IPv4 Fragmentation Demonstration

NAME:SAGAR | ROLL:211CH063 | COURSE:CS301M

Problem Statement — Demonstrate IPv4 fragmentation using a client-server architecture.

I. DESIGN, PLAN OF THE PROJECT

The project aims to demonstrate IPv4 fragmentation using a client-server architecture. The client program will accept a file with text data and the Maximum Transmission Unit (MTU) size from the user. It will then fragment the data based on the MTU size and send the packets to the server. The server will receive the fragmented packets, reassemble the datagram, and display the original data.

II .EXPECTED FINAL DESIGN

a)Abbreviations and Acronyms

i) MTU: Maximum Transmission Unit - The largest size of data packet that can be transmitted over a network.

ii) IPv4: Internet Protocol version 4 - The fourth version of the Internet Protocol, which is the primary protocol used for sending data packets across the Internet.

iii) TCP: Transmission Control Protocol - A connection-oriented protocol used for reliable and ordered delivery of data packets over a network.

iv) API: Application Programming Interface - A set of rules and protocols that allow different software applications to communicate with each other.

b) Modules and API:

I. Client Module:
   A. Accepts file path and MTU size from the user.
   B. Fragments the data based on the MTU size.
   C. Sends packets to the server.

II. Server Module:
   A. Listens for incoming connections from the client.
   B. Receives fragmented packets.
   C. Reassembles the datagram.
   D. Displays the original data.

III. Socket Module:

A.Creating Socket: `socket.socket()` to establish communication.
B.Connecting: `connect()` to connect to server.
C.Sending Data: `sendall()` to transmit data packets.
D.Binding and Listening: `bind()` and `listen()` for incoming connections.
E.Accepting Connections: `accept()` to accept client connections.
F.Closing Connection: `close()` for termination.

IV. Math module:
It is used for making arithmetic operations simple in our program.

III.Summary

Client Side Code Summary:

The client program (client.py) performs the following steps:

1. Imports the socket module for network communication.
2. Defines the fragment_and_send_data() function, which takes the file path and MTU size as input.
3. Creates a socket object (client_socket) using socket.socket().
4. Connects to the server using the connect() method with the server's address and port.
5. Opens the specified file in read mode and reads the data.
6. Fragments the data into smaller chunks based on the MTU size.
7. Sends each fragment to the server using sendall() method of the socket.
8. Closes the socket connection.

Server Side Code Summary:

The server program (server.py) performs the following steps:

1. Imports the socket module for network communication.
2. Defines the receive_and_reassemble_data() function.
3. Creates a socket object (server_socket) using socket.socket().
4. Binds the server socket to a specific address and port using bind() method.
5. Listens for incoming connections using listen() method.
6. Accepts incoming connection from the client using accept() method, which returns a new socket object (client_socket) and the client's address.
7. Receives data fragments from the client using the recv() method in a loop until no more data is received.
8. Reassembles the fragments into the original data.
9. Prints information about each received fragment.
10. Prints the reassembled data.
11. Closes the client and server sockets.

These summaries provide an overview of the functionality and main steps performed by the client and server programs in the IPv4 fragmentation demonstration.

IV SNAPSHOTS:

c1.py

C: > Users > sagar _ > cn > c1.py > ...

