CS 305 Lab Tutorial
Lecture 13 RAW SOCKET

Dept. Computer Science and Engineering

Southern University of Science and Technology

SUSTech
Southern University
of Science and Technology

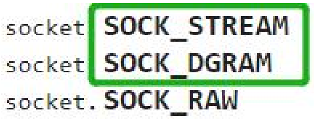
Thanks to Wei Wang

Topic

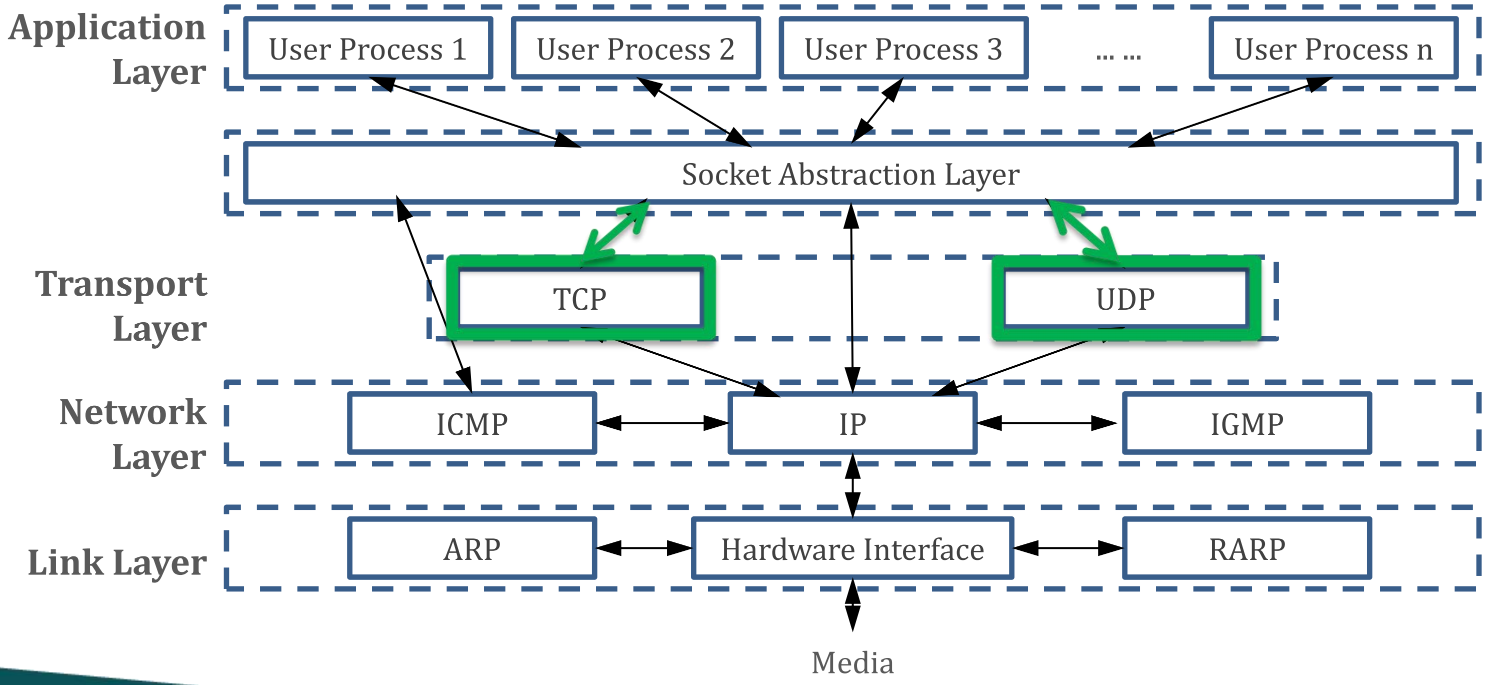
- Raw Socket
  - create socket & setsockopt
  - send & receive
- Packet
  - struct, pack, unpack
  - demo

socket in python

Socket module provides access to the BSD socket interface. It is available on all modern Unix systems, Windows, MacOS, and probably additional platforms.

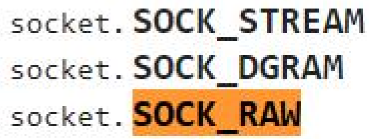


```
socket SOCK_STREAM
socket SOCK_DGRAM
socket.SOCK_RAW
```

