MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

TOOL FOR CAPTURING THE TRAFFIC BASED ON THE TCP HEADER FLAGS. (URGENT, PUSH, SYN, ACK, FIN)

A MINI PROJECT REPORT

SUBMITTED BY:

AYUSH RAO- 190905001

SUHAS BHAT – 190905026

NAMAN ARYAN- 190905080

In partial fulfillment for the award of the degree of

B-Tech in Computer Science Engineering

In

COMPUTER NETWORKS LAB

Department of Computer Science & Engineering

BONAFIDE CERTIFICATE

Certified that this project report "**Tool for capturing the traffic based on the TCP header flags. (URG, PSH, SYN, ACK, FIN)**" is the bonafide work of

“AYUSH RAO – 190905001

SUHAS BHAT – 190905026

NAMAN ARYAN -190905080”

Who carried out the mini project work under my supervision.

Dr. Ashalatha Nayak	Dr. Krishnamoorthi M
HOD, CSE	Professor, CSE

Submitted to the Viva-voce Examination held on

____

EXAMINER 1 EXAMINER 2

ABSTRACT

Flags are an essential part of the TCP packet. They signify the role/content of the packet in the sequence of the packet. Our tool is designed to read through TCP packets, understand the flag's language in those packets, and provide the relevant information on flags to the user.

Our motive through the tool is to appreciate the importance of flags and their role in Transport layer communication between hosts. We intend to provide the information on their function in the packet and sort through the collection of packets to focus only on packets with a specific flag.

Hence, our tool provides two functionalities; firstly, to analyze the TCP packets based on their flags, and secondly, to filter the packets with respect to a specific flag. With these two functionalities, we aim to achieve the above motive.

ACKNOWLEDGEMENT

We want to express our most significant appreciation to all the individuals who have helped and supported us throughout the project. We are thankful to our **COMPUTER NETWORKS LAB** teachers for their ongoing support during the project, from initial advice and encouragement, which led to the final report of this project. I would also like to thank **Dr. Krishnamoorthi M** and **Ms. Sucharitha S**, who always guided us in the computer lab.

A special acknowledgment goes to our classmates who helped us complete the project by exchanging interesting ideas and sharing their experiences.

We wish to thank our parents for their undivided support and interest in our interests and well-being, inspiring us and encouraging us to go our own way. Without their support and blessing, we wouldn't be in this position.

In the end, we want to thank our friends who displayed appreciation for our work and motivated us to continue our work.

TABLE OF CONTENT

Chapter 4: TOOL TO ANALYZE FLAG OF A TCP PACKET: BACK END

Chapter 5: OUTPUT AND APPLICATION OF THE TOOL

LIST OF FIGURES

LIST OF TABLES

Chapter 1: INTRODUCTION TO TCP

1.1 INTRODUCTION TO COMPUTER NETWORK

A computer network is defined as an interconnection of multiple devices, known as hosts, connected using multiple paths to send/receive data or media. Computer networks can include numerous devices/ mediums that help communicate between two devices, and these devices/ mediums are known as Network Devices and include routers, switches, hubs, and bridges.

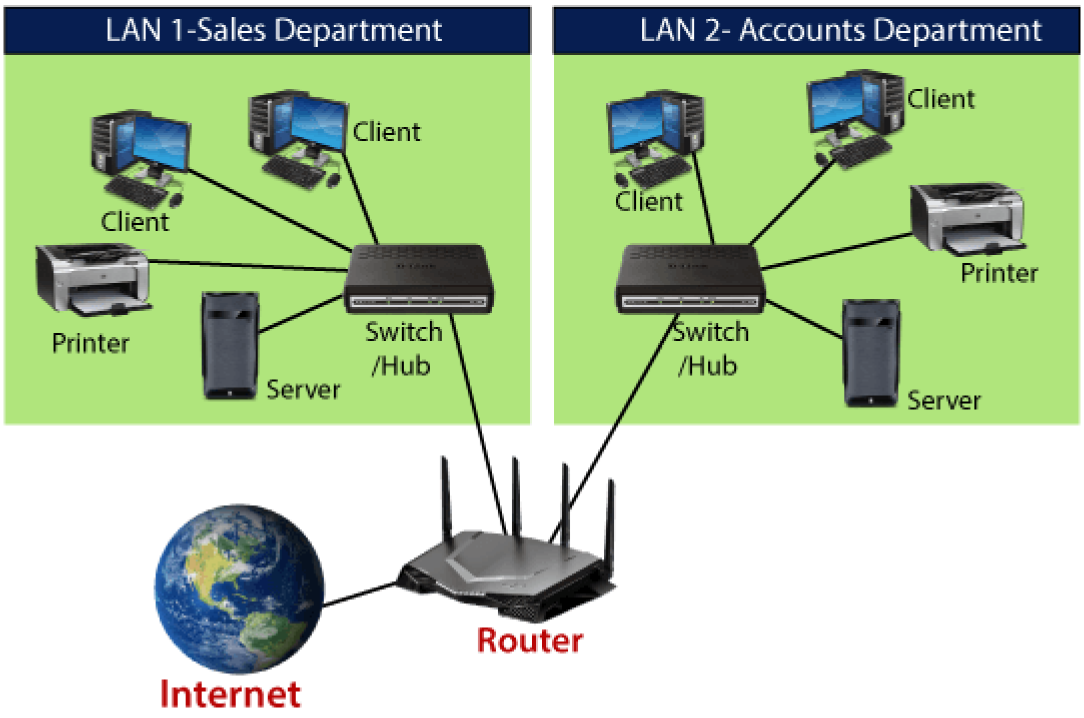


Figure 1.1: A computer network with hosts, switches/ hubs, servers, and routers

Ref: https://www.javatpoint.com/router

1.2 WHAT IS INTERNET?

Internet is basically a computer network that connects hundreds of millions of computing devices across the world.

These computing devices can be called hosts or end systems in internet jargon. These end systems are connected by a network of communication links and packet switches. When one system has to send other systems any data, it segments the data and adds header information to each segment. These segments are called packets and are sent to the destination system through the network, which consists of routers and link-layer switches. End systems access the Internet through ISPs (Internet Service Providers). The TCP (Transmission Control Protocol [Transport Layer]) and IP (Internet Protocol [Network Layer]) are the two most important protocols of the Internet. We will discuss Transport layer protocol TCP briefly in this chapter later.

1.3 MODULAR APPROACH TO INTERNET

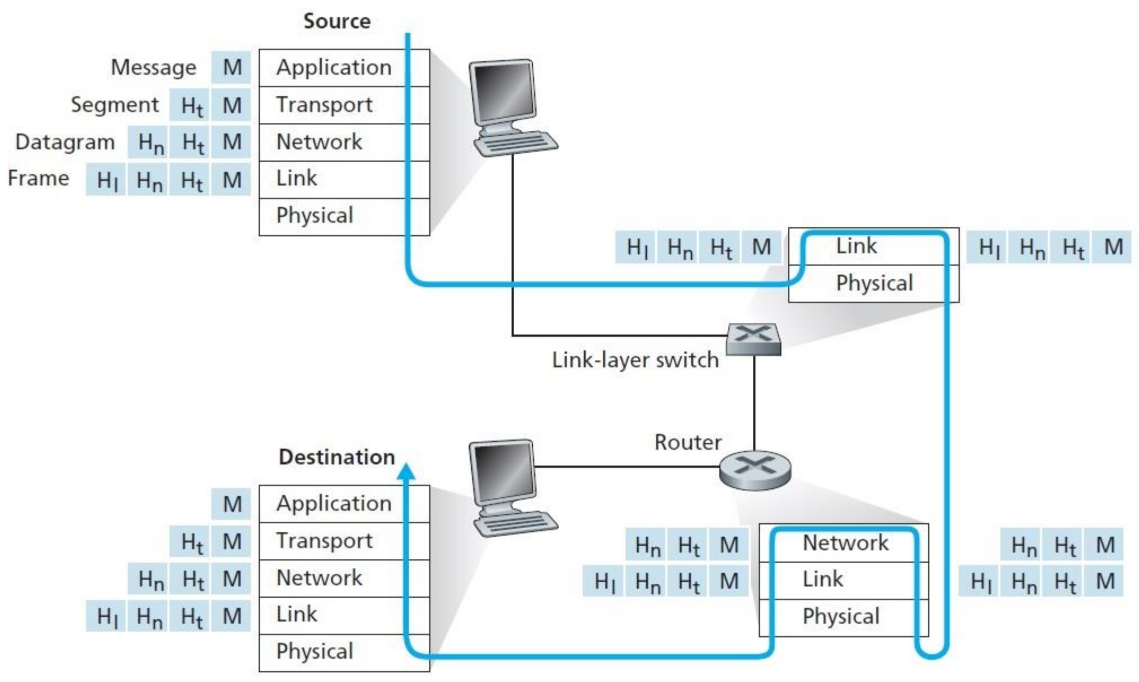


Figure 1.2 Host, Link-Layer Switches, and Routers with their different layers indicating their differences in functionality

Ref: Computer Networking: A Top-Down Approach 6th edition

We know that the Internet is a complex system with various applications, end systems, link-layer switches, routers, protocols, etc. For simplifying this complexity, the Internet can be modeled as a layered approach. TCP/IP has at most five layers for a network device, namely 1) Application, 2) Transport, 3) Network, 4) Link, and 5) Physical layer. When taken together, the protocols of the various layers is called protocol stack.

Our project is a tool to analyze flags in a TCP datagram. TCP is a transport layer protocol, and we'll discuss the transport layer specifically before discussing TCP.

1.4 TRANSPORT LAYER

The transport layer is a central piece of the layered network architecture between the application and network layers. It has the critical role of providing communication services directly to the application processes running on different hosts. The Internet has two protocols—TCP and UDP.

UDP (User Datagram Protocol) is a connectionless protocol, and TCP is a connection-oriented protocol. Usually, UDP (best-effort protocol) is chosen when the speed is more important than the reliability. It's primarily used in video conferencing, live streaming, gaming, etc. It doesn't have mechanisms to handle flow/ congestion/ loss in the network. Hence where reliability is of importance, TCP is preferred over UDP.

1.5 TCP (TRANSMISSION CONTROL PROTOCOL)

TCP is a connection-oriented protocol. It is called a connection-oriented protocol because before sending data between two hosts, a connection has to be set up between sender and receiver with three-way hand-shaking.