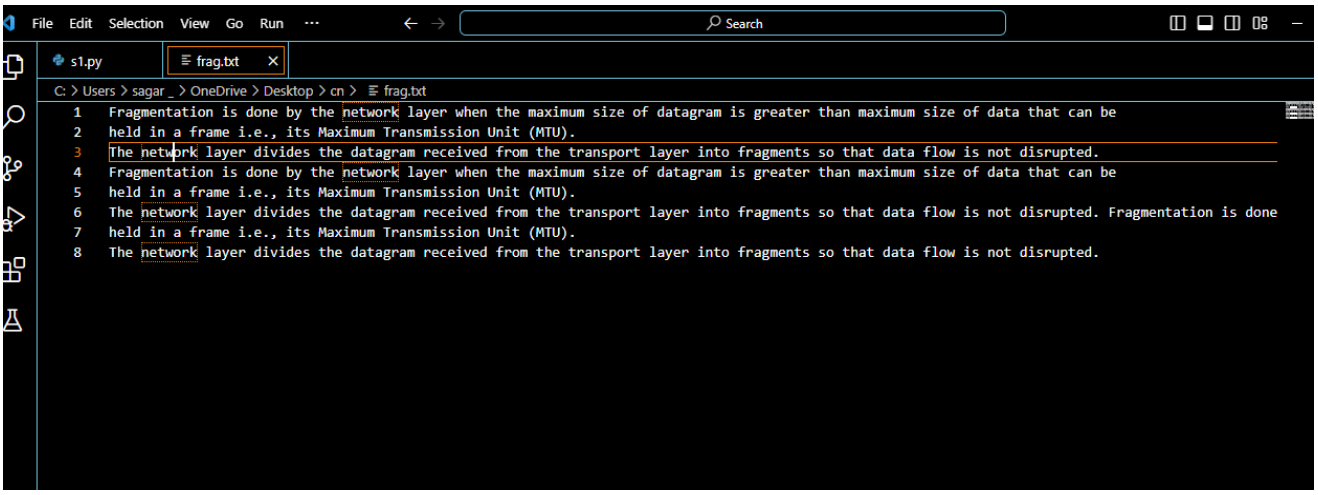
```python

def send_fragmented_data(file_path, mtu_size):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = ('localhost', 8889)

    try:
        client_socket.connect(server_address)
        print("Connected to server.")

        with open(file_path, 'r', encoding='utf-8') as file:
            data = file.read()

        print(f"Sending data of size {len(data)} bytes...")

        # Calculate the total number of fragments
        total_fragments = math.ceil(len(data) / mtu_size)

        # Calculate fragment sizes
        fragment_sizes = [mtu_size] * (total_fragments - 1)
        fragment_sizes.append(len(data) - (total_fragments - 1) * mtu_size)

        for i, fragment_size in enumerate(fragment_sizes):
            offset = calculate_offset(i + 1, fragment_sizes)
            identification = 123  # Common identification number for all fragments
            flags = 1 if i < total_fragments - 1 else 0  # Set flag M
            header = f"{identification}/{total_fragments}/{i}/{flags}/{offset}".encode('utf-8')
            fragment_data = data[i * mtu_size: (i + 1) * mtu_size]
            packet = header + b'|' + fragment_data.encode('utf-8')
            client_socket.send(packet)
            print(f"Sent fragment {i + 1} of size {len(fragment_data)} bytes.")

