UDP packet creation and validation

Question 1: UDP segment creation

Given the following details:

a) Source IP
b) Source port
c) Destination IP
d) Destination port
e) Data to be transmitted

output the hexadecimal representation of of the UDP segment. To compute the hexadecimal representation, simply convert each field of the segment into hexadecimal, concatenate them and output it. For instance, the hexadecimal representation of the following UDP segment

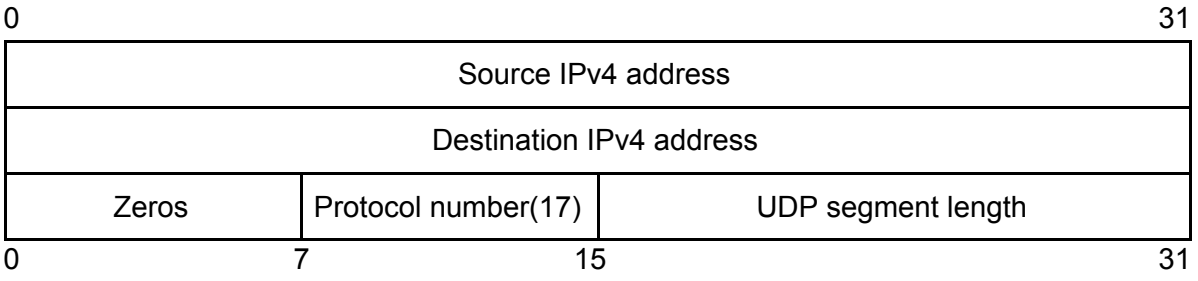
1099	20000
21	17708
ABCDEFGHIJKLM	

is 044b4e200015452c4142434445464748494a4b4c4d(Note: The checksum may not be correct. This is for illustration purpose only).

In the real world, the UDP checksum is computed in a slightly different manner than discussed in class. The steps are described below.

1. The data is first expanded(by appending 0's) so that it's length is a multiple of 16("two octets"). In the above example, 1 0 is appended to the data to make length 14 bytes(an even number of octets). Note that character with ASCII value 0 is appended and not character 0(which has ASCII value 48).

2. A pseudo IP header is computed. The structure of the pseudo IPv4 header is as follows:



3. After that, the pseudo header and the entire segment are divided into multiple groups, each 16 bits in length. Be sure to convert the dotted notation of the IP address into a 32 bit value before dividing them. There are standard functions which implement this functionality.

4. Two 16 bit groups are taken at a time and added together using 1s complement addition. Any carry is also added to the resultant 1s complement sum using 1s complement addition. The result should always be a 16 bit value.
5. Finally, you will be left with 1 value that fits in inside 16 bits. This value is the checksum of the segment.

How we will test your program

1. You are required to implement the solution in Python. We will execute the Python source code with the following arguments: source IP address, source port, destination IP address, destination port and the application layer data data(in hexadecimal form) to be transmitted. For the above example, the invocation will be:

$ python udp_create.py 192.168.56.1 1099 192.168.56.2 20000 4142434445464748494a4b4c4d

where 192.168.56.1 is the source and 192.168.56.2 is the destination.
2. Only the hexadecimal representation of the segment should be written to stdout. If you wish to print any debug messages, write them to stderr instead. The autograder may mark your program as failed if it receives unexpected output in stdout.

Sample test cases

$ python udp_create.py 192.168.56.1 4444 192.168.56.2 9999 3132333435
115c270f000d3cae3132333435
$ python udp_create.py 192.168.56.1 4444 192.168.56.2 9999 74657374
115c270f000cee3c74657374

Question 2: UDP segment parsing

Given the hexadecimal representation of a segment, source IP and destination IP, validate it(using length field and checksum). If the segment is valid, output the following in this exact order:

a) Source port
b) Destination port
c) Length
d) Checksum
e) SHA256 hash of the segment data

else output “Invalid UDP segment”.

The checksum algorithm is the same as the one used in the previous question.

How we will test your program

1. You are required to implement the solution in Python. We will execute the Python source code with the following arguments: the application layer data data(in hexadecimal form) to be transmitted, source IP address and destination IP address. For the above example, the invocation will be:

$ python udp_parse.py 044b4e200015452c41424344454647484 94a4b4c4d 192.168.56.1 192.168.56.2

2. Only print required data to stdout - 1 per line in the same order. If you wish to print any debug messages, write them to stderr instead. The autograder may mark your program as failed if it receives unexpected output in stdout.

$ python udp_parse.py 115c270f000cee3c74657374 192.168.56.1 192.168.56.2
4444
9999
12
0xee3c
9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08

$ python udp_parse.py 115c270f000d3cae3132333435 192.168.56.1 192.168.56.2
4444
9999
13
0x3cae
5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5