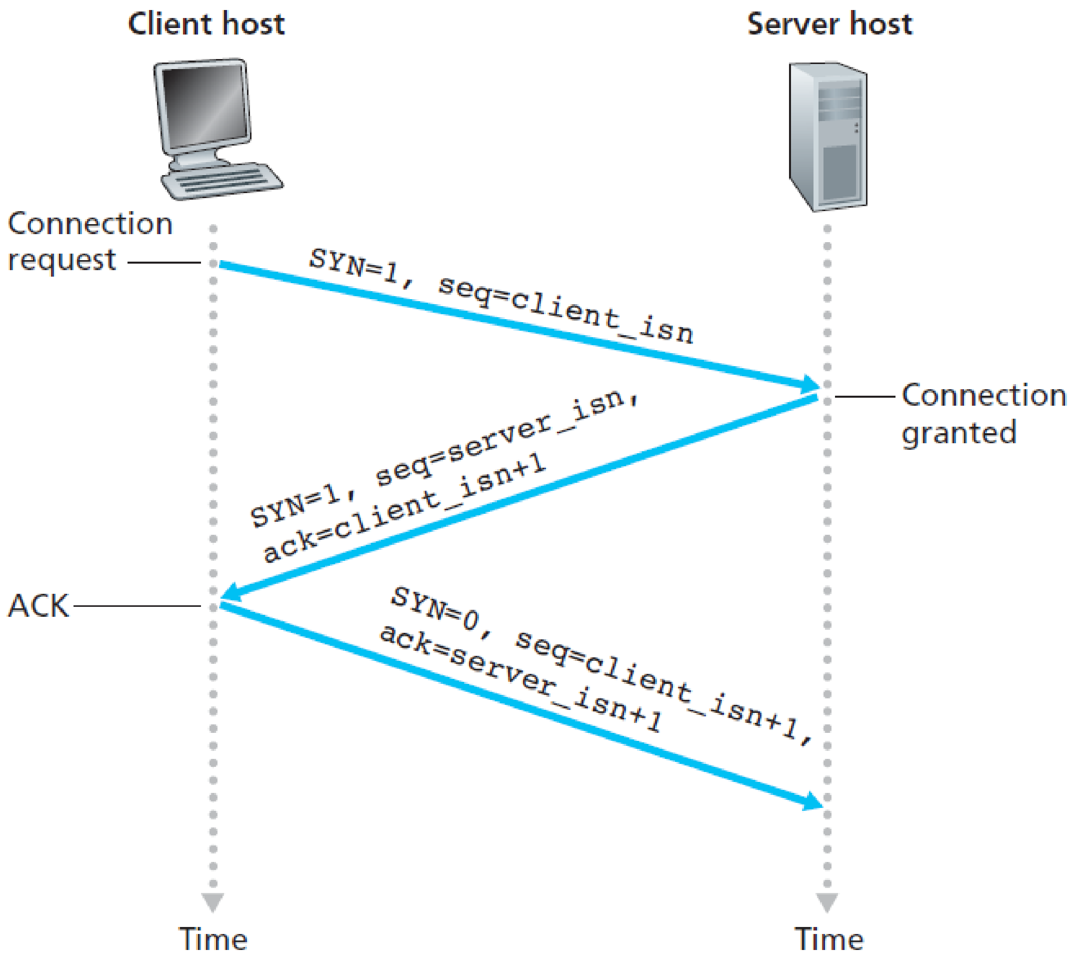


Figure 1.3 Three-way hand-shaking to establish the connection.

Ref: Computer Networking: A Top-Down Approach 6th edition

Three-way hand-shaking is done by sending a connection request to the destination. At the destination side, if a connection request can be accepted, the connection is granted, and also a connection request is transmitted from destination to source. The source finally acknowledges the connection request from the destination and indicates that the connection is ready for data transfer. The connection termination also similarly ends with a three-way handshake. Due to this connection setup mechanism, TCP is called a connection-oriented internet protocol.

TCP is also considered a reliable protocol for data transfer, contrary to UDP. This reliability is achieved by ensuring in-order transfer of data, acknowledgments, flow control, and congestion control mechanisms.

TCP is a very vast topic, but we'll not cover everything in this project. In the next chapter, we'll go through the structure of the TCP datagram and its flags, which is the main topic of our project.

[*This page is intentionally left blank*]

Chapter 2: TCP PACKETS AND FLAGS

2.1 INTRODUCTION

In the previous chapter, we briefly introduced computer networks and TCP. This chapter will delve into more specific parts of our project, i.e., TCP packets and Flags. We'll go through various parts of the TCP packet and discuss what each flag is used for. From the next chapter onwards, we'll discuss how our project analyzes TCP packets and their flags.

2.2 TCP SEGMENT/ PACKET

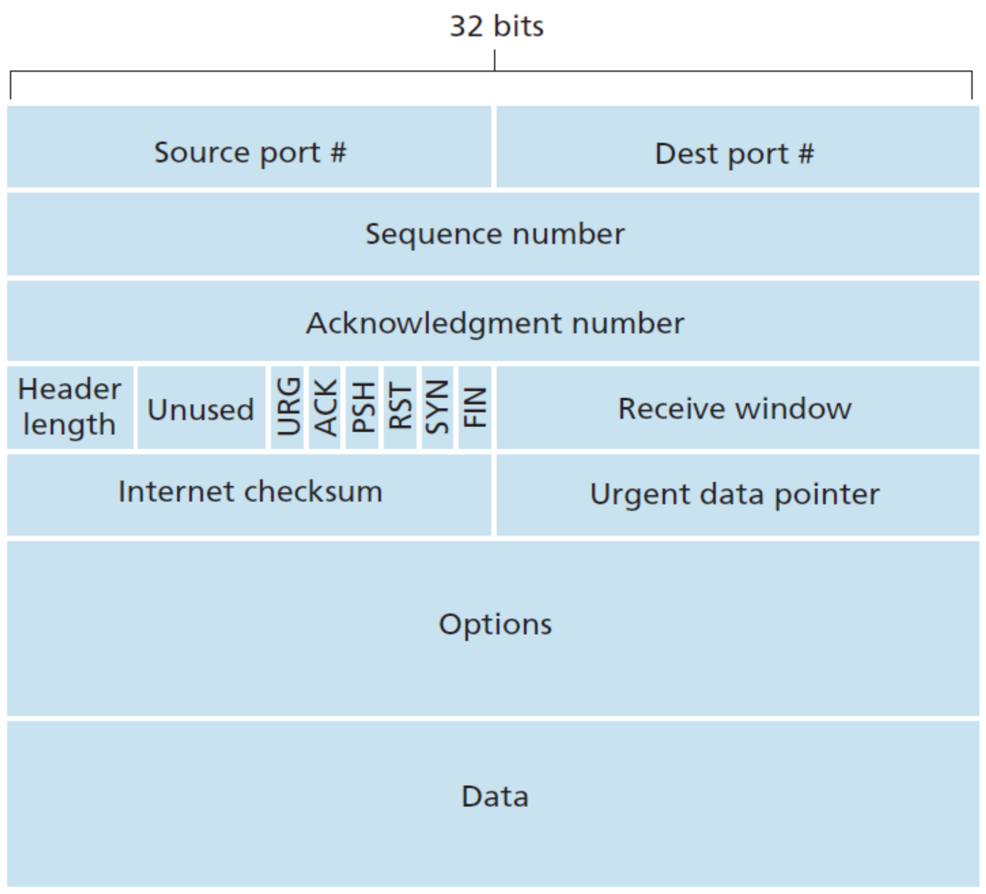


Figure 2.1 TCP segment structure

Ref: Computer Networking: A Top-Down Approach 6th edition

The TCP segment consists of header fields and a data field. The data field contains a chunk of application data. The MSS limits the maximum size of a segment's data field. When TCP sends a large file, such as an image, as part of a Web page, it typically breaks the file into chunks of size MSS (except for the last chunk, which will often be less than the MSS). The MSS is typically set by first determining the length of the largest link-layer frame that the local sending host can send (the so-called maximum transmission unit, MTU), and then setting the MSS to ensure that a TCP segment (when encapsulated in an IP datagram) plus the TCP/IP header length (typically 40 bytes) will fit into a single link-layer frame.

TCP header size is typically 20 bytes and includes the following fields:

- Source and Destination Port Numbers: Used for multiplexing/ demultiplexing data from/ to upper-layer applications.
- 32-bit Sequence Number and 32-bit Acknowledgement number: Used for in-order and reliable data transfer between source and destination.
- 16-bit Receive window: It is mainly used for the flow control mechanism. It is used to indicate the number of bytes the receiver is willing to accept.
- 4-bit Header Length: It is used to specify the length of the header in 32-bit words. TCP header can be of variable length due to the TCP options field. (Typically, the Options field is empty, so that the length of the typical TCP header is 20)

- <u>Variable-length Options field</u>: The Options field is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks. A time-stamping option is also defined.
- <u>6-bit Flags</u>: (Will be discussed in next section.)

2.3 FLAGS

In TCP connection, flags indicate a particular state of the connection or provide some additional helpful information like troubleshooting purposes or handling control of a particular connection. The flag section contains 6 bits (SYN, ACK, PSH, URG, RST, and FIN), out of which our project is focused on 5 of those bits, particularly SYN, ACK, PSH, URG, and FIN. RST bit isn’t in the scope of this project. But we’ll briefly explain all the flag bits in this section.

- <u>SYN</u>: It is used in first step of connection establishment or 3-way handshake process between the two hosts. Only the first packet from the sender and receiver should have this flag set. This is used to synchronize sequence numbers, i.e., to tell the other end which sequence number they should accept.

- <u>ACK</u>: The ACK bit is used to indicate that the value carried in the acknowledgment field is valid; that is, the segment contains an acknowledgment for a segment that has been successfully received.

- <u>PSH</u>: The transport layer, by default, waits for some time for the application layer to send enough data equal to maximum segment size so that the number of packets transmitted on the network minimizes, which is not desirable by some applications like interactive applications (chatting). Similarly, the transport layer at the receiver end buffers packets and transmits them to the application layer if it meets certain criteria. This problem is solved by using PSH. The transport layer sets PSH = 1 and immediately sends the segment to the network layer as soon as it receives a signal from the application layer. On seeing PSH = 1, the receiver transport layer immediately forwards the data to the application layer. It generally tells the receiver to process these packets as they are received instead of buffering them.

- URG: Data inside a segment with URG = 1 flag is immediately forwarded to the application layer even if there is more data is to be given to the application layer. It is used to notify the receiver to process the urgent packets before processing all other packets. The receiver will be notified when all known urgent data has been received.

- FIN: It is used to request connection termination, i.e., when there is no more data from the sender, it requests connection termination. This is the last packet sent by sender and receiver. It frees the reserved resources and gracefully terminates the connection.

- RST: It is used to terminate the connection if the RST sender feels something is wrong with the TCP connection or that the conversation should not exist. It can get sent from the receiver side when a packet is sent to a particular host that was not expecting it.

2.4 PSH v/s URG FLAG

PSH flag	URG flag
• All data in the buffer to be pushed to Network Layer (sender side)/ Application Layer (receiver side).	• Only the urgent data to be given to the application layer immediately.
• Data is delivered in sequence.	• Data is delivered out of sequence.

Table 2.1 Difference between PSH and URG flag.

Ref: https://www.geeksforgeeks.org/tcp-flags/

2.5 FIN v/s RST FLAG

FIN flag	RST flag
• Gracefully terminates the connection.	• Abruptly tells the other side to stop communicating.
• Only one side of the conversation is stopped.	• The whole conversation is stopped.
• No data loss.	• Data is discarded.
• Receiver of FIN keeps communicating if it wants to.	• Receiver has to stop conversation.

Table 2.2 Difference between FIN flag and RST flag.

Ref: https://www.geeksforgeeks.org/tcp-flags/

In the next chapter, we will discuss the front end of our project, where we design a tool to capture TCP packets with their flags and then analyze them based on various criteria.

[*This page is intentionally left blank*]

Chapter 3: TOOL TO ANALYZE FLAGS OF A TCP PACKET: FRONT END

3.1 INTRODUCTION TO TOOL

The tool is designed to analyze TCP packets captured in the format of a Wireshark capture file. The tool aims to give the working of flags and their importance in a particular packet. It also aims to filter the packets according to the flag present in them. The tool has a front end and a back end. Both the ends are designed using Python3. The front end provides with UI, and the back end helps with the output. In this chapter, we'll look at how the front end works and how it's modeled.

3.2 INTRODUCTION TO FRONT END

The front end is designed to have two UI (UI1 and UI2) and an evaluation component. UI1 deals with the input of file. Whereas UI2 is a recurring User input-driven menu. The evaluation component is a connection block between the front end and the back end. It evaluates the input given by UI2 and calls the appropriate back end function. Figure 3.1 illustrates the basic model of the front end.