```

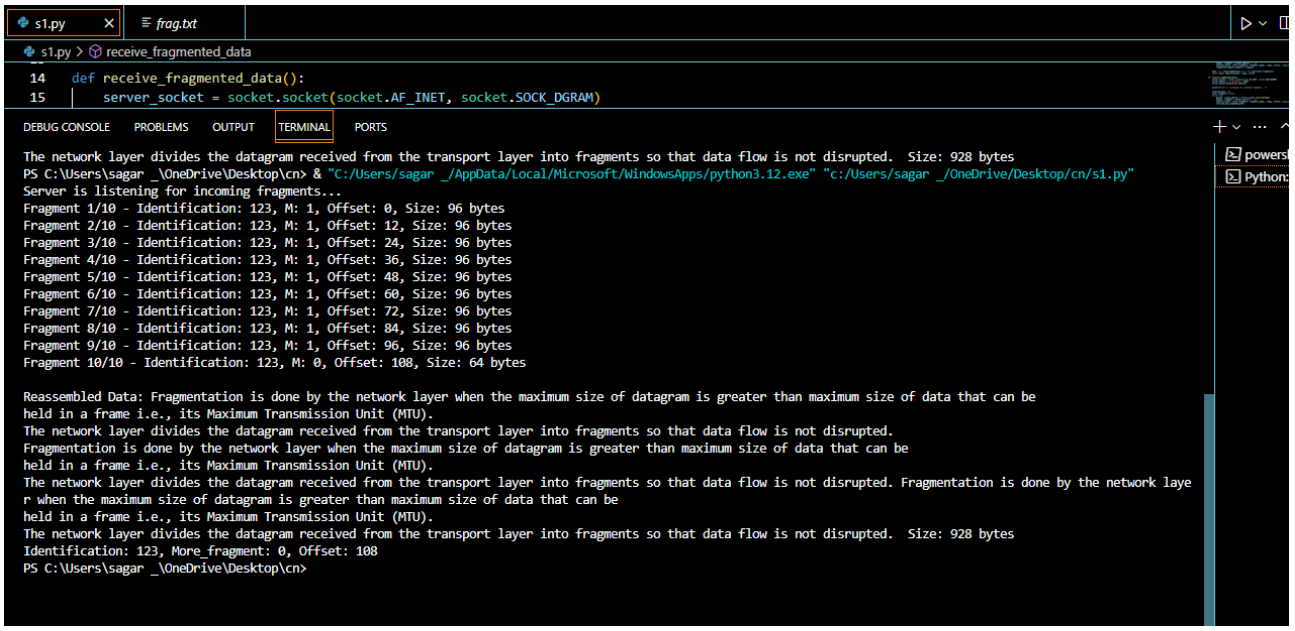
s1.py | frag.txt

s1.py > receive_fragmented_data

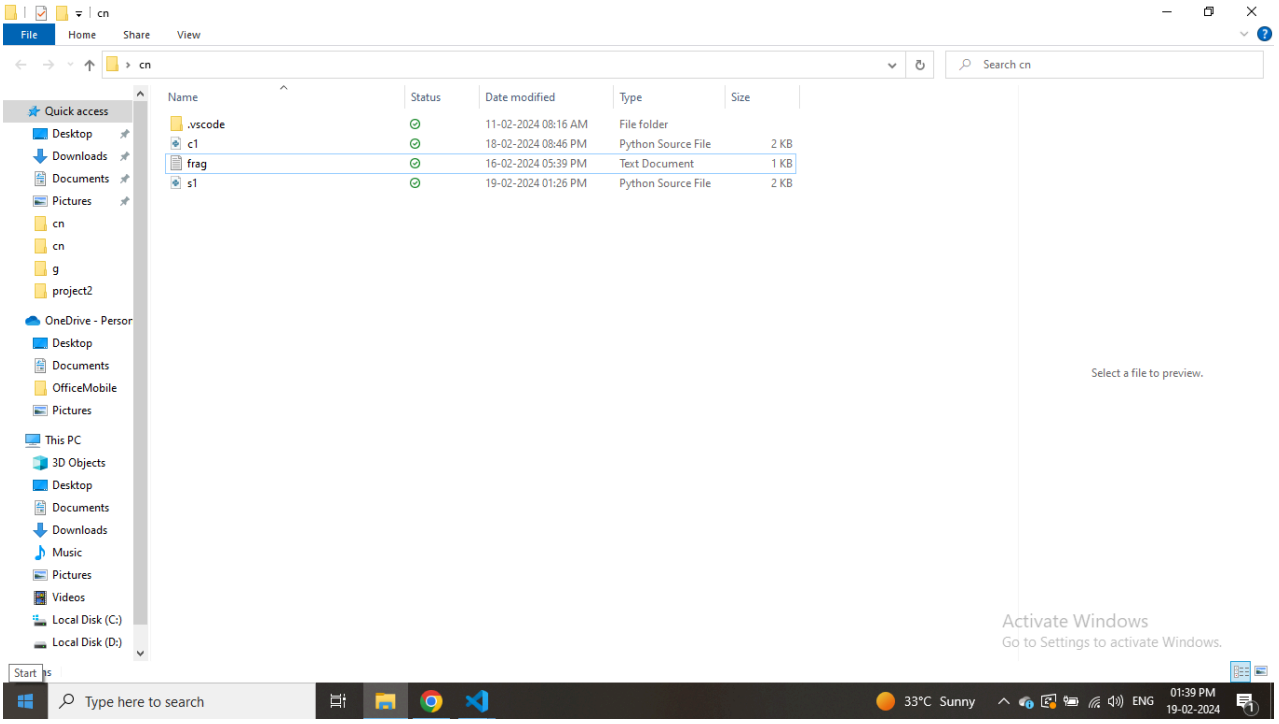
```python
def receive_fragmented_data():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = ('localhost', 8889)
    server_socket.bind(server_address)

    print("Server is listening for incoming fragments...")

    received_data = []
    total_fragments = None
    while True:
        packet, client_address = server_socket.recvfrom(1024)
        header, fragment = packet.split(b'|', 1)
        identification, total_fragments, fragment_number, flags, offset = map(int, header.split(b'/'))
        received_data.append(packet)

        # creation of files for each fragment
        with open(f"fragment_{fragment_number}.txt", "wb") as file:
            file.write(fragment)

        # Display identification, flags, offset, and size of data received in each fragment
        fragment_size = len(fragment)
        print(f"Fragment {fragment_number + 1}/{total_fragments} - Identification: {identification}, M: {flags}, Offset: {offset}, Size: {fragmen

        # Reassemble data when all fragments are received
        if len(received_data) == total_fragments:
            data, identification, flags, offset = reassemble_data(received_data)
            print(f"\nReassembled Data: {data.decode()} Size: {len(data)} bytes")
            print(f"Identification: {identification}, More_fragment: {flags}, Offset: {offset}")

            break

