

Q1 Exercise 3

3 Points

Q1.1 maximum iperf3 payload size

1 Point

What is the maximum `iperf3` payload size (e.g. largest `-l` argument) that can be sent without IP fragmentation?

(Make sure to specify units!)

1472 bytes

Show the `iperf3` command you ran that generates packets with this payload size, and show evidence from `tcpdump` or Wireshark output to support your answer. How can you tell that there is no IP fragmentation?

from the tcpdump result, we can see the last packet is a UDP packet and it is not an ip-proto-17 packet which is an IP fragmentation

▼ iperf.png

Download

```
ty2069@juliet:~$ iperf3 -c romeo -u -k 1 -l 1472
Connecting to host romeo, port 5201
[ 4] local 10.10.0.101 port 52663 connected to 10.10.0.100 port 5201
[ ID] Interval      Transfer    Bandwidth   Total Datagrams
[ 4] 0.00-0.00 sec  1.44 KBytes 51.9 Mbits/sec 1
-----
[ ID] Interval      Transfer    Bandwidth   Jitter      Lost/Total Datagrams
[ 4] 0.00-0.00 sec  1.44 KBytes 51.9 Mbits/sec 0.000 ms    0/0 (-nan%)
[ 4] Sent 0 datagrams

iperf Done.
```

▼ tcpdump-1472.png

Download

```
23:40:03.299958 02:1a:28:86:1d:c9 > 02:2f:40:02:02:6c, ethertype IPv4 (0x0800),
length 46: 10.10.0.101.52663 > 10.10.0.100.5201: UDP, length 4
23:40:03.300500 02:2f:40:02:02:6c > 02:1a:28:86:1d:c9, ethertype IPv4 (0x0800),
length 46: 10.10.0.100.5201 > 10.10.0.101.52663: UDP, length 4
23:40:03.339959 02:1a:28:86:1d:c9 > 02:2f:40:02:02:6c, ethertype IPv4 (0x0800),
length 1514: 10.10.0.101.52663 > 10.10.0.100.5201: UDP, length 1472
```

Explain the maximum `iperf3` payload size in terms of MTU and header lengths. What headers are appended to the `iperf3` payload at each layer of the network stack? What size is each header?

Compare the MTU to the sum of payload + header lengths.

The maximum iperf3 payload size is 1472B for each packet. IP header which is 8B is appended to the data at the transport layer and the UDP header which is 20B is appended to the data at the network layer. So the MTU is equal to data size plus two headers $1472 + 20 + 8 = 1500\text{B}$.

Q1.2 avoiding IP fragmentation

1 Point

Which condition is necessary and sufficient to avoid IP fragmentation?

- ☐ Sum of application layer header (if any), application layer payload, and transport layer header should not be greater than the MTU.
- ☒ Sum of application layer header (if any), application layer payload, transport layer header, and IP layer header, should not be greater than the MTU.
- ☐ Sum of application layer header (if any), application layer payload, transport layer header, IP layer header, and Ethernet (or other Layer 2) header and FCS, should not be greater than the MTU.

Q1.3 fragmentation header fields

0.5 Points

Explain the tcpdump output for the `iperf3` flow with `-l 2048` in terms of the IP header fields (i.e., id, offset, flags, length) that are used in fragmentation.

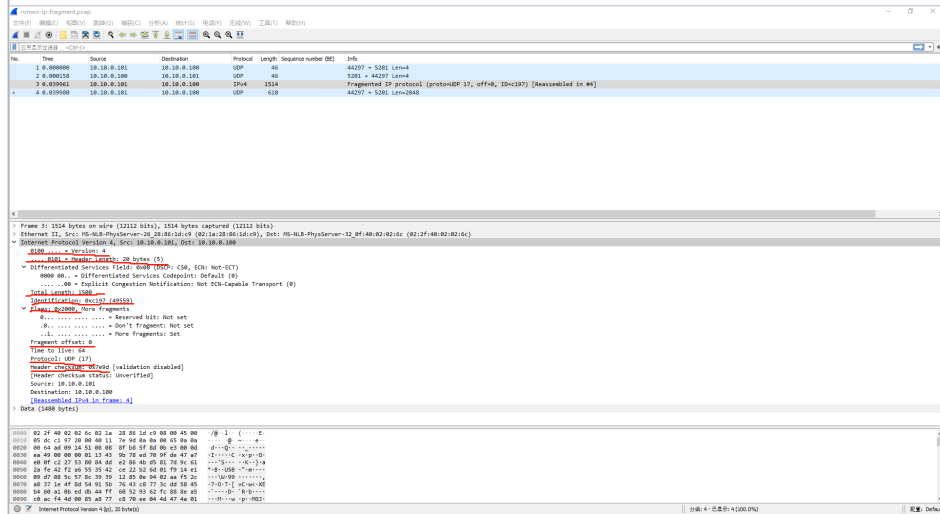
Upload Wireshark screenshots showing the IP header details for both fragments. Annotate the screenshots to draw a circle or box around each of the header fields mentioned above.

Explain *why* each of these header fields has the value it does.

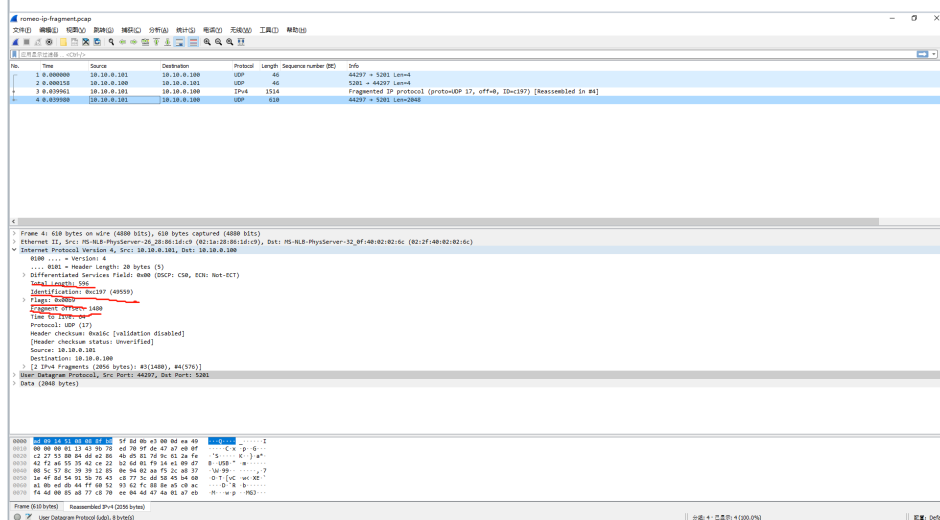
Identification is c197 which is the same in all fragments.
Fragment offsets are 0 for the first fragment and 1480 for the

second fragment which is the offset of the current fragment in the original datagram. flags contain reserve bits which must be 0, Don't fragment bit which is not set, and More fragment bit which is set for the first fragment and not set for the second fragment. and the Length is the total length of the payload plus ip and udp header.

▼ wireshark-ip-fragmentation.png

[Download](#)


▼ wireshark-ip-fragmentation2.png

[Download](#)


Q1.4 transport protocol header in fragments

0.5 Points

When IP fragmentation occurs, only one of the fragments has the UDP header.

Upload Wireshark screenshots showing two fragments of the same packet, and show that only one has the UDP header.

— fragment1.png

▼ fragment1.png Download

romeo-ip-fragment.pcap

No.	Time	Source	Destination	Protocol	Length	Sequence number (BE)	Info
1	0.000000	10.10.0.101	10.10.0.100	UDP	46	44297 → 5201 Len=4	
2	0.000158	10.10.0.100	10.10.0.101	UDP	46	5201 → 44297 Len=4	
3	0.039961	10.10.0.101	10.10.0.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=197) [Reassembled in #4]	
4	0.039980	10.10.0.101	10.10.0.100	UDP	610	44297 → 5201 Len=2048	

Frame 3: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)

- Ethernet II, Src: MS-MLB-PhysServer-26_28:86:1d:c9 (02:1a:28:86:1d:c9), Dst: MS-MLB-PhysServer-32_0f:40:02:02:6c (02:2f:40:02:02:6c)
- Internet Protocol Version 4, Src: 10.10.0.101, Dst: 10.10.0.100
- Data (1480 bytes)

0010 05 dc c1 97 20 00 40 11 7e 9d 0a 0a 00 65 0a 0a@.....
 0020 00 64 ad 09 14 51 08 08 8f b8 5f 8d 0b e3 00 0dQ.....
 0030 ea 49 00 00 00 01 13 43 9b 78 ed 70 9f de 47 a7C.X.p-6
 0040 e0 0f c2 27 53 80 84 dd e2 86 4b d5 81 7d 9c 61S...K...a
 0050 2a fe 42 f2 a6 55 35 42 ce 22 b2 6d 01 f9 14 e1 *.B..USB...m...
 0060 09 d7 00 5c 57 8c 39 39 12 05 0e 94 02 aa f5 2c ...W.99.....
 0070 a8 37 1e 4f 8d 54 91 5b 76 43 c8 77 3c dd 58 45 7.O.T[vC.wc.XE

Identification (p.id), 2 byte(s)

▼ fragment2.png Download

romeo-ip-fragment.pcap

No.	Time	Source	Destination	Protocol	Length	Sequence number (BE)	Info
1	0.000000	10.10.0.101	10.10.0.100	UDP	46	44297 → 5201 Len=4	
2	0.000158	10.10.0.100	10.10.0.101	UDP	46	5201 → 44297 Len=4	
3	0.039961	10.10.0.101	10.10.0.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=197) [Reassembled in #4]	
4	0.039980	10.10.0.101	10.10.0.100	UDP	610	44297 → 5201 Len=2048	

Frame 4: 610 bytes on wire (4880 bits), 610 bytes captured (4880 bits)

- Ethernet II, Src: MS-MLB-PhysServer-26_28:86:1d:c9 (02:1a:28:86:1d:c9), Dst: MS-MLB-PhysServer-32_0f:40:02:02:6c (02:2f:40:02:02:6c)
- Internet Protocol Version 4, Src: 10.10.0.101, Dst: 10.10.0.100
- User Datagram Protocol, Src Port: 44297, Dst Port: 5201
 - Source Port: 44297
 - Destination Port: 5201
 - Length: 2056
 - Checksum: 0x0fb8 [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 0]
 - [Timestamps]
- Data (2048 bytes)

0010 02 54 c1 97 00 b9 40 a1 6c 0a 0a 00 65 0a 0a T...-...
 0020 00 64 9b f0 6e 51 e7 0d 59 31 5a 9f 16 85 76 25 d...nQ...Y1Z...v%
 0030 d7 96 47 29 4e 63 0d ca ea 1c d1 10 e5 ad cc 99 ...G)Nc...
 0040 c6 68 8a 34 b9 71 41 12 a3 9b b2 b9 20 28 de f7 h.4.qA...(-
 0050 be 25 21 8c 09 2e d6 73 4b a8 83 38 55 58 c9 1b Xl...s K...BUP...

Frame (610 bytes) Reassembled IPv4 (2056 bytes)

Identification (p.id), 2 byte(s)

List the size of each of the two fragments. (You can describe the size in terms of the entire frame with Ethernet header, the IP packet with IP header, or the IP payload without IP header - just make sure to specify which size you are using.)

How can you tell from the sizes alone, without inspecting the packet contents, that the UDP header is not duplicated?

the first fragment is 1514B which equal to an ethernet header with 14B plus an IP header with 20B and data size 1480B. The second fragment is 610B which equal to an ethernet header with 14B plus an IP header with 20B plus UDP header and data size 576B.

Sum up the data size we have 2056 which equals a payload of 2048B plus a UDP header with size 8B.

Q2 Exercise 4

1 Point

What is the maximum size of the `iperf3` UDP *payload* (not including headers) that the system can send, even when fragmentation is allowed?

(Make sure to specify units!)

65507 bytes

Show a screenshot of `iperf3` output that supports your answer (or copy and paste from your terminal). (What happens if you try to send a payload that is larger than this value?)

If I send a payload larger than this value, it will return a parameter error saying the block size too large and won't send the packet at all.

▼ error.png

Download

```
ty2069@juliet:~$ iperf3 -c romeo -u -k 1 -l 1048576
iperf3: parameter error - block size too large (maximum = 65507 bytes)

Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]
```

Explain this value in terms of the header sizes and the "length" header field:

How many bits are allocated for the "length" field in the IP header?

16 bits

What is the maximum value that this field can hold?

it can hold $2^{16} - 1 = 65535$ bytes

How many bytes are reserved for the IP header?

20 bytes

How many bytes are reserved for the UDP header?

8 bytes

Show how this limits the maximum size of the UDP payload.

65535 - 20 bytes(IP header) - 8 bytes(UDP header) = 65507 bytes

Q3 Exercise 8

2 Points

Q3.1 tftp packets

1 Point

Examine the TFTP header in your captured packets, and list all the different types of packets exchanged during the TFTP session (note the value of the opcode field for each). Compare them with the TFTP message format in Fig. 5.3 in the textbook.

Show a Wireshark screenshot of the TFTP header for each type of TFTP packet. Annotate each screenshot by drawing a circle or a box around the `opcode` field.

Read Request with Opcode: 1
Data packet with Opcode: 3
Acknowledgment with Opcode: 4

▼ tftp-ack.png

Download

The screenshot shows a Wireshark capture of a TFTP session. The packet list pane highlights packet 4, which is an Acknowledgement (Opcode: Acknowledgement (4)) from 10.10.0.101 to 10.10.0.100. The packet details pane shows the source file as 'small.bin' and the full block number as 1. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Sequence number (BE)	Info
1	0.000000	10.10.0.101	10.10.0.100	TFTP	60		Read Request, File: small.bin, Transfer type: octet
2	0.011815	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 1
3	0.012795	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 1
4	0.012828	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 2
5	0.013500	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 2
6	0.013612	10.10.0.100	10.10.0.101	TFTP	46		Data Packet, Block: 3 (last)
7	0.014138	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 3
8	5.014612	MS-NLB-PhysServer-3...	MS-NLB-PhysServer-2...	ARP	42		Who has 10.10.0.101? Tell 10.10.0.100
9	5.015819	MS-NLB-PhysServer-2...	MS-NLB-PhysServer-3...	ARP	42		10.10.0.101 is at 02:1a:28:86:1d:c9

▼ tftp-data-packet.png

Download

The screenshot shows a Wireshark capture of a TFTP session. The packet list pane highlights packet 3, which is a Data Packet (Opcode: Data Packet (3)) from 10.10.0.101 to 10.10.0.100. The packet details pane shows the source file as 'small.bin' and the full block number as 1. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Sequence number (BE)	Info
1	0.000000	10.10.0.101	10.10.0.100	TFTP	60		Read Request, File: small.bin, Transfer type: octet
2	0.011815	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 1
3	0.012795	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 1
4	0.012828	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 2
5	0.013500	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 2
6	0.013612	10.10.0.100	10.10.0.101	TFTP	46		Data Packet, Block: 3 (last)
7	0.014138	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 3
8	5.014612	MS-NLB-PhysServer-3...	MS-NLB-PhysServer-2...	ARP	42		Who has 10.10.0.101? Tell 10.10.0.100
9	5.015819	MS-NLB-PhysServer-2...	MS-NLB-PhysServer-3...	ARP	42		10.10.0.101 is at 02:1a:28:86:1d:c9

▼ tftp-read-request.png

Download

The screenshot shows a Wireshark capture of a TFTP session. The packet list pane highlights packet 1, which is a Read Request (Opcode: Read Request (1)) from 10.10.0.101 to 10.10.0.100. The packet details pane shows the source file as 'small.bin' and the type as octet. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Sequence number (BE)	Info
1	0.000000	10.10.0.101	10.10.0.100	TFTP	60		Read Request, File: small.bin, Transfer type: octet
2	0.011815	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 1
3	0.012795	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 1
4	0.012828	10.10.0.100	10.10.0.101	TFTP	558		Data Packet, Block: 2
5	0.013500	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 2
6	0.013612	10.10.0.100	10.10.0.101	TFTP	46		Data Packet, Block: 3 (last)
7	0.014138	10.10.0.101	10.10.0.100	TFTP	46		Acknowledgement, Block: 3
8	5.014612	MS-NLB-PhysServer-3...	MS-NLB-PhysServer-2...	ARP	42		Who has 10.10.0.101? Tell 10.10.0.100
9	5.015819	MS-NLB-PhysServer-2...	MS-NLB-PhysServer-3...	ARP	42		10.10.0.101 is at 02:1a:28:86:1d:c9

Q3.2 tftp ports

0.5 Points

Why does the server's port number change during the TFTP session? What kind of data is sent using port 69, and what kind of data is sent using the new port number?

TFTP server uses UDP port 69 for the TFTP control message and A different ephemeral port number is used by the server for data transfer. So the port number changed. Ack and the data blocks are sent using the new port number.

Q3.3 transferring large files

0.5 Points

In a previous exercise, we found the maximum size of a UDP datagram on this network. With TFTP, which uses UDP, we transferred a file larger than the maximum UDP datagram size. How do you explain this?

TFTP divides the file into multiple UDP packet and reassembles the packets as the file at the receiver side.

Q4 UDP as a connectionless transport protocol

2 Points

Q4.1 RTT of UDP and TCP ping

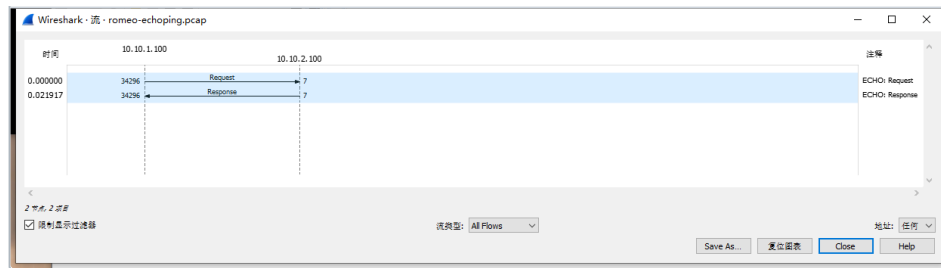
2 Points

Show the output of the `echoping` command and the Wireshark flow graph for the UDP echo.

```
ty2069@romeo:~$ echoping -f x -u 10.10.2.100
Elapsed time: 0.022057 seconds
```

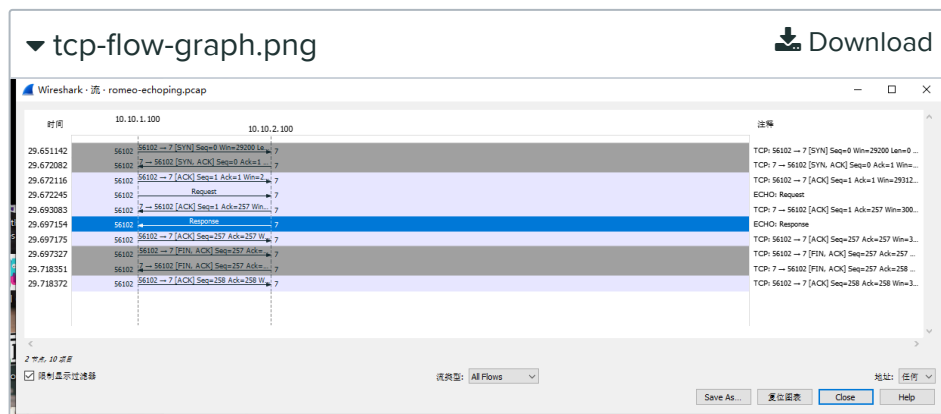
▼ udp-flow-graph.png

 Download



Show the output of the `echoping` command and the Wireshark flow graph for the TCP echo.

```
ty2069@romeo:~$ echoping -f x 10.10.2.100
Elapsed time: 0.046256 seconds
```



For the echo with connection establishment (TCP),

How much time elapses from when romeo starts to establish the connection (the time of the first packet in the TCP flow graph) until romeo actually sends the echo request (time of the echo request in the TCP flow graph)?

$$29.672245 - 29.651142 = 0.021103 \text{ sec}$$

How much time elapses from when romeo starts to establish the connection (the time of the first packet in the TCP flow graph) until romeo receives the echo response from juliet (time of the echo response in the TCP flow graph)? This number should be similar to the output of the `echoping` command, although it may be slightly smaller because it does not include some application layer overhead.

$$29.697154 - 29.651142 = 0.046012 \text{ sec}$$

What percent of the total round trip time (from the output of the `echoping` command) is due to the connection establishment, before the echo request is even sent?

$0.046256/0.06723*100\% = 68.8\%$

In the UDP case, what percent of the total round trip time (from the output of the `echoping` command) is due to the connection establishment, before the echo request is even sent?

0%

Q5 UDP sockets

2 Points

Q5.1 socket functions that cause packets to appear on network

1 Point

Which of these UDP socket functions will cause one or more packets to appear on the network link? Select all that apply.

☐ Creating a socket

☐ `bind`

☒ `sendto`

☐ `recvfrom`

Show evidence for your answer from your experiment for each choice you selected - show a screenshot of a message in your the `tcpdump` output and the Python command that triggered it. If your screenshot includes multiple Python commands, circle the one that caused the message to be sent.

▼ sendto.png

Download

```
>>> sent = sock.sendto(b'Hello', ('10.10.2.100', 4000))
```

▼ tcpdump.png

Download

```
02:38:54.946555 02:98:b9:cb:f4:38 > 02:f9:c8:b7:3c:83, ethertype IPv4 (0x0800),
length 47: (tos 0x0, ttl 63, id 31572, offset 0, flags [DF], proto UDP (17), len
gth 33)
10.10.1.100.60980 > 10.10.2.100.4000: UDP, length 5
0x0000: 4500 0021 7b54 4000 3f11 a89c 0a0a 0164 E...!{T@.?......d
0x0010: 0a0a 0264 ee34 0fa0 000d 17fa 4865 6c6c ...d.4.....Hell
0x0020: 6f
o
```

Q5.2 socket functions that reserve IP+port

1 Point

Which of these UDP socket functions will assign a transport layer port on one or more IP addresses to the socket that it is called on, so that no other socket can bind to the same address and port?

☐ Creating a socket

☒ bind

☒ sendto (if the socket is not previously bound)

☐ recvfrom

Show evidence for your answer from your experiment for each choice you selected - show output of `lsof` and the Python command that triggered it. If your screenshot includes multiple Python commands, circle the relevant one.

▼ bind.png

Download

```
>>> sock.bind(('10.10.2.100', 4000))
```

▼ losf1.png

Download

```
ty2069@juliet:~$ lsof -n -i udp
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
python3 5100 ty2069 3u IPv4 24184 0t0 UDP 10.10.2.100:4000
```

▼ losf2.png

Download

```
ty2069@romeo:~$ lsof -n -i udp
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
python3 4989 ty2069 3u IPv4 23653 0t0 UDP *:60980
```

▼ sendto.png

Download

```
>>> sent = sock.sendto(b'Hello', ('10.10.2.100', 4000))
```

Lab 5: UDP and its Applications

● **UNGRADED**

STUDENT

Tingyu Yang

TOTAL POINTS

- / 10 pts

QUESTION 1

Exercise 3 3 pts

1.1 maximum iperf3 payload size 1 pt

1.2 avoiding IP fragmentation 1 pt

1.3 fragmentation header fields 0.5 pts

1.4 transport protocol header in fragments 0.5 pts

QUESTION 2

Exercise 4 1 pt

QUESTION 3

Exercise 8 2 pts

3.1 tftp packets 1 pt

3.2 tftp ports 0.5 pts

3.3 transferring large files 0.5 pts

QUESTION 4

UDP as a connectionless transport protocol 2 pts

4.1 RTT of UDP and TCP ping 2 pts

QUESTION 5

UDP sockets 2 pts

5.1 socket functions that cause packets to appear on network 1 pt

5.2 socket functions that reserve IP+port 1 pt