Part A:

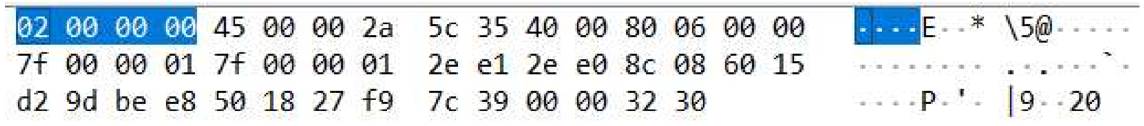
1. Yes they are.

2. Yes

   The TCP segment has the different messages clearly shown.

3. The client sends a total of 24 segments, of which the first is SYN, second is ACK, second last is [FIN, ACK] and last is ACK. The rest 20 are the 20 messages.

4. No, the lengths are not all the same. The length of the messages 9-20 is 2 each while it is 1 for each of 1-9. The below screenshot shows the application data for 20th message.

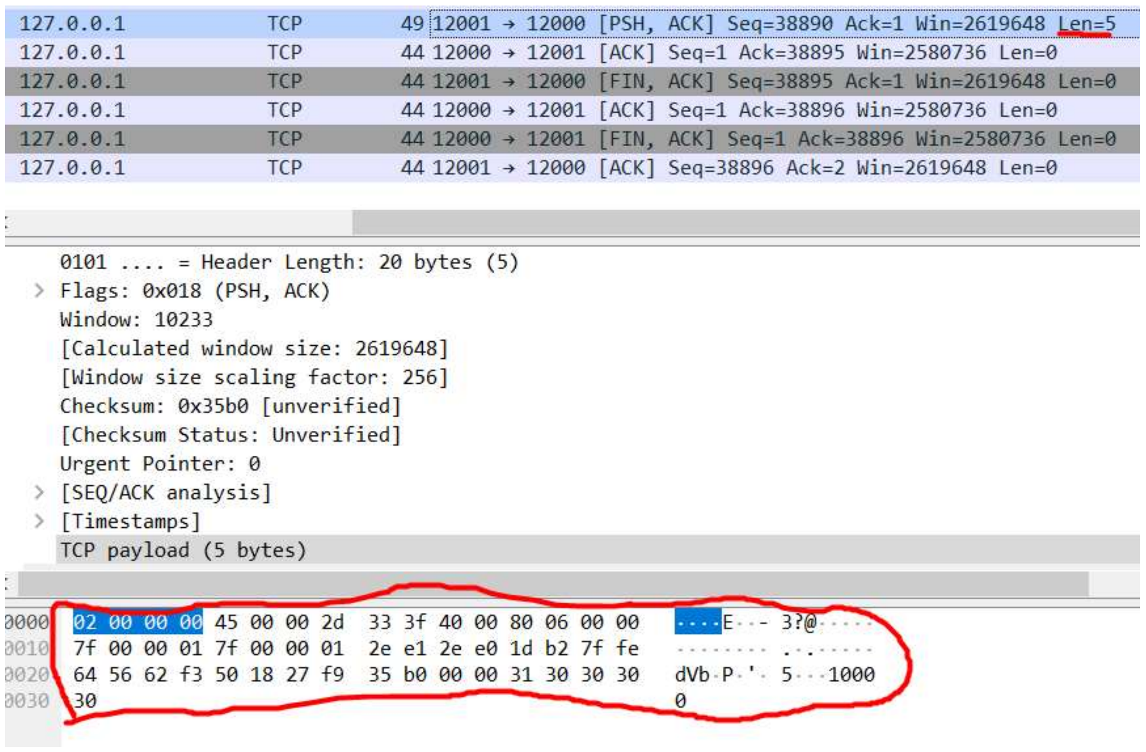


```
02 00 00 00 45 00 00 2a  5c 35 40 00 80 06 00 00   ····E··* \5@·····
7f 00 00 01 7f 00 00 01  2e e1 2e e0 8c 08 60 15   ········ .·.···`·
d2 9d be e8 50 18 27 f9  7c 39 00 00 32 30         ····P·'· |9··20
```

5. Code attached
6. The numbers differ and the difference depends on send_delay (61 calls to recv at the server). This shows that the server has encountered some erroneous recv calls which are the same as we are trying to catch. The error that we are trying to catch are 'Nonauthoritative host not found' and 'Resource temporarily unavailable' that correspond to codes **WSATRY_AGAIN** (11002) and **WSAEWOULDBLOCK** (10035) on windows.

Part B:

1. They are received in order
2. The message boundaries are no longer preserved
3. N/A
4. The maximum length is 5 bytes. The application data for this segment is:



```
127.0.0.1    TCP    49 12001 → 12000 [PSH, ACK] Seq=38890 Ack=1 Win=2619648 Len=5
127.0.0.1    TCP    44 12000 → 12001 [ACK] Seq=1 Ack=38895 Win=2580736 Len=0
127.0.0.1    TCP    44 12001 → 12000 [FIN, ACK] Seq=38895 Ack=1 Win=2619648 Len=0
127.0.0.1    TCP    44 12000 → 12001 [ACK] Seq=1 Ack=38896 Win=2580736 Len=0
127.0.0.1    TCP    44 12000 → 12001 [FIN, ACK] Seq=1 Ack=38896 Win=2580736 Len=0
127.0.0.1    TCP    44 12001 → 12000 [ACK] Seq=38896 Ack=2 Win=2619648 Len=0
```

```
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 10233
    [Calculated window size: 2619648]
    [Window size scaling factor: 256]
    Checksum: 0x35b0 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (5 bytes)
```

```
0000  02 00 00 00 45 00 00 2d  33 3f 40 00 80 06 00 00   ····E··- 3?@·····
0010  7f 00 00 01 7f 00 00 01  2e e1 2e e0 1d b2 7f fe   ········ ..······
0020  64 56 62 f3 50 18 27 f9  35 b0 00 00 31 30 30 30   dVb·P·'· 5···1000
0030  30                                                 0
```

5. Coded
6. Recv is called 665 times of which 356 are not erroneous calls (same as mentioned in part A-6). However, Wireshark shows 10000 TCP segments sent by the client. This is because message boundaries are not preserved in TCP and send_delay is too less to preserve any boundaries.

Comment:
TCP preserves the order of messages but it doesn't preserve message boundaries in general. However, if the delay between each message is large enough, message boundaries get preserved.

Part C:

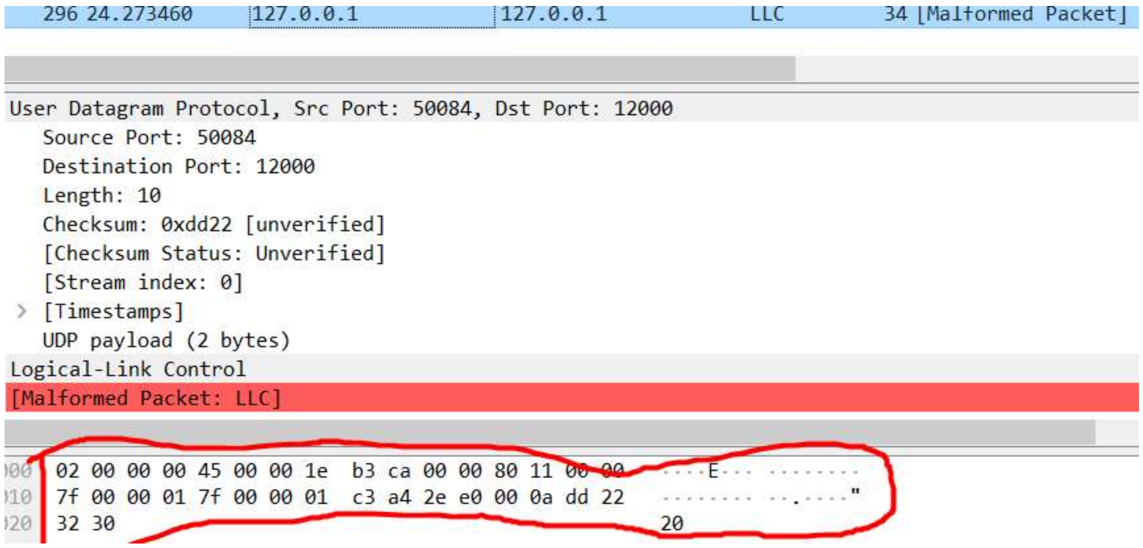
1. Padding: We can send a fixed amount of data each time. To apply this in our case, we need to find the largest length of TCP segment. We can add as many zeros before each message as are in the deficit of 5 bytes and then take the numerical value of the received message.
2. Prefixing the binary value of the message length: We can find the maximum length of message and then find the length of its binary equivalent. We would then prefix each message with its length in binary. Eg. here the max length is 5 bytes so a message 721 will be prefixed with 011 (011721) where the server would read only first 3 bytes to know the length of message.
3. Use a marker: A character like '.' can be used to separate the messages. We would send the messages alternated in between by '.' messages. On encountering a '.', the server would break the received message.

The solution has been implemented using padding. Codes attached.

Part D:

Part A:

1. Yes they are
2. Yes they are
3. 20 segments are sent
4. The segments are not of the same length. Same answer as TCP. The application data for the largest segment is:

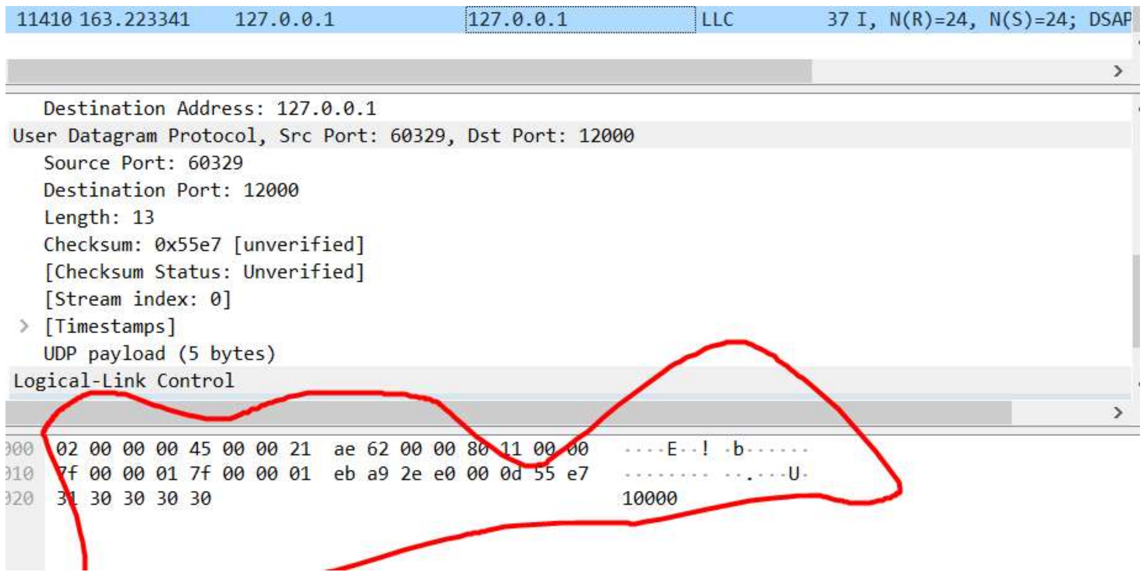


```
296 24.273460    127.0.0.1    127.0.0.1    LLC    34 [Malformed Packet]

User Datagram Protocol, Src Port: 50084, Dst Port: 12000
   Source Port: 50084
   Destination Port: 12000
   Length: 10
   Checksum: 0xdd22 [unverified]
   [Checksum Status: Unverified]
   [Stream index: 0]
 > [Timestamps]
   UDP payload (2 bytes)
Logical-Link Control
[Malformed Packet: LLC]

00  02 00 00 00 45 00 00 1e  b3 ca 00 00 80 11 00 00   ····E··· ········
10  7f 00 00 01 7f 00 00 01  c3 a4 2e e0 00 0a dd 22   ········ ··.····"
20  32 30                                              20
```

5. Coded
6. Recv is called the same number of times as the number of Wireshark TCP segments.

Part B:

7. Yes they are
8. Yes they are
9. 10000 segments are sent
10. The segments are not of the same length. Same answer as TCP. The application data for the largest segment is:



```
11410 163.223341     127.0.0.1               127.0.0.1             LLC          37 I, N(R)=24, N(S)=24; DSAP

   Destination Address: 127.0.0.1
User Datagram Protocol, Src Port: 60329, Dst Port: 12000
   Source Port: 60329
   Destination Port: 12000
   Length: 13
   Checksum: 0x55e7 [unverified]
   [Checksum Status: Unverified]
   [Stream index: 0]
 > [Timestamps]
   UDP payload (5 bytes)
Logical-Link Control

000  02 00 00 00 45 00 00 21  ae 62 00 00 80 11 00 00   ····E··! ·b······
010  7f 00 00 01 7f 00 00 01  eb a9 2e e0 00 0d 55 e7   ········ ··.···U·
020  31 30 30 30 30                                     10000
```

11. Coded
12. Recv is called the same number of times as the number of Wireshark TCP segments.

Comments:

UDP packets are received in order and the message boundaries are preserved irrespective of the send_delay value. This is because the UDP connection is initiated for each message sent.