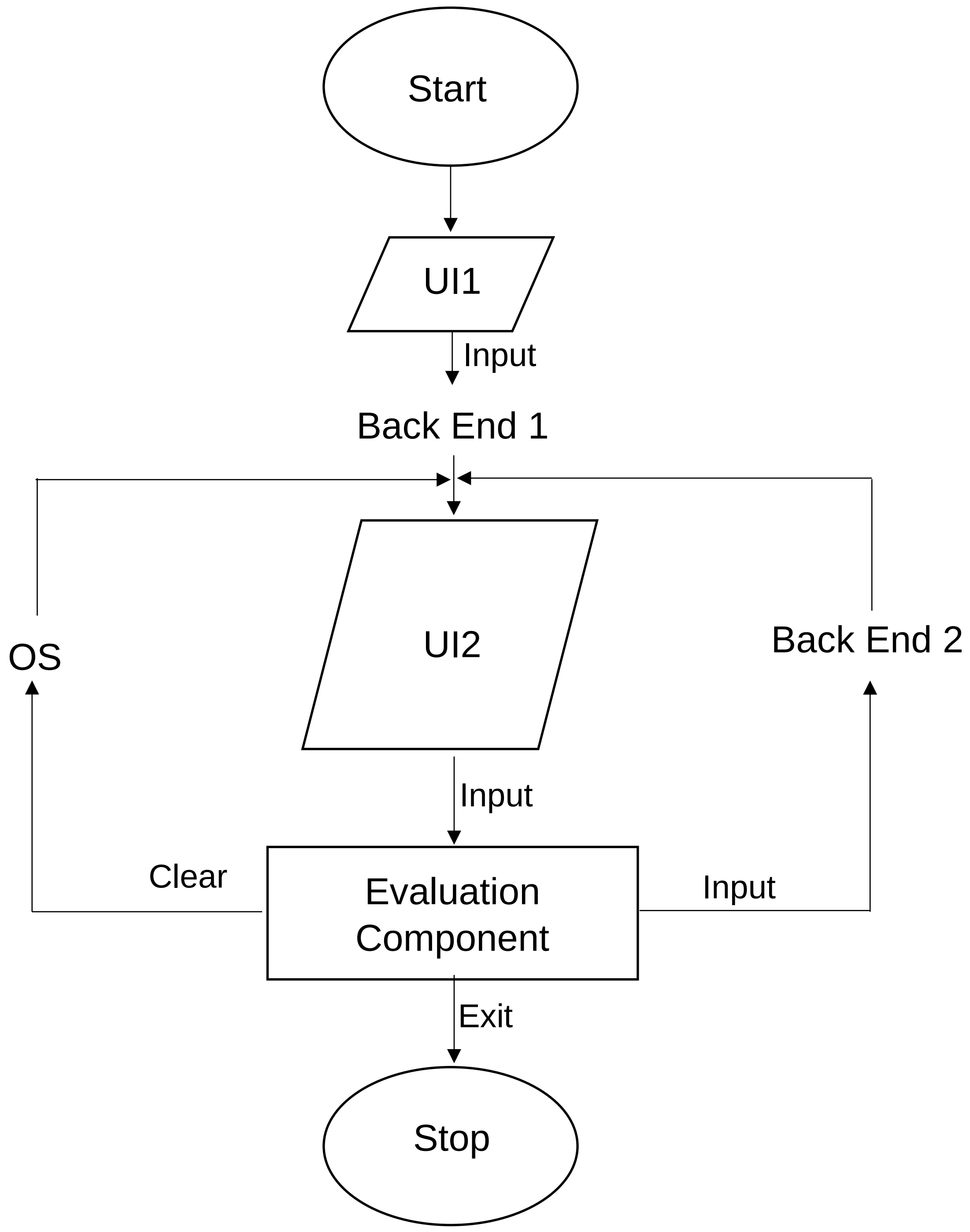


Figure 3.1 Flow-chart of Front End

3.3 UI 1

UI 1 is a very basic input module of the program. It is called only once at the beginning of the program. It inputs the File name that has the Wireshark capture, and it sends the filename to the back end module called createpacketinfo, which reads the file and returns the TCP packets captured in the inputted file. We'll learn more about it in the next chapter.

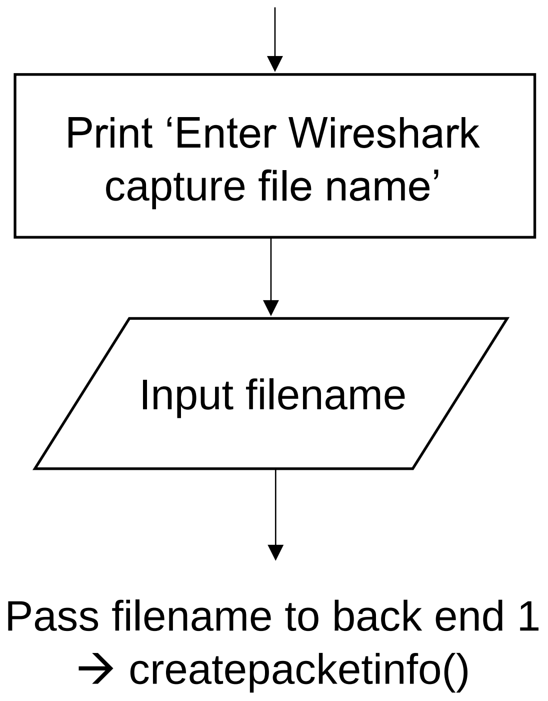
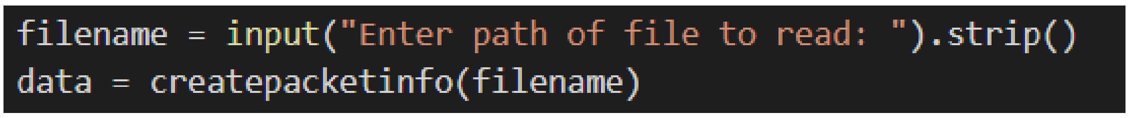

Print 'Enter Wireshark capture file name'
Input filename
Pass filename to back end 1 → createpacketinfo()

Figure 3.2 Flow-chart of UI 1



```
filename = input("Enter path of file to read: ").strip()
data = createpacketinfo(filename)
```

Figure 3.3 Screenshot of code for UI 1

3.4 UI 2

UI 2 is a recurring user interface that gives the user to enter the choice of output. The tool is designed to provide two different types of output, one of which is to analyze flags in each packet and print its relevance. The other one is a filter that filters the packet based on the flag of choice. The user interface also provides the option to clear and exit the recurring loop. It gets the choice from the user and forwards it to the Evaluation component.

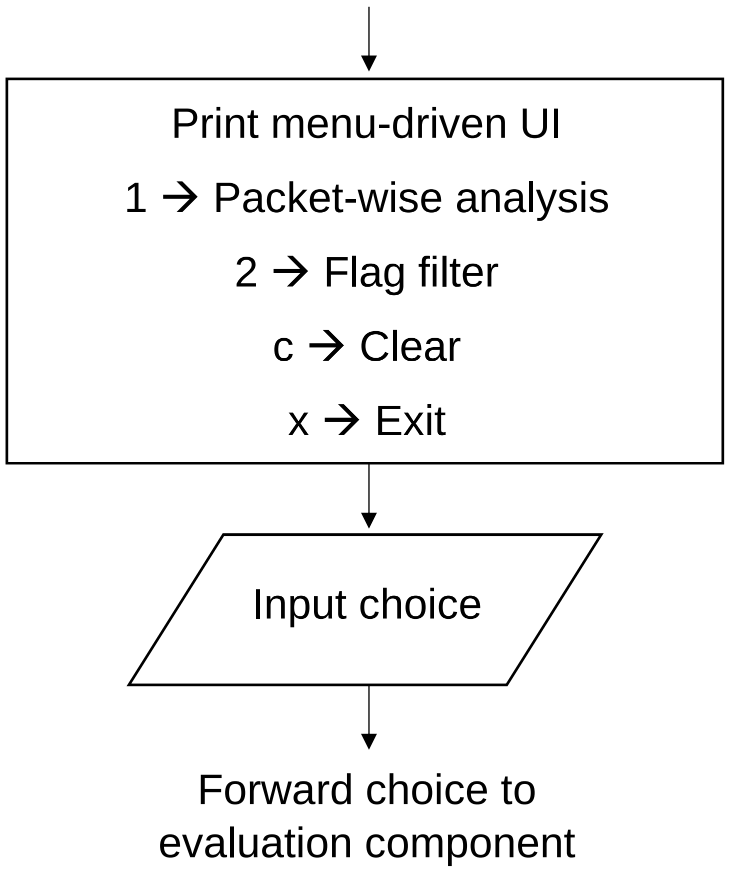
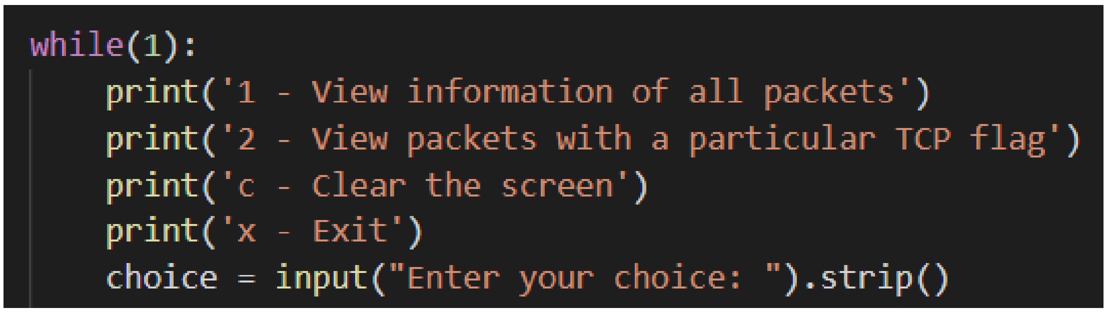


Figure 3.4 Flow-chart of UI 2



```
while(1):
    print('1 - View information of all packets')
    print('2 - View packets with a particular TCP flag')
    print('c - Clear the screen')
    print('x - Exit')
    choice = input("Enter your choice: ").strip()
```

Figure 3.5 Screenshot of code for UI 2

3.5 Evaluation Component

The Evaluation Component of the front end is responsible for evaluating the choices entered in UI 2. It evaluates the choice and calls the appropriate back end function to execute the required analysis. If the choice entered by the user in UI 2 is 1, it calls printallpacketinfo(), which gives a packet-wise analysis regarding the flags it contains. Else if the choice entered is 2, it asks the user for which flag the packets need to be filtered, and after taking a valid flag input from the user, it calls printflaginfo(). The UI 2 also provides the option 'c' for clearing the screen, which the Evaluation component handles by making a call to system call to OS for clearing the screen. If the user enters 'x', the Evaluation component exits the loop and ends the program's execution. For any other value of choice than the values mentioned above, the program prints an invalid input message and re-iterates to UI 2.

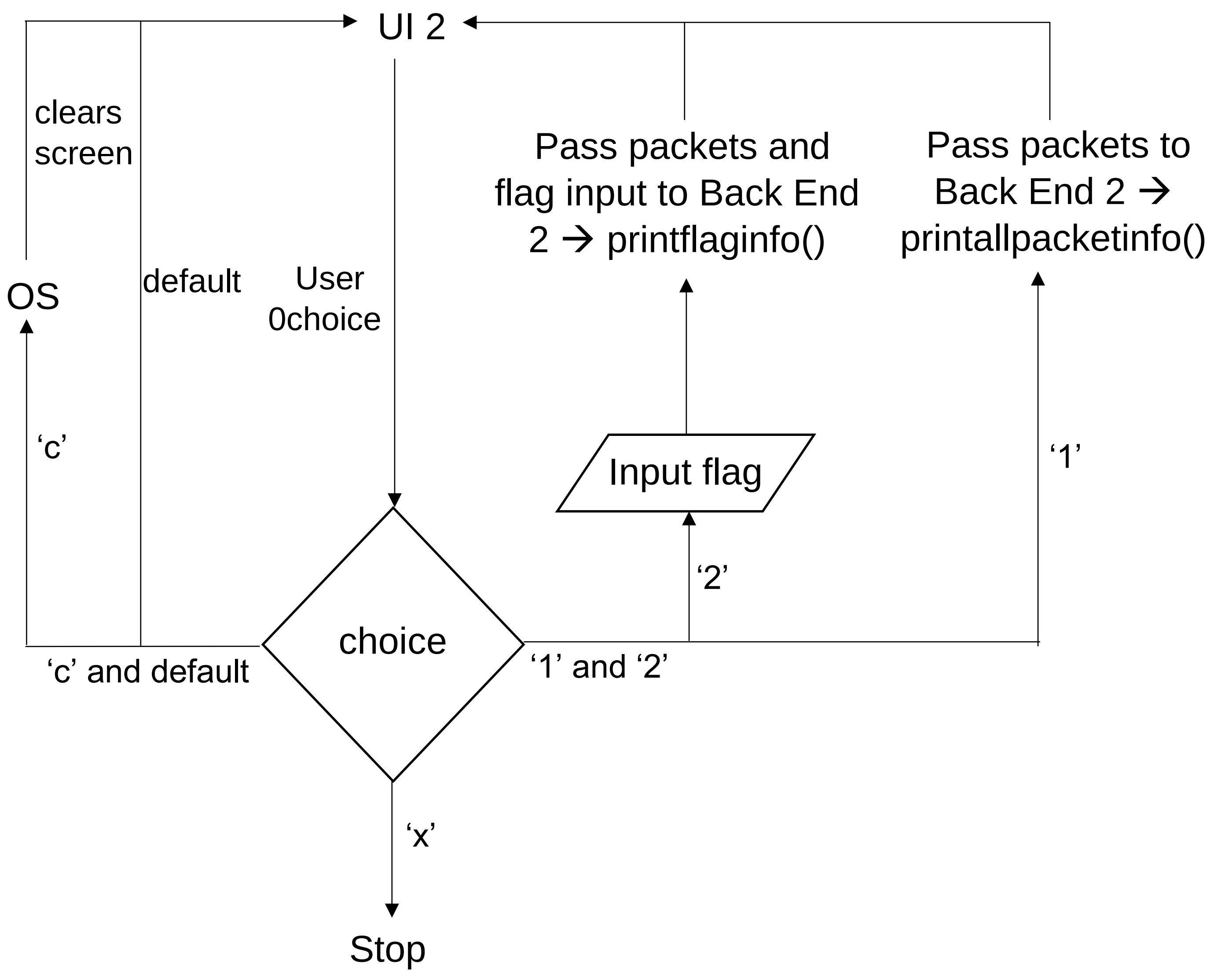
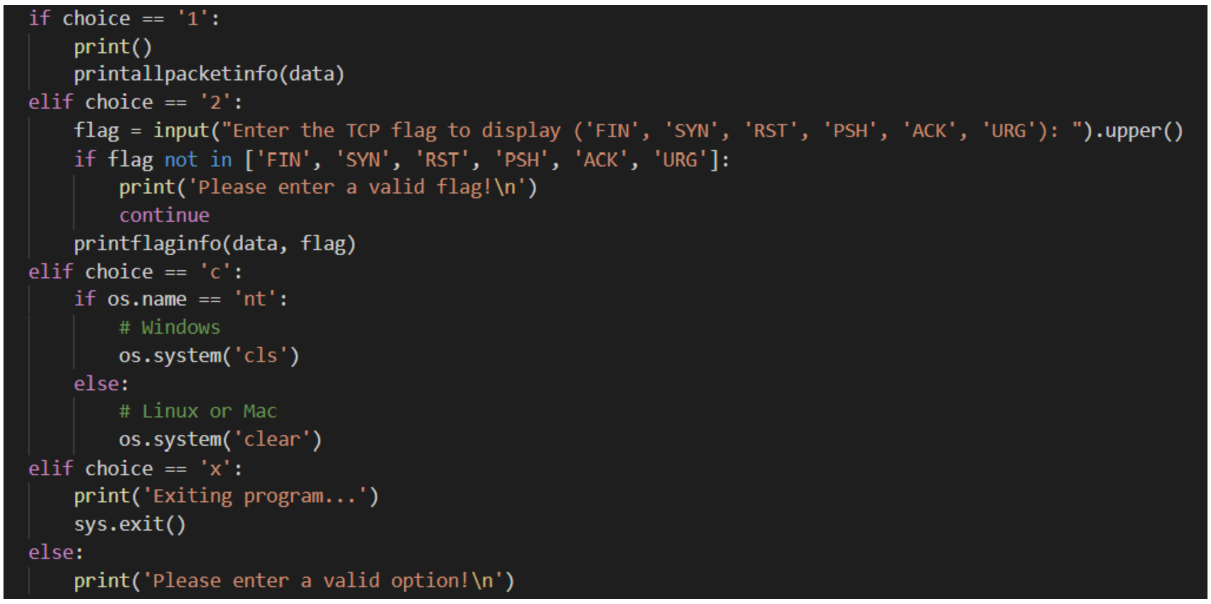

UI 2
clears screen
default
User 0choice
OS
'c'
'c' and default
choice
'x'
Stop
'1' and '2'
'2'
Input flag
Pass packets and flag input to Back End 2 → printflaginfo()
'1'
Pass packets to Back End 2 → printallpacketinfo()

Figure 3.6 Flow-chart of Evaluation Component



```
if choice == '1':
    print()
    printallpacketinfo(data)
elif choice == '2':
    flag = input("Enter the TCP flag to display ('FIN', 'SYN', 'RST', 'PSH', 'ACK', 'URG'): ").upper()
    if flag not in ['FIN', 'SYN', 'RST', 'PSH', 'ACK', 'URG']:
        print('Please enter a valid flag!\n')
        continue
    printflaginfo(data, flag)
elif choice == 'c':
    if os.name == 'nt':
        # Windows
        os.system('cls')
    else:
        # Linux or Mac
        os.system('clear')
elif choice == 'x':
    print('Exiting program...')
    sys.exit()
else:
    print('Please enter a valid option!\n')
```

Figure 3.7 Screenshot of code for Evaluation component

In this chapter saw how UI 1, UI 2, and Evaluation Component worked together to form the front end. In the next chapter, we'll see the tool's back end where the output is generated.

[*This page is intentionally left blank*]

Chapter 4: TOOL TO ANALYZE FLAGS OF A TCP PACKET: BACK END

4.1 INTRODUCTION TO BACK END:

In the previous chapter, we saw the connection between the back end and the front end. In this chapter, we'll have a closer look at the back end and its working. There are two Back end components; for our purpose, let's name it Back End 1 and Back End 2. Back End 1 deals with parsing of the input file, which is a Wireshark capture File. Back End 2 does the analysis and prints the output. First, we'll have a glance at how Back End 1 works.

4.2 BACK END 1

This component of Back End deals with the parsing of the input file. The input file should be a Wireshark capture file, i.e., .pcapng OR .pcap extensions. This component, createpacketinfo(), returns an array of packets parsed from the input filename. It uses an inbuilt Python library, **'pyshark'**, to read the input files and apply the filters. It iterates over all TCP packets and parses flags and other essential data of a packet, like Source and Destination IP Address and Port Number; it also parses Sequence number and acknowledgment number of the packet.

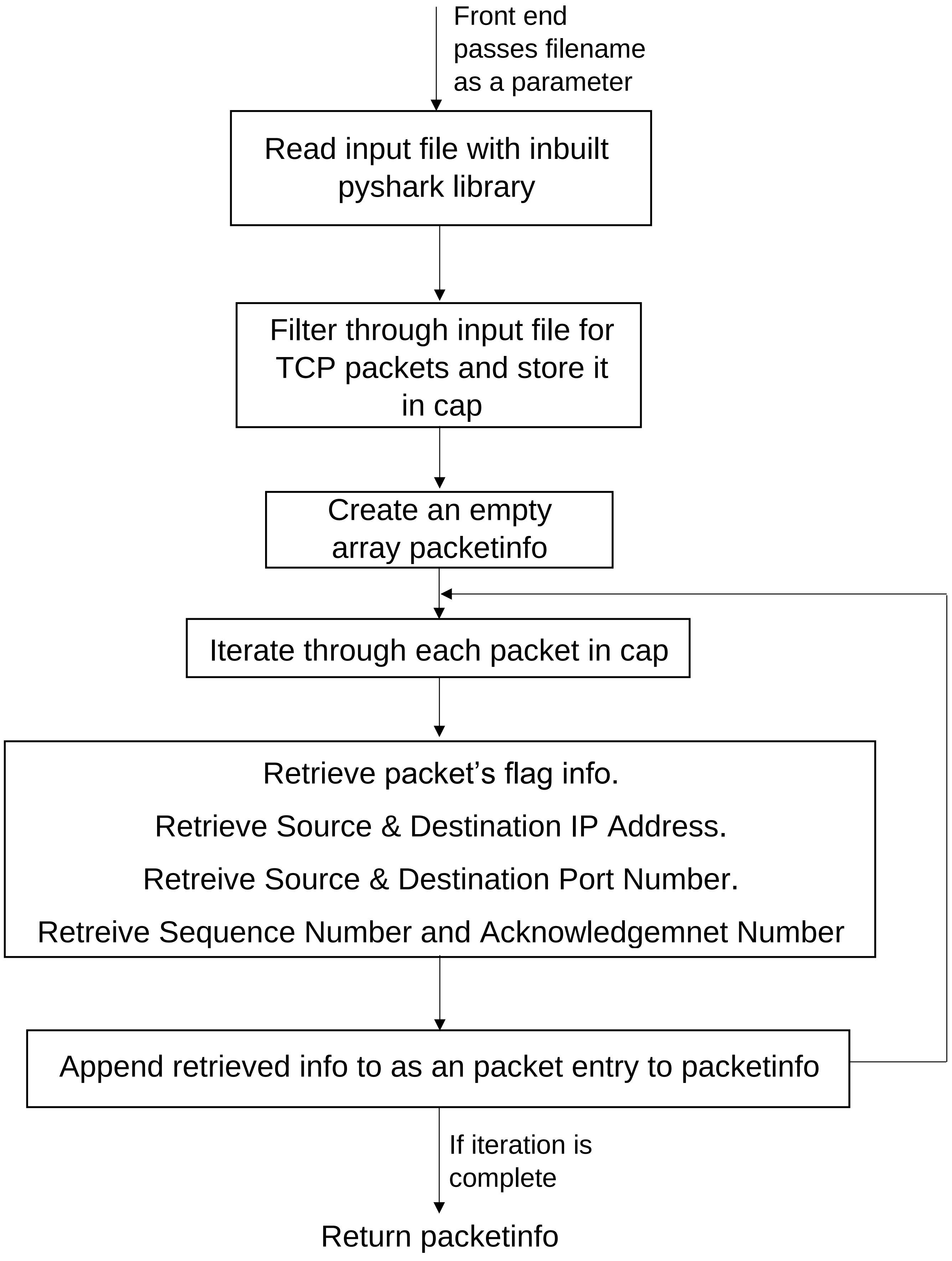
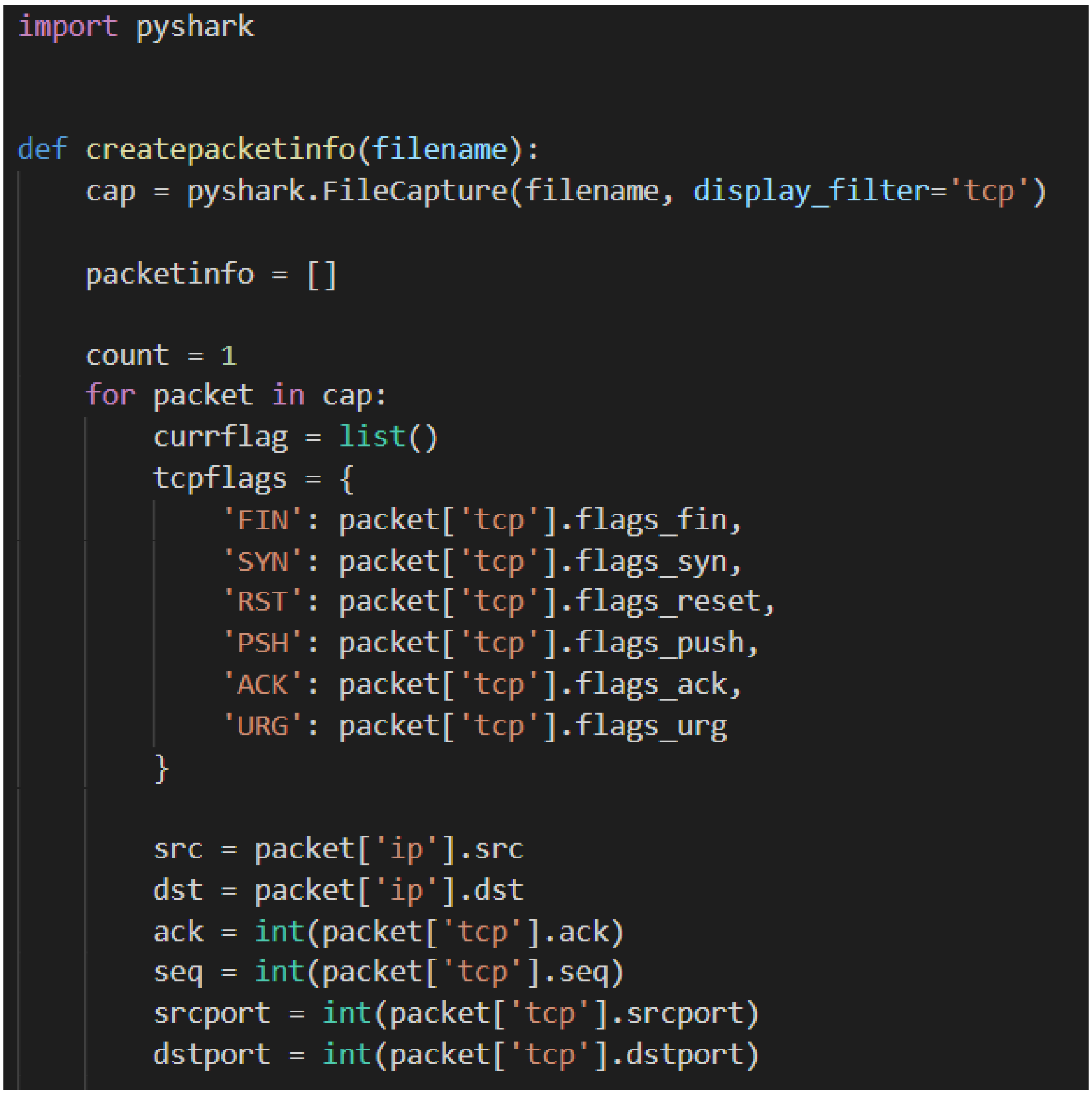
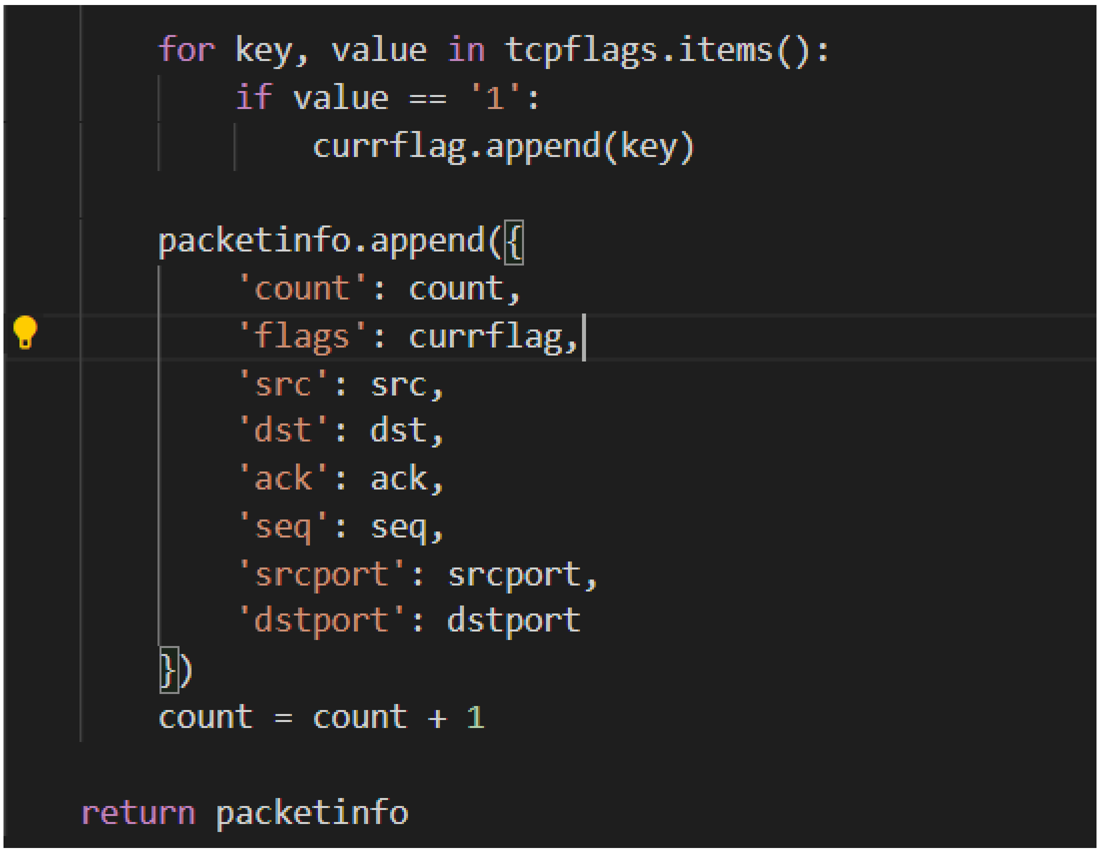

Front end passes filename as a parameter
Read input file with inbuilt pyshark library
Filter through input file for TCP packets and store it in cap
Create an empty array packetinfo
Iterate through each packet in cap
Retrieve packet's flag info.
Retrieve Source & Destination IP Address.
Retreive Source & Destination Port Number.
Retreive Sequence Number and Acknowledgemnet Number
Append retrieved info to as an packet entry to packetinfo
If iteration is complete
Return packetinfo

Figure 4.1 Flow chart of Back End 1 → createpacketinfo()



```python
import pyshark


def createpacketinfo(filename):
    cap = pyshark.FileCapture(filename, display_filter='tcp')

    packetinfo = []

    count = 1
    for packet in cap:
        currflag = list()
        tcpflags = {
            'FIN': packet['tcp'].flags_fin,
            'SYN': packet['tcp'].flags_syn,
            'RST': packet['tcp'].flags_reset,
            'PSH': packet['tcp'].flags_push,
            'ACK': packet['tcp'].flags_ack,
            'URG': packet['tcp'].flags_urg
        }

        src = packet['ip'].src
        dst = packet['ip'].dst
        ack = int(packet['tcp'].ack)
        seq = int(packet['tcp'].seq)
        srcport = int(packet['tcp'].srcport)
        dstport = int(packet['tcp'].dstport)
```

Figure 4.2.1 Code Screenshot for Back End1 → createfileinfo() PART 1



```
        for key, value in tcpflags.items():
            if value == '1':
                currflag.append(key)

        packetinfo.append({
            'count': count,
            'flags': currflag,
            'src': src,
            'dst': dst,
            'ack': ack,
            'seq': seq,
            'srcport': srcport,
            'dstport': dstport
        })
        count = count + 1

    return packetinfo
```

Figure 4.2.2 Code screenshot for Back End → createpacketinfo() PART 2

4.3 BACK END 2

While Back End 1 was designed to parse the input file to get the array of TCP packets, Back End 2 is designed to analyze those TCP packets' flags based on the user's choice and provide appropriate output. Back End 2 is actually two different components, the first one is printallpacketinfo(), and the second one is printflaginfo(). Let's see each one of them in detail.

4.4 BACK END 2 COMPONENT → printallpacketinfo()

The functionality of printallpacketinfo() is to print all packets, display their flags, and mention the relevance of the flag. This is done by analyzing the individual flags, the combination of flags, and the sequence of flags.

Let's see what the message printed/ relevance for all possible packet combinations or sequence is:

SYN: Connection request from source to destination.

SYN + ACK: Connection request is accepted. A connection request is initiated from this side as well.

ACK 1(first ACK from SYN sender's side): Connection is set up, and the connection is ready for data transfer.

ACK 2(any other ACK than the above two combinations): Acknowledges that packet till 'acknowledgment number - 1' has been received. Expecting (acknowledgment number).

FIN: Connection closed from the sender's side.

PSH: Indicates packet needs to be pushed to Application/ Network Layer immediately.

URG: Indicates the transport layer of the receiver to prioritize the packet.

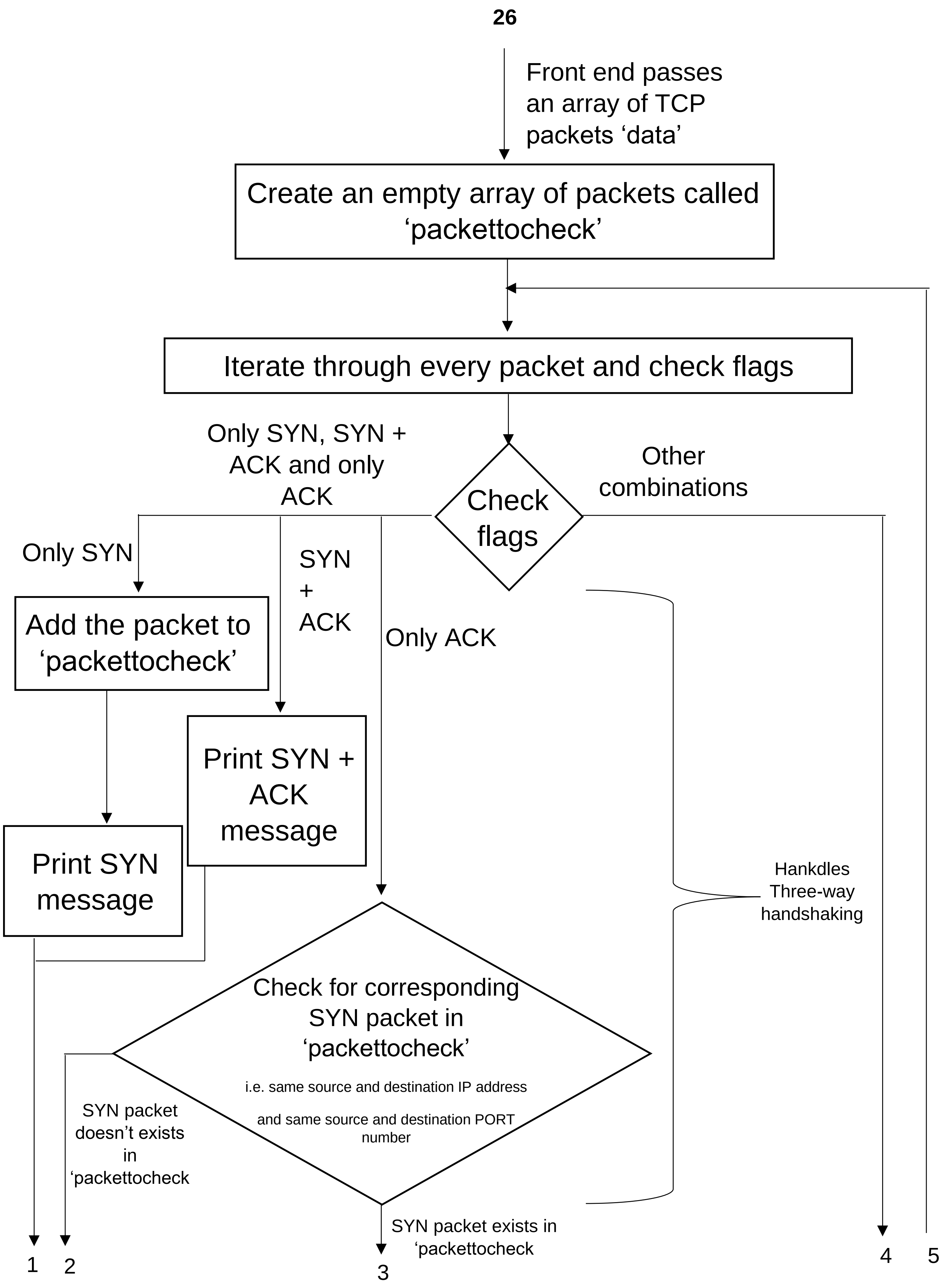

26
Front end passes an array of TCP packets ‘data’
Create an empty array of packets called ‘packettocheck’
Iterate through every packet and check flags
Only SYN, SYN + ACK and only ACK
Check flags
Other combinations
Only SYN
SYN + ACK
Only ACK
Add the packet to ‘packettocheck’
Print SYN + ACK message
Print SYN message
Hankdles Three-way handshaking
Check for corresponding SYN packet in ‘packettocheck’
i.e. same source and destination IP address
and same source and destination PORT number
SYN packet doesn’t exists in ‘packettocheck
SYN packet exists in ‘packettocheck
1
2
3
4
5

Figure 4.3.1 Flow chart of Back End 2 → printallpacketinfo() PART 1

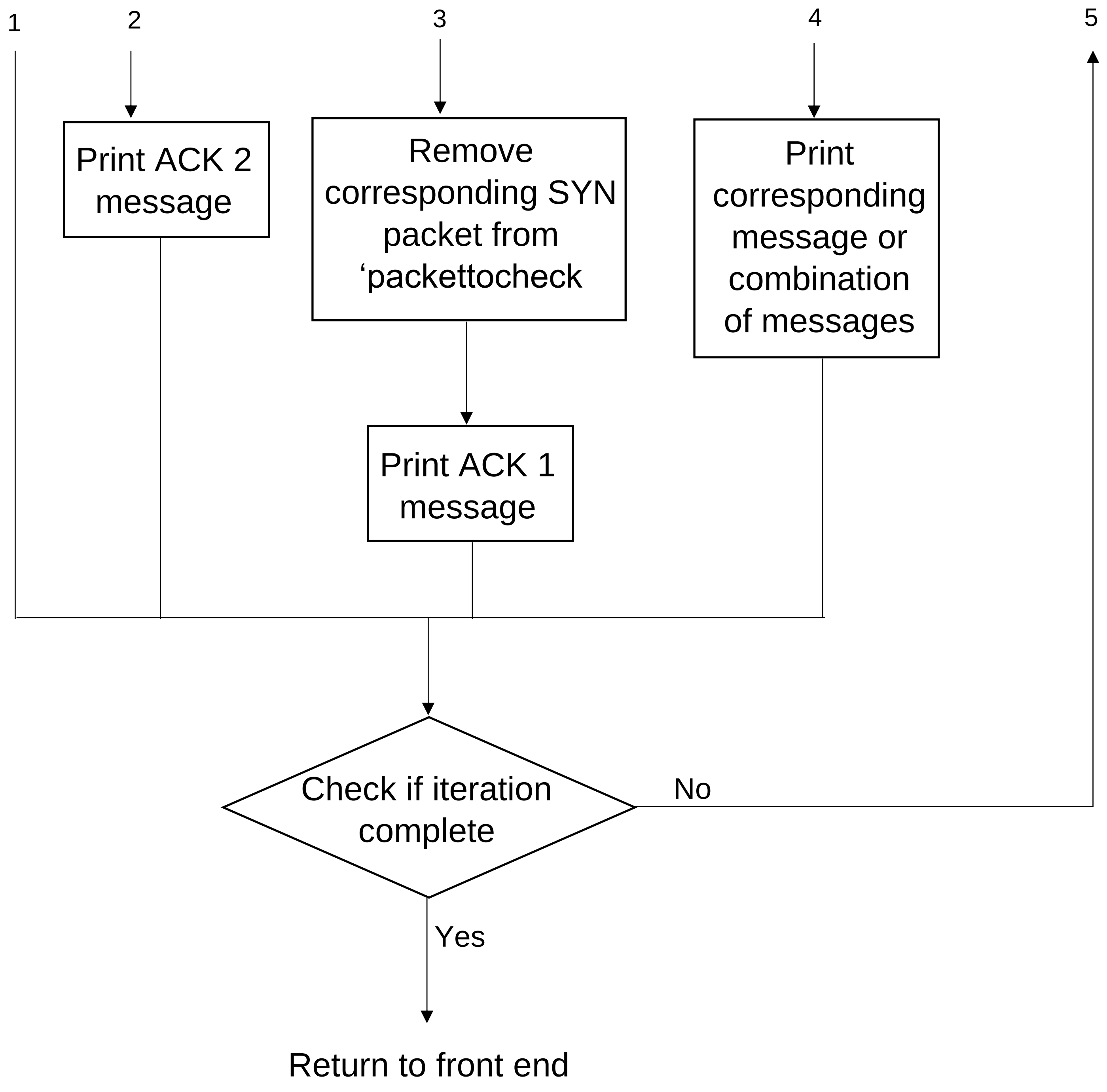
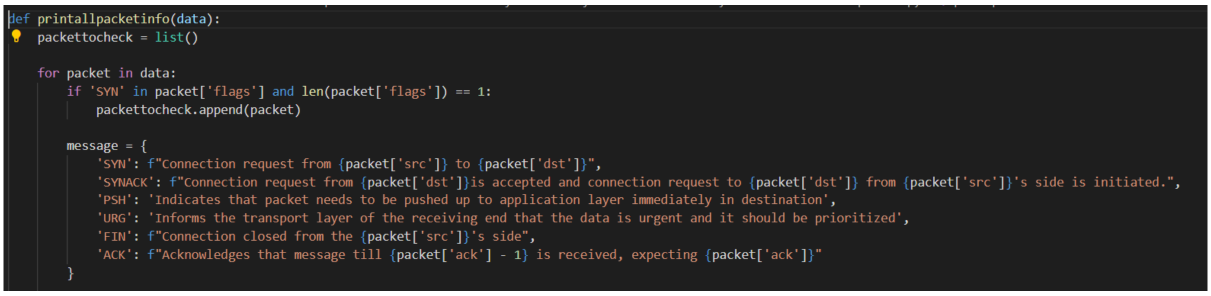
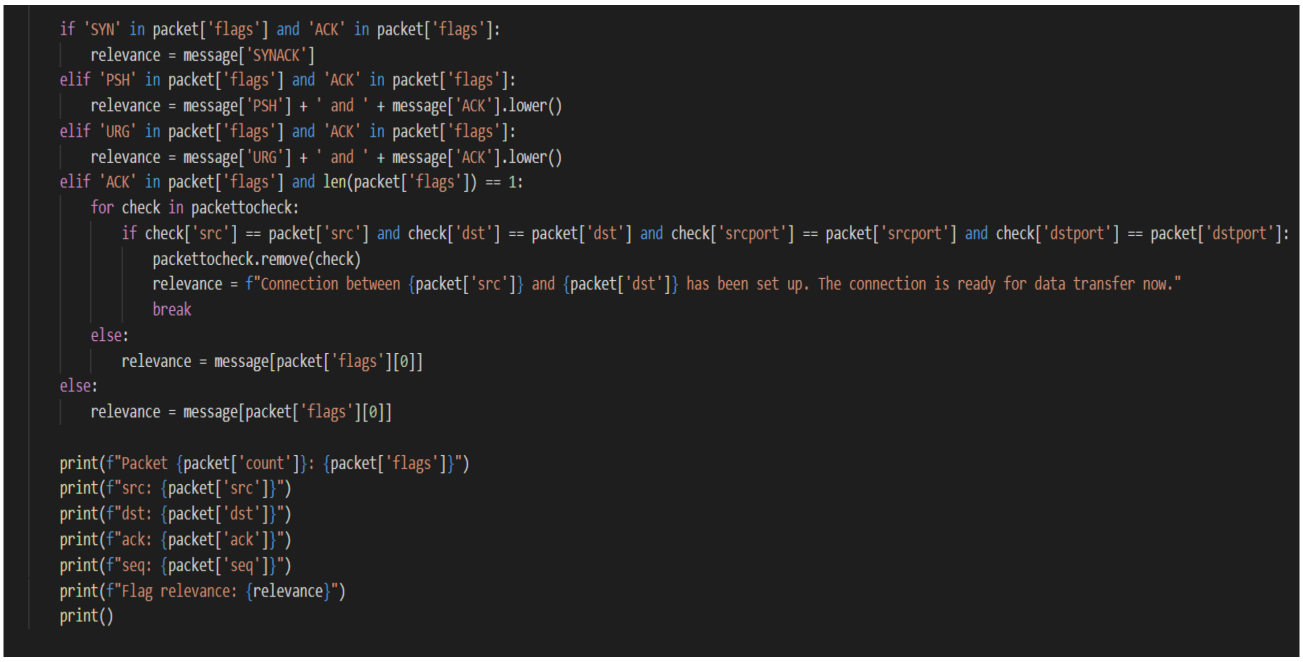

1
2
3
4
5
Print ACK 2 message
Remove corresponding SYN packet from ‘packettocheck
Print corresponding message or combination of messages
Print ACK 1 message
Check if iteration complete
No
Yes
Return to front end

Figure 4.3.2 Flow chart of Back End 2 → printallpacketinfo() PART 2



```python
def printallpacketinfo(data):
    packettocheck = list()

    for packet in data:
        if 'SYN' in packet['flags'] and len(packet['flags']) == 1:
            packettocheck.append(packet)

        message = {
            'SYN': f"Connection request from {packet['src']} to {packet['dst']}",
            'SYNACK': f"Connection request from {packet['dst']}is accepted and connection request to {packet['dst']} from {packet['src']}'s side is initiated.",
            'PSH': 'Indicates that packet needs to be pushed up to application layer immediately in destination',
            'URG': 'Informs the transport layer of the receiving end that the data is urgent and it should be prioritized',
            'FIN': f"Connection closed from the {packet['src']}'s side",
            'ACK': f"Acknowledges that message till {packet['ack'] - 1} is received, expecting {packet['ack']}"
        }
```

Figure 4.4.1 Code screenshot for Back End 2 → printallpacketinfo() PART 1



```python
        if 'SYN' in packet['flags'] and 'ACK' in packet['flags']:
            relevance = message['SYNACK']
        elif 'PSH' in packet['flags'] and 'ACK' in packet['flags']:
            relevance = message['PSH'] + ' and ' + message['ACK'].lower()
        elif 'URG' in packet['flags'] and 'ACK' in packet['flags']:
            relevance = message['URG'] + ' and ' + message['ACK'].lower()
        elif 'ACK' in packet['flags'] and len(packet['flags']) == 1:
            for check in packettocheck:
                if check['src'] == packet['src'] and check['dst'] == packet['dst'] and check['srcport'] == packet['srcport'] and check['dstport'] == packet['dstport']:
                    packettocheck.remove(check)
                    relevance = f"Connection between {packet['src']} and {packet['dst']} has been set up. The connection is ready for data transfer now."
                    break
            else:
                relevance = message[packet['flags'][0]]
        else:
            relevance = message[packet['flags'][0]]

        print(f"Packet {packet['count']}: {packet['flags']}")
        print(f"src: {packet['src']}")
        print(f"dst: {packet['dst']}")
        print(f"ack: {packet['ack']}")
        print(f"seq: {packet['seq']}")
        print(f"Flag relevance: {relevance}")
        print()
```

Figure 4.4.2 Code screenshot for Back End 2 → printallpacketinfo() PART 2

4.5 BACK END 2 COMPONENT→ printflaginfo()

The second component of Back End 2 is printflaginfo(). Its function is to display the packets that have a particular flag. Whenever the user chooses the option of flag filter from UI 2, the Evaluation Component of the front end asks the user to input a flag. Then the input flag and an array of TCP packets are passed as a parameter to printflaginfo(). Then this component only iterates through all the packets and displays the packets with this flag bit equal to 1.

This component helps in getting the packets with a particular flag. And it also displays statistics like the percentage of packets with that specific flag. Once the execution is completed, it returns the control to the Front End.

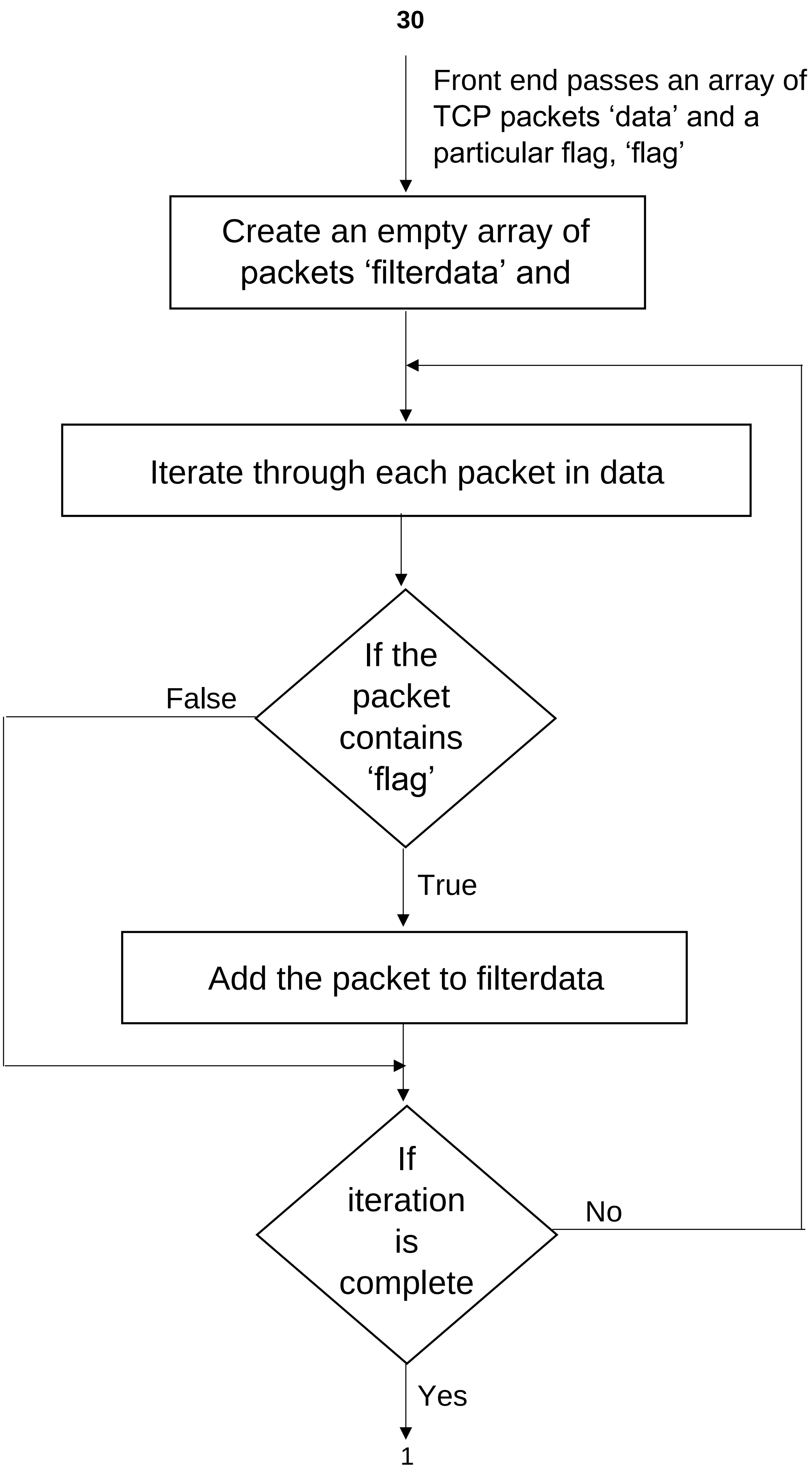

30
Front end passes an array of TCP packets 'data' and a particular flag, 'flag'
Create an empty array of packets 'filterdata' and
Iterate through each packet in data
If the packet contains 'flag'
False
True
Add the packet to filterdata
If iteration is complete
No
Yes
1

Figure 4.5.1 Flow Chart of Back End 2 →printflaginfo() PART 1

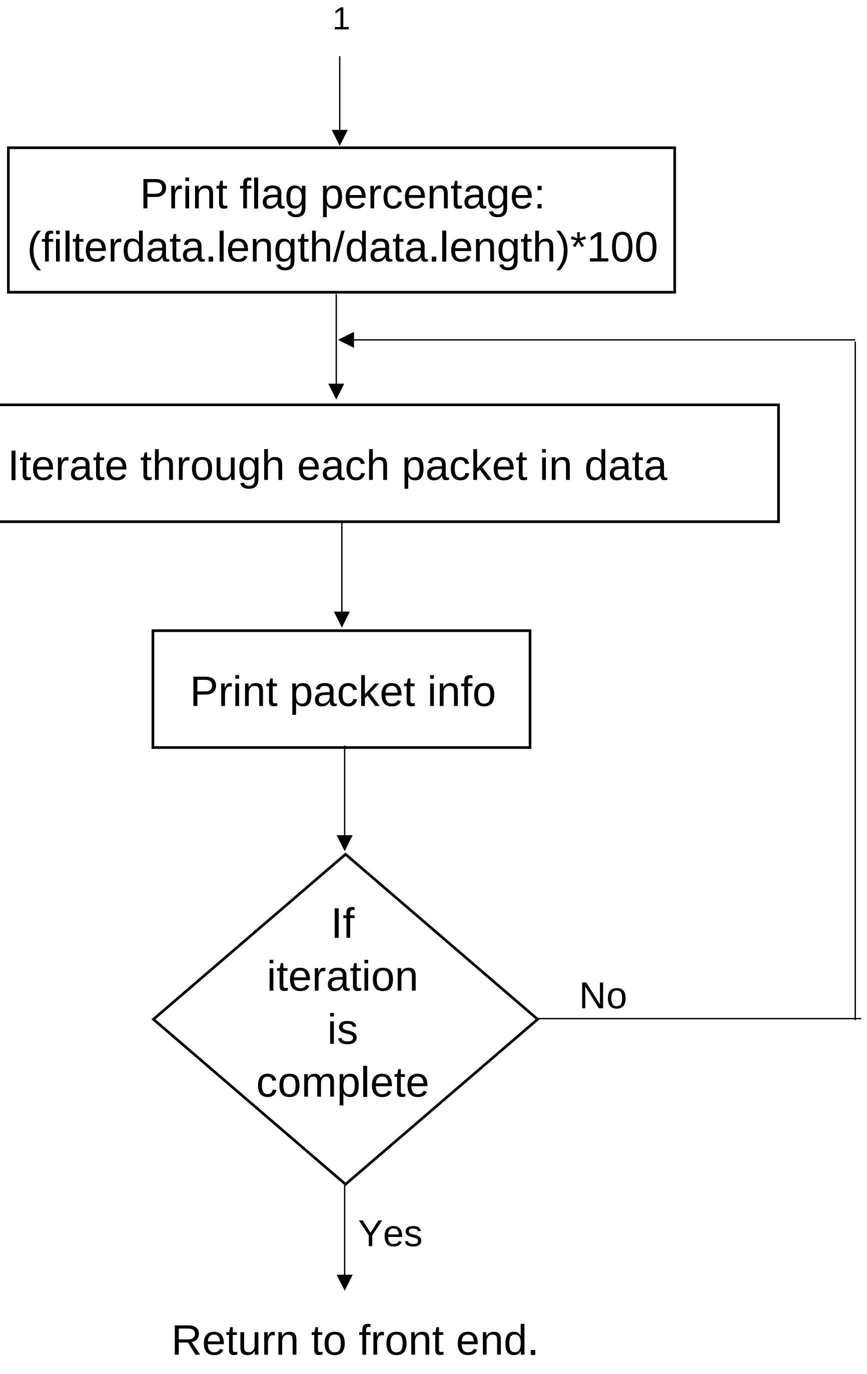
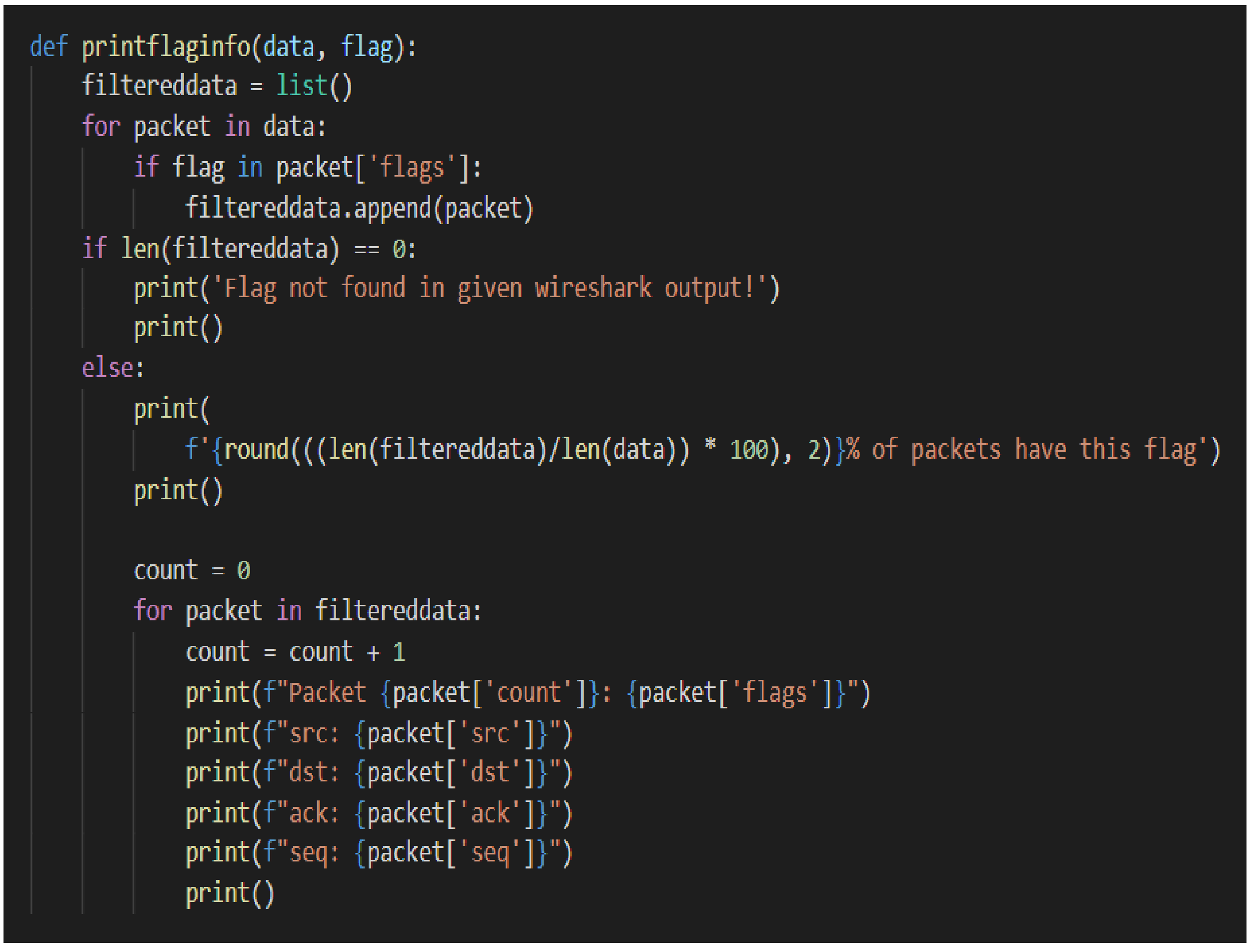

1
Print flag percentage: (filterdata.length/data.length)*100
Iterate through each packet in data
Print packet info
If iteration is complete
No
Yes
Return to front end.

Figure 4.5.1 Flow Chart of Back End 2 → printflaginfo() PART 2



```python
def printflaginfo(data, flag):
    filtereddata = list()
    for packet in data:
        if flag in packet['flags']:
            filtereddata.append(packet)
    if len(filtereddata) == 0:
        print('Flag not found in given wireshark output!')
        print()
    else:
        print(
            f'{round(((len(filtereddata)/len(data)) * 100), 2)}% of packets have this flag')
        print()

        count = 0
        for packet in filtereddata:
            count = count + 1
            print(f"Packet {packet['count']}: {packet['flags']}")
            print(f"src: {packet['src']}")
            print(f"dst: {packet['dst']}")
            print(f"ack: {packet['ack']}")
            print(f"seq: {packet['seq']}")
            print()
```

Figure 4.6 Code screenshot for Back End 2 → printflagtinfo() PART 2

In this chapter, we went through the back-end components and their functionality. In the next chapter, we will see the output of the tool and its applications.

[*This page is intentionally left blank*]

Chapter 5: OUTPUT AND APPLICATION OF THE TOOL

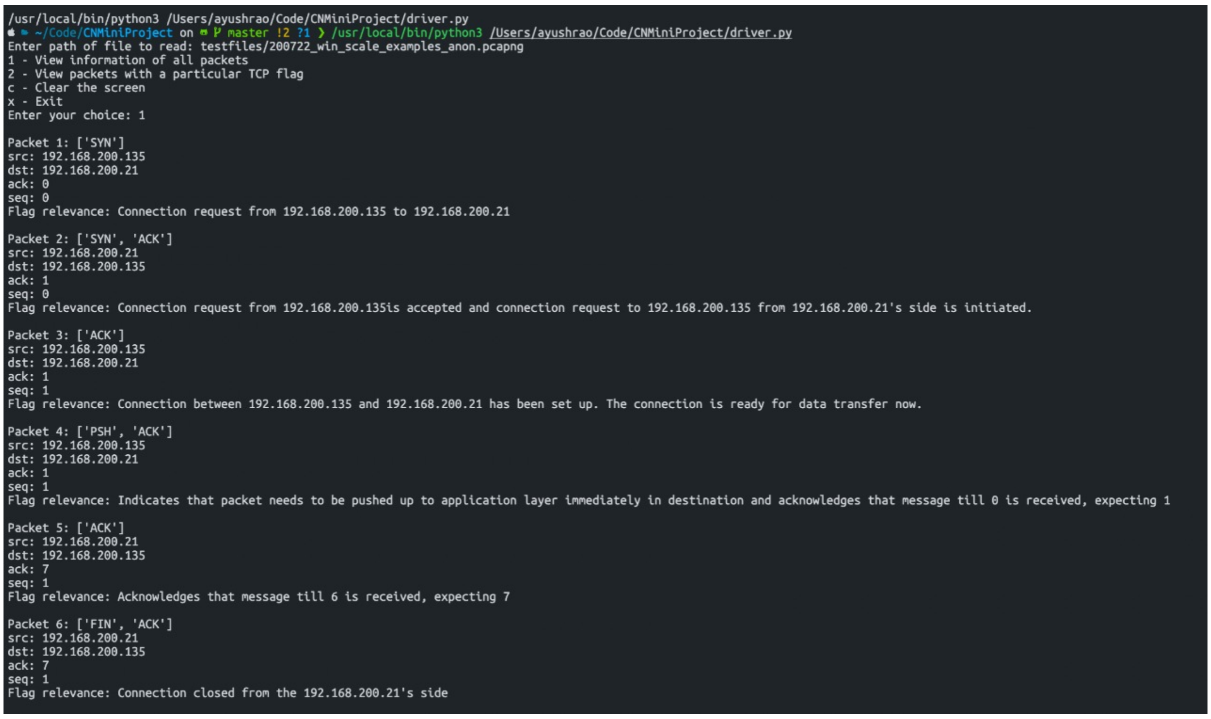
5.1 TOOL’S OUTPUT

This tool intends to give the user data on the relevance of flags in packets and provides a filter to analyze packets concerning flags.

The UI provides the user an interface to enter the filename of the Wireshark capture file, and then the file is read in as input, and TCP packets are filtered from the whole capture file.

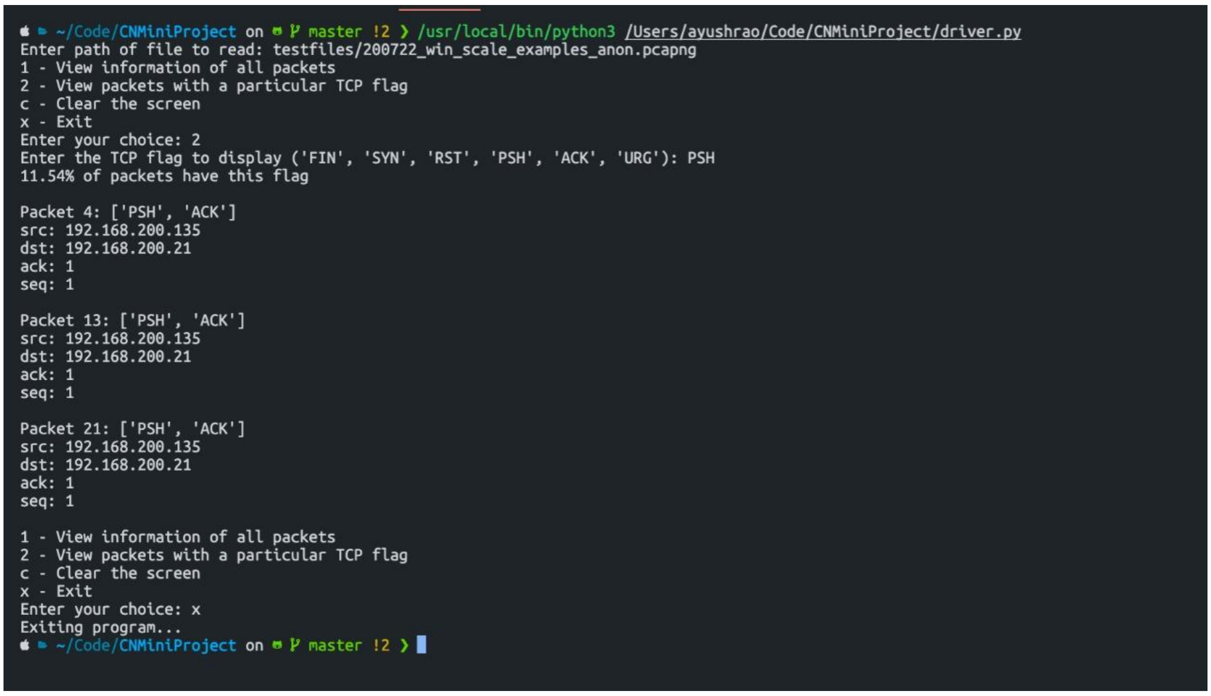
Once the packets are filtered, the UI allows the user to perform packet-wise flag analysis or filter the packets concerning flags which also provides the user with statistics related to the percentage of packets containing the flag.

Figure 5.1 Gives the output for packet-wise analysis of flags on a sample Wireshark capture. Figure 5.2 gives the output for flag filter on FIN flag.



```
/usr/local/bin/python3 /Users/ayushrao/Code/CNMiniProject/driver.py
 ~/Code/CNMiniProject on  master !2 ?1 ❯ /usr/local/bin/python3 /Users/ayushrao/Code/CNMiniProject/driver.py
Enter path of file to read: testfiles/200722_win_scale_examples_anon.pcapng
1 - View information of all packets
2 - View packets with a particular TCP flag
c - Clear the screen
x - Exit
Enter your choice: 1

Packet 1: ['SYN']
src: 192.168.200.135
dst: 192.168.200.21
ack: 0
seq: 0
Flag relevance: Connection request from 192.168.200.135 to 192.168.200.21

Packet 2: ['SYN', 'ACK']
src: 192.168.200.21
dst: 192.168.200.135
ack: 1
seq: 0
Flag relevance: Connection request from 192.168.200.135is accepted and connection request to 192.168.200.135 from 192.168.200.21's side is initiated.

Packet 3: ['ACK']
src: 192.168.200.135
dst: 192.168.200.21
ack: 1
seq: 1
Flag relevance: Connection between 192.168.200.135 and 192.168.200.21 has been set up. The connection is ready for data transfer now.

Packet 4: ['PSH', 'ACK']
src: 192.168.200.135
dst: 192.168.200.21
ack: 1
seq: 1
Flag relevance: Indicates that packet needs to be pushed up to application layer immediately in destination and acknowledges that message till 0 is received, expecting 1

Packet 5: ['ACK']
src: 192.168.200.21
dst: 192.168.200.135
ack: 7
seq: 1
Flag relevance: Acknowledges that message till 6 is received, expecting 7

Packet 6: ['FIN', 'ACK']
src: 192.168.200.21
dst: 192.168.200.135
ack: 7
seq: 1
Flag relevance: Connection closed from the 192.168.200.21's side
```

Figure 5.1: Output for viewing packet information with flags



```
 ~/Code/CNMiniProject on  master !2 ❯ /usr/local/bin/python3 /Users/ayushrao/Code/CNMiniProject/driver.py
Enter path of file to read: testfiles/200722_win_scale_examples_anon.pcapng
1 - View information of all packets
2 - View packets with a particular TCP flag
c - Clear the screen
x - Exit
Enter your choice: 2
Enter the TCP flag to display ('FIN', 'SYN', 'RST', 'PSH', 'ACK', 'URG'): PSH
11.54% of packets have this flag

Packet 4: ['PSH', 'ACK']
src: 192.168.200.135
dst: 192.168.200.21
ack: 1
seq: 1

Packet 13: ['PSH', 'ACK']
src: 192.168.200.135
dst: 192.168.200.21
ack: 1
seq: 1

Packet 21: ['PSH', 'ACK']
src: 192.168.200.135
dst: 192.168.200.21
ack: 1
seq: 1

1 - View information of all packets
2 - View packets with a particular TCP flag
c - Clear the screen
x - Exit
Enter your choice: x
Exiting program...
 ~/Code/CNMiniProject on  master !2 ❯
```

Figure 5.2: Output of flag filter

5.2 APPLICATIONS OF TOOL

The tool for analyzing flags in packets can have multiple applications. Some of them are listed below:

1. To debug a connection failure.
2. To have a better understanding of TCP architecture.
3. To have a better look at the three-way handshaking and observe a TCP connection taking place.
4. To understand the relevance of flags in a particular scenario.
5. To get the statistics like the percentage of packets having a specific flag.
6. To observe the packets closely that have a particular flag.
7. For special packets with PSH and URG, we can note their usage and understand their significance.
8. To understand and appreciate the importance of the flags in TCP/IP architecture.

REFERENCE

Fig 1.1: https://www.javatpoint.com/router

Figure 1.2: Computer Networking: A Top-Down Approach 6th edition

Figure 1.3: Computer Networking: A Top-Down Approach 6th edition

Figure 2.1: Computer Networking: A Top-Down Approach 6th edition

Table 2.1: https://www.geeksforgeeks.org/tcp-flags/

Table 2.2: https://www.geeksforgeeks.org/tcp-flags/