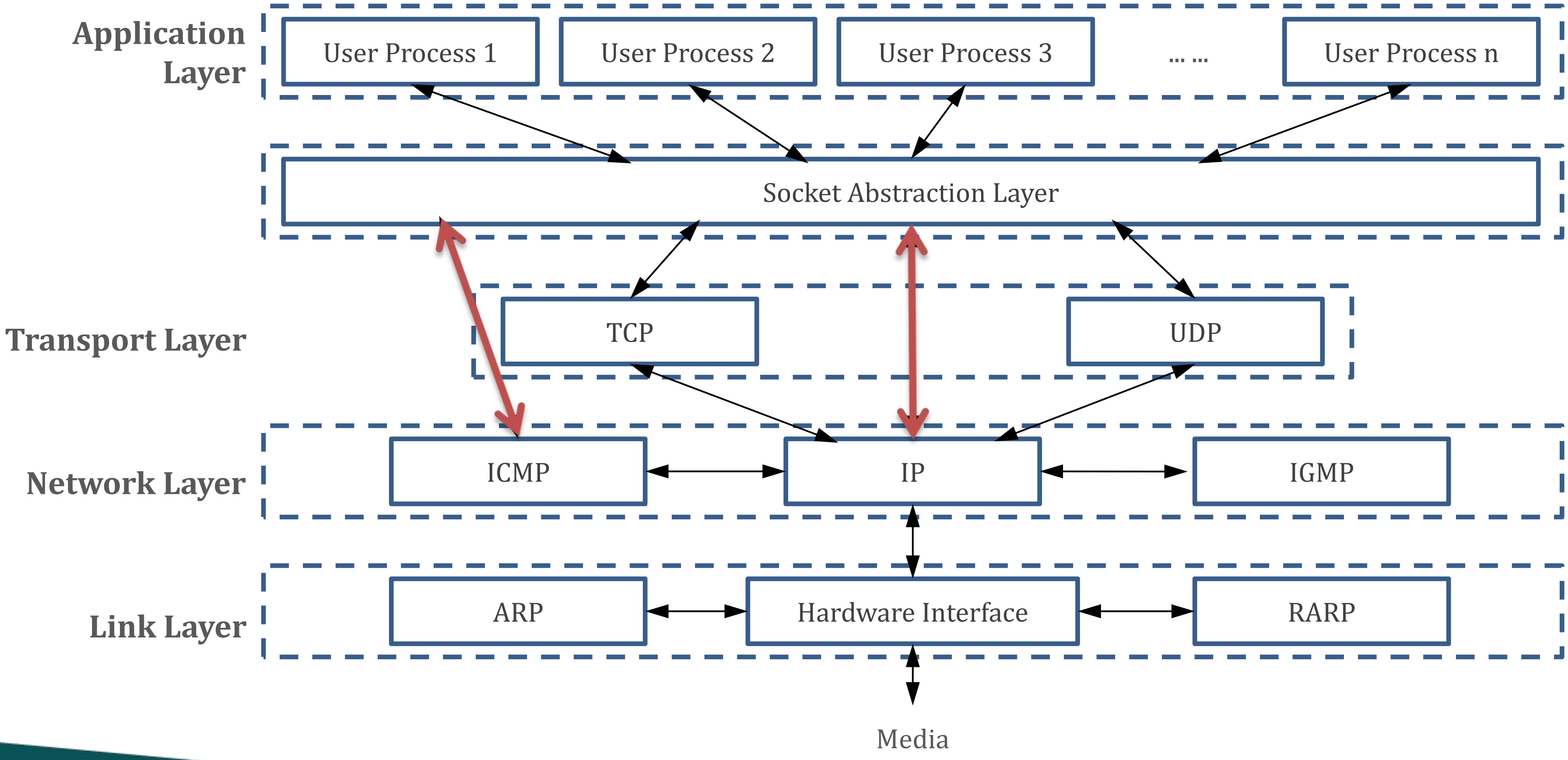


https://docs.python.org/3.9/library/socket.html

Raw Socket(1)



```
socket.SOCK_STREAM
socket.SOCK_DGRAM
socket.SOCK_RAW
```



Application Layer

User Process 1

User Process 2

User Process 3

… …

User Process n

Socket Abstraction Layer

Transport Layer

TCP

UDP

Network Layer

ICMP

IP

IGMP

Link Layer

ARP

Hardware Interface

RARP

Media

Create socket

socket.socket(family=AF_INET, type=SOCK_STREAM, proto=0, fileno=None)

- Create a new socket using the given address family, socket type and protocol number.
- The address family should be **AF_INET** (the default), AF_INET6, AF_UNIX, AF_CAN, AF_PACKET, or AF_RDS. The socket type should be SOCK_STREAM (the default), SOCK_DGRAM, **SOCK_RAW** or perhaps one of the other SOCK_ constants. The protocol number is usually zero and may be omitted or in the case where the address family is AF_CAN the protocol should be one of CAN_RAW, CAN_BCM or CAN_ISOTP.

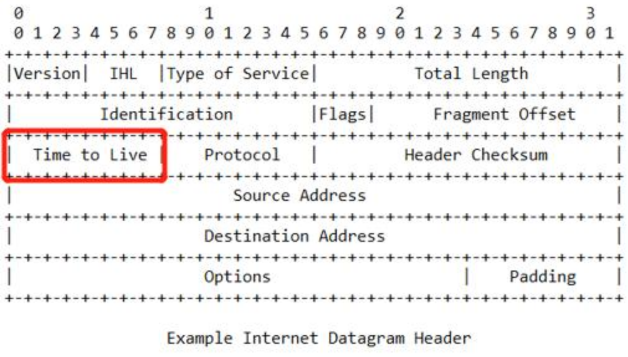
- demo1:
  - socket(AF_INET, SOCK_STREAM)
  - socket(AF_INET, SOCK_DGