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 Datag
rams
[  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 = 1500B.

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.

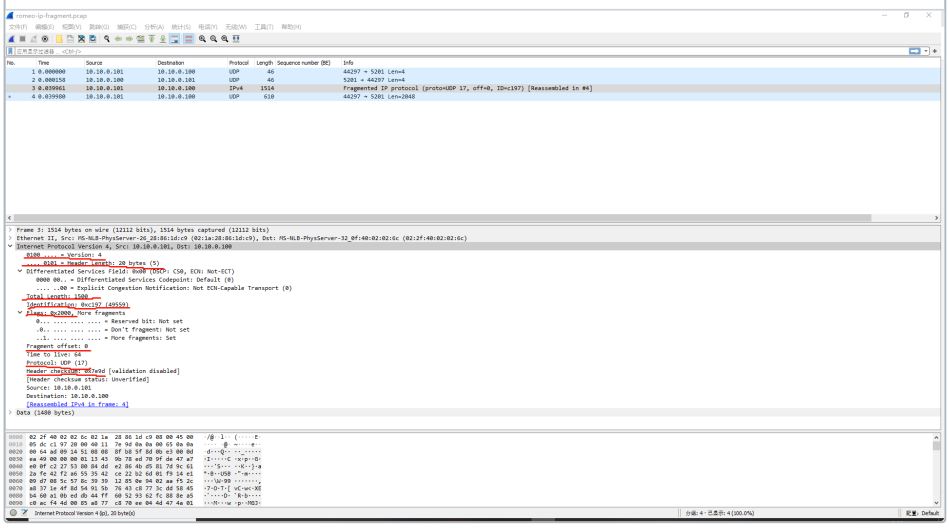
Upload Wireshark screenshots showing the IP header details for both fragments. Annotate the screenshots to draw a circle or box around each of the header fields mentioned above.

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

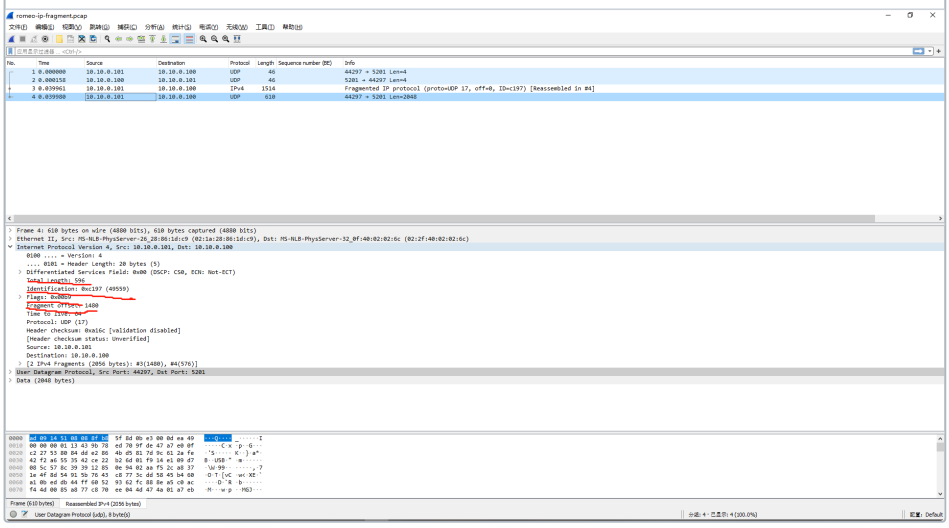
> Identification is c197 which is the same in all fragments. Fragment offsets are 0 for the first fragment and 1480 for the

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

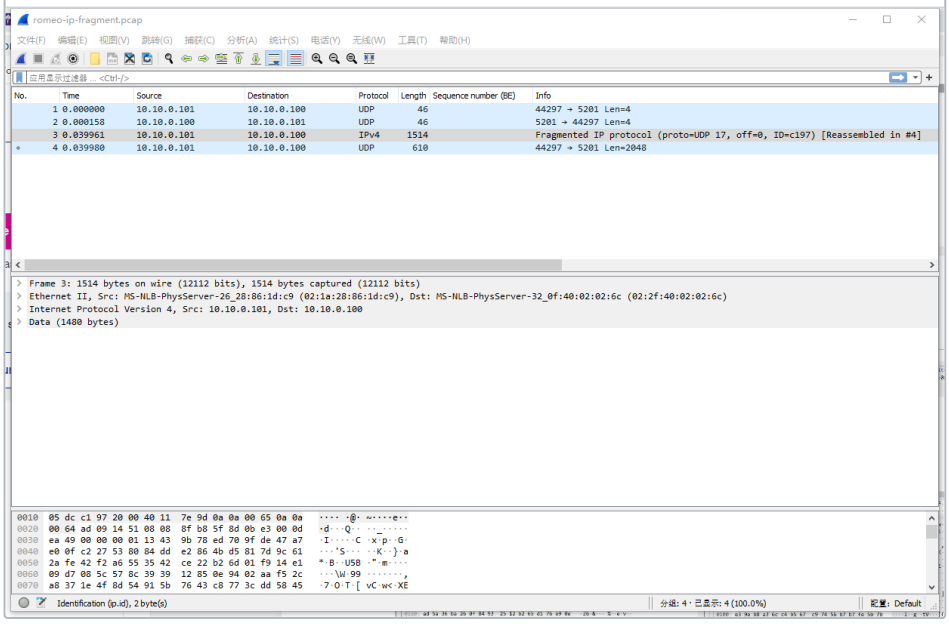
0.5 Points

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

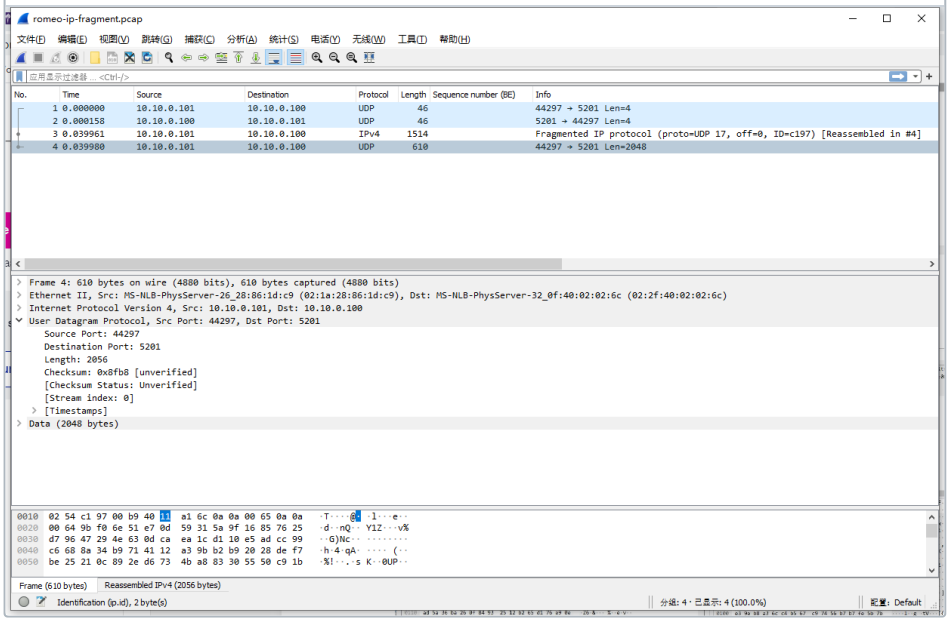
Upload Wireshark screenshots showing two fragments of the same packet, and show that only one has the UDP header.

fragment1.png

▾ fragment1.png Download



▾ fragment2.png Download



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?

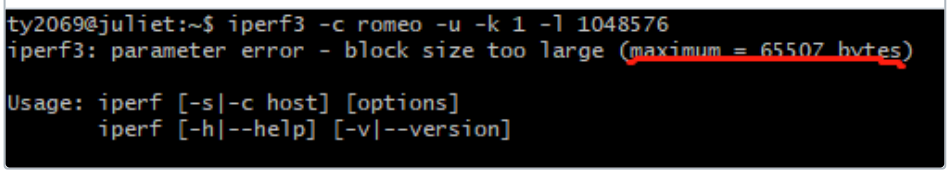
(Make sure to specify units!)

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



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

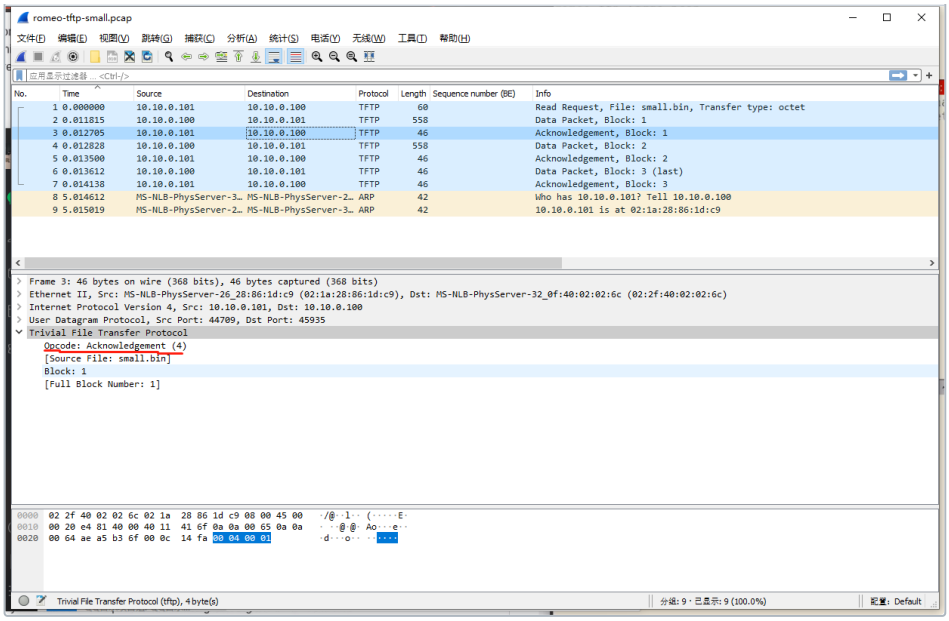
1 Point

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.

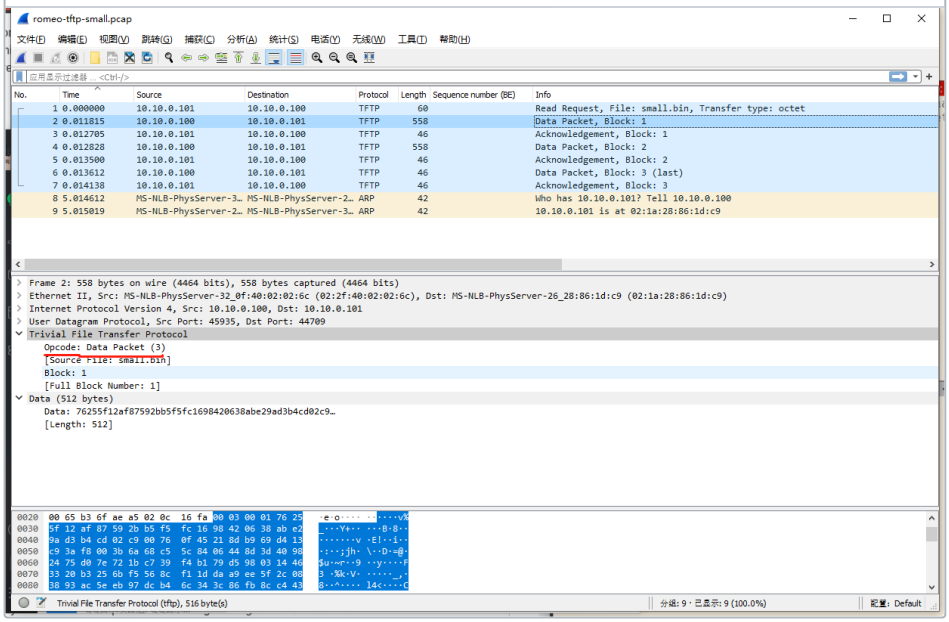
Read Request with Opcode: 1
Data packet with Opcode: 3
Acknowledgment with Opcode: 4

▾ tftp-ack.png Download



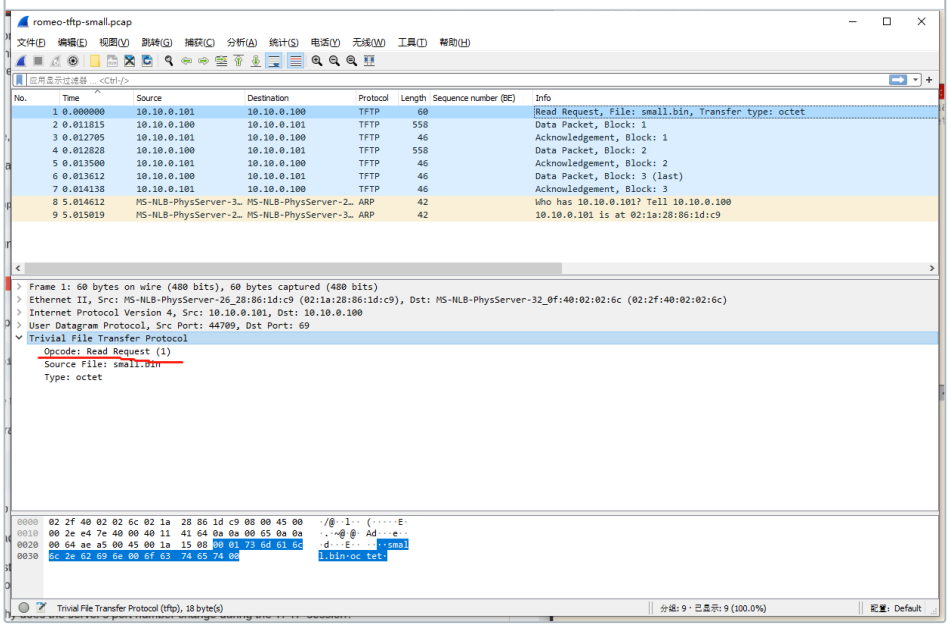
tftp-data-packet.png

Download



tftp-read-request.png

Download



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 UPD 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.

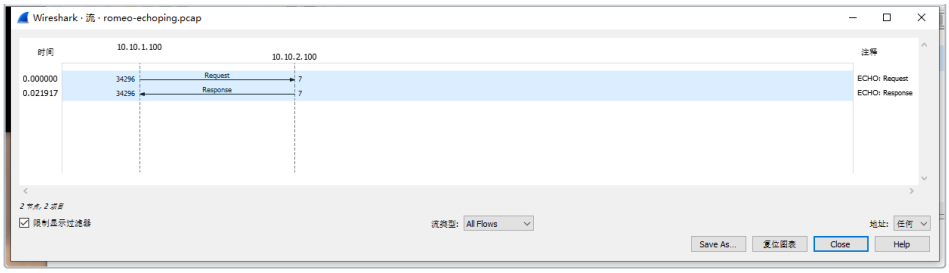
Q4 UDP as a connectionless transport protocol
2 Points

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
```

tcp-flow-graph.png Download

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 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 reponse 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 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`
- [x] `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
- [x] `bind`
- [x] `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