```

```
DEBUG CONSOLE   PROBLEMS   OUTPUT   TERMINAL   PORTS

PS C:\Users\sagar _> & "C:/Users/sagar _/AppData/Local/Microsoft/WindowsApps/python3.12.exe" "c:/Users/sagar _/cn/c1.py"
enter file path and size:
oneDrive\Desktop\cn\frag.txt
100
Connected to server.
Sending data of size 928 bytes...
Sent fragment 1 of size 96 bytes.
Sent fragment 2 of size 96 bytes.
Sent fragment 3 of size 96 bytes.
Sent fragment 4 of size 96 bytes.
Sent fragment 5 of size 96 bytes.
Sent fragment 6 of size 96 bytes.
Sent fragment 7 of size 96 bytes.
Sent fragment 8 of size 96 bytes.
Sent fragment 9 of size 96 bytes.
Sent fragment 10 of size 64 bytes.
Data sent successfully.
PS C:\Users\sagar _>
```

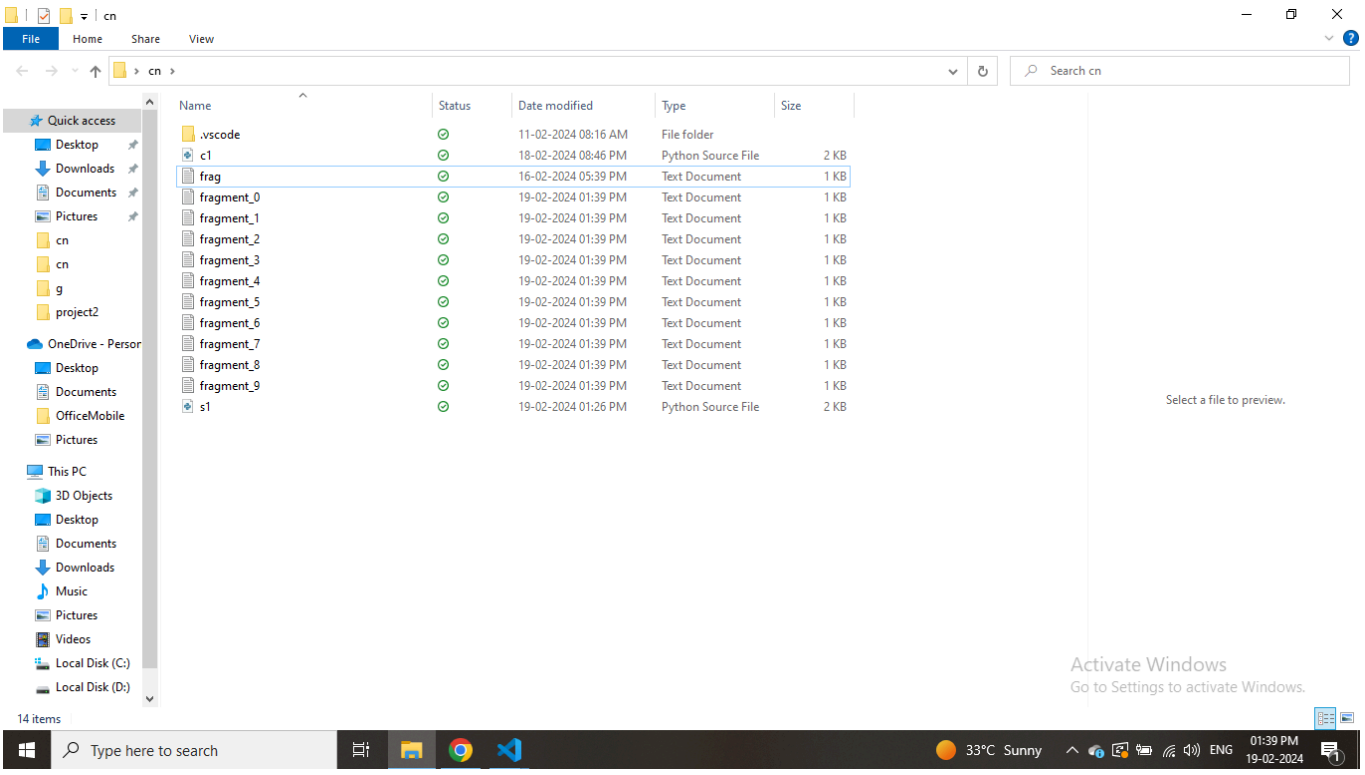


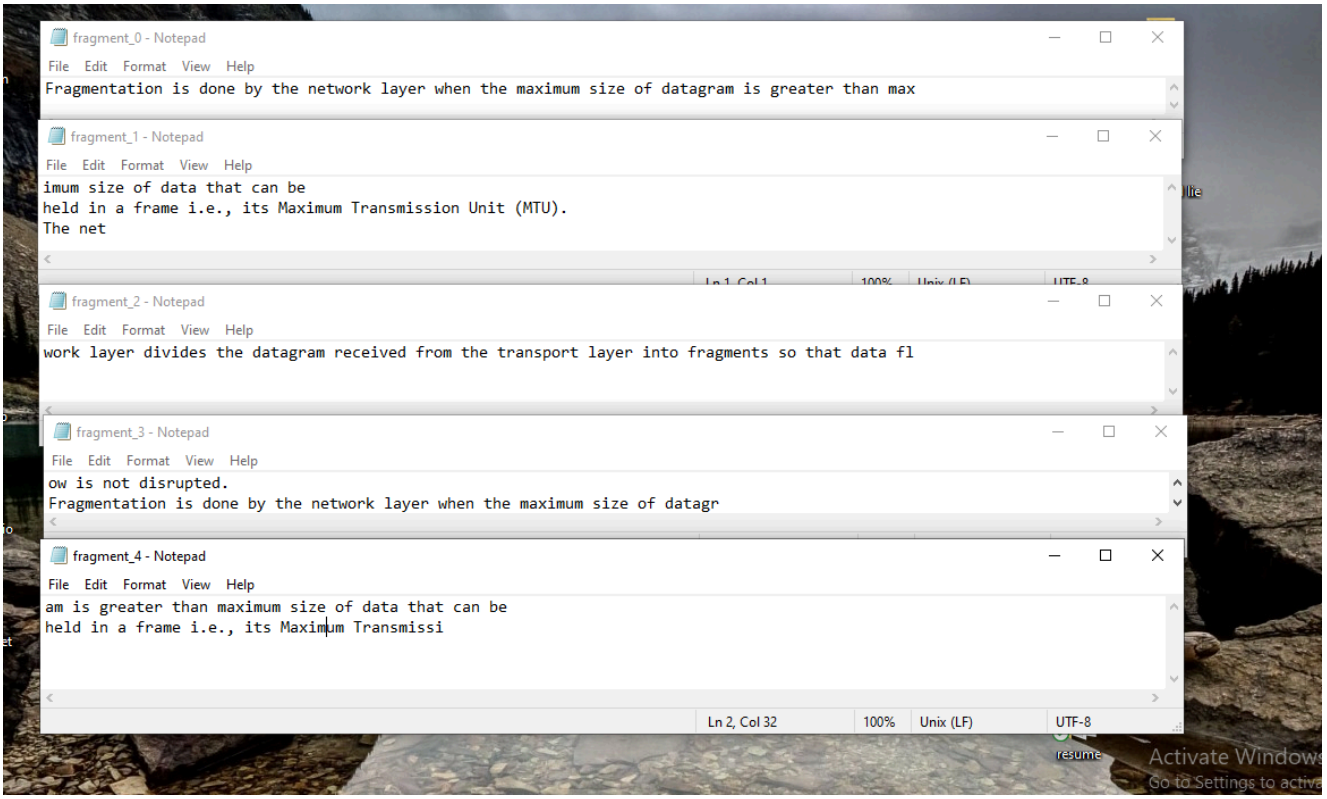


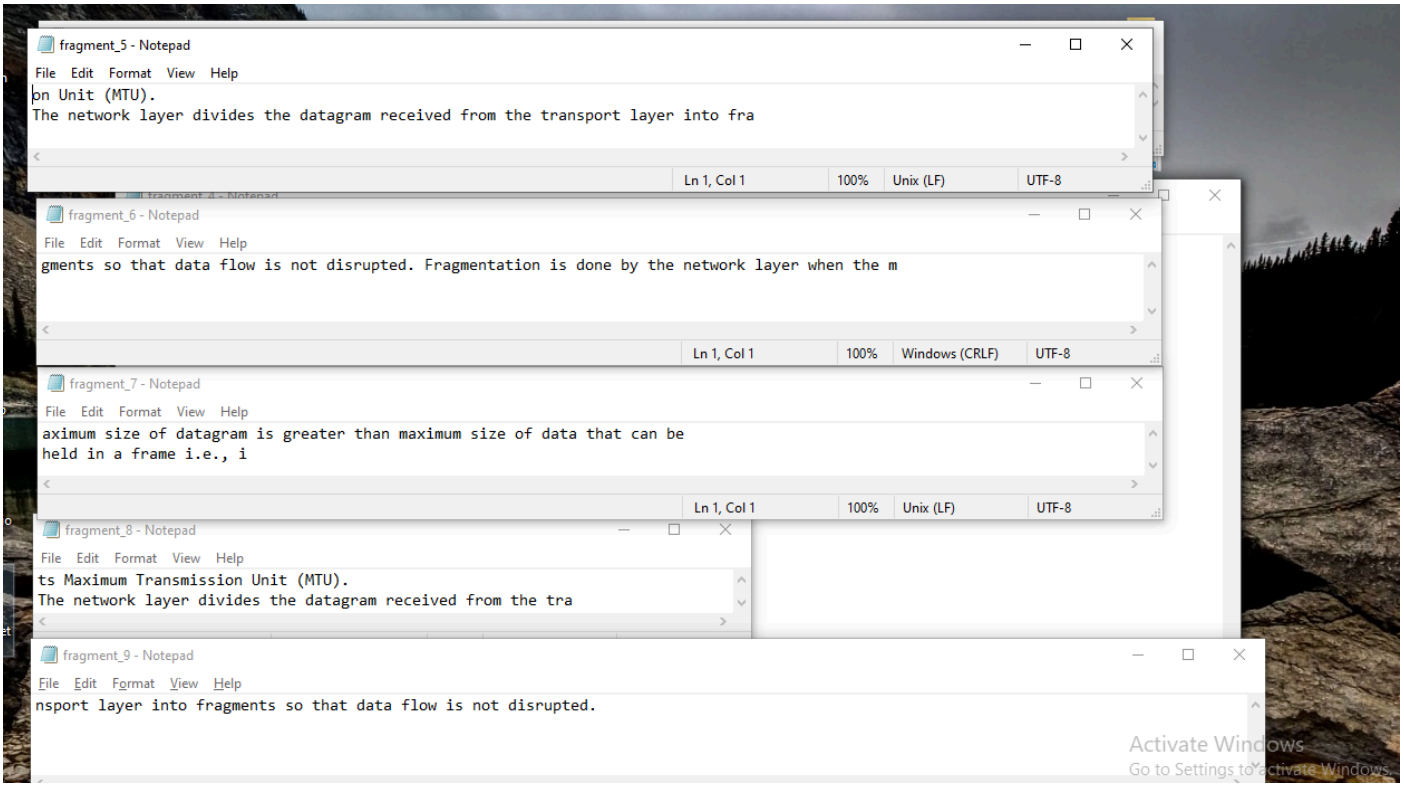
FOLDER STATE BEFORE CREATION->



10 TXT FILE CREATION:




fragment_0 - Notepad
File Edit Format View Help
Fragmentation is done by the network layer when the maximum size of datagram is greater than max
fragment_1 - Notepad
File Edit Format View Help
imum size of data that can be
held in a frame i.e., its Maximum Transmission Unit (MTU).
The net
fragment_2 - Notepad
File Edit Format View Help
work layer divides the datagram received from the transport layer into fragments so that data fl
fragment_3 - Notepad
File Edit Format View Help
ow is not disrupted.
Fragmentation is done by the network layer when the maximum size of datagr
fragment_4 - Notepad
File Edit Format View Help
am is greater than maximum size of data that can be
held in a frame i.e., its Maximum Transmissi
Ln 2, Col 32
100%
Unix (LF)
UTF-8
Activate Windows
Go to Settings to activa


fragment_5 - Notepad
File Edit Format View Help
on Unit (MTU).
The network layer divides the datagram received from the transport layer into fra
Ln 1, Col 1
100%
Unix (LF)
UTF-8
fragment_6 - Notepad
File Edit Format View Help
gments so that data flow is not disrupted. Fragmentation is done by the network layer when the m
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8
fragment_7 - Notepad
File Edit Format View Help
aximum size of datagram is greater than maximum size of data that can be
held in a frame i.e., i
Ln 1, Col 1
100%
Unix (LF)
UTF-8
fragment_8 - Notepad
File Edit Format View Help
ts Maximum Transmission Unit (MTU).
The network layer divides the datagram received from the tra
fragment_9 - Notepad
File Edit Format View Help
nsport layer into fragments so that data flow is not disrupted.
Activate Windows
Go to Settings to activate Windows.

References

[1] Python Documentation - Socket Programming

[2] RFC 791 - Internet Protocol (IPv4)

[3] https://www.geeksforgeeks.org/fragmentation-network-layer/