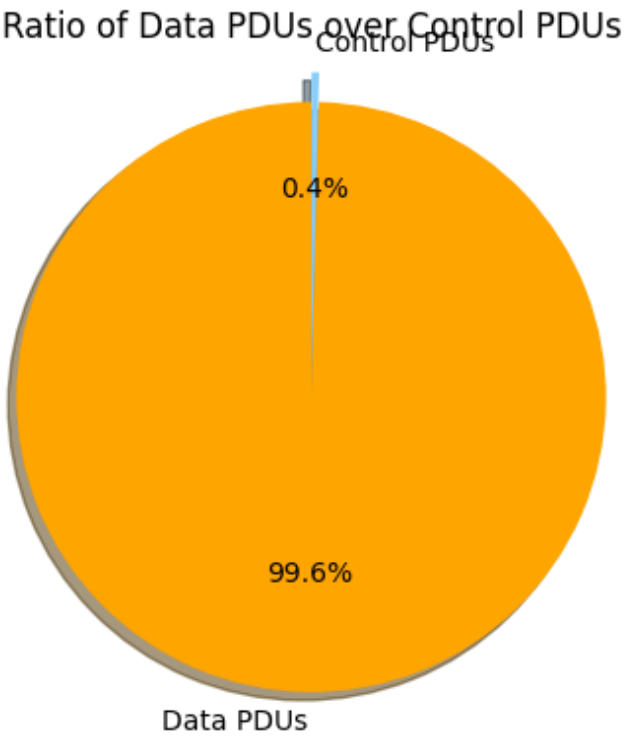



Figure 1

In the figure above is an example of the statistical model out put with the ratio in Figure 2 below

```
Ratio of data PDUs over control PDUs: 249.0
```

Figure 2

## *Part 6: Conclusion*

In conclusion, we discussed a Python script that sniffs network traffic and analyzes the captured packets using different protocols and layers of the OSI model. The script uses several Python libraries such as Scapy and PrettyTable to capture and display information about Ethernet, IP, transport layer, and application layer packets.

The script starts by sniffing the network for one packet and then analyzing it using the ethernet_packet, ip_packet, transport_layer_packet, and classify_application functions. The ethernet_packet function extracts information about the Ethernet frame, such as source and destination MAC addresses. The ip_packet function extracts information about the IP packet, such as source and destination IP addresses and IP version. The transport_layer_packet function extracts information about the transport layer packet, such as source and destination port numbers and port classification. The classify_application function attempts to identify the application type based on the destination port number.

The script then uses the PrettyTable library to create a table that summarizes the information collected from the different layers of the OSI model. The table includes information about the layer, PDU, address type, source and destination addresses, and protocols. Finally, the script waits for one minute before capturing and analyzing the next packet.

Overall, this Python script provides a useful tool for network administrators and security analysts at Filbert Company to monitor and analyze network traffic and identify potential security threats or performance issues.

## References:


- C. Trueman, S. Henry-Stocker, J. Burke, J. Gold, M. Cooney, T. Greene, A. Patrizio, A. Ghoshal, and A. Bednarz, “Welcome to network World.com,” *Network World*. [Online]. Available: https://www.networkworld.com/. [Accessed: 08-Apr-2023].

- “A computer science portal for geeks,” *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/. [Accessed: 08-Apr-2023].

- K. Greenberg, M. Clarke, M. Crouse, K. Kimachia, D. Partida, J. Wallen, B. Stone, E. Eckel, C. Pernet, and T. R. Academy, “Home,” *TechRepublic*, 07-Apr-2023. [Online]. Available: https://www.techrepublic.com/. [Accessed: 08-Apr-2023].

- “Where developers learn, share, & build careers,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/. [Accessed: 08-Apr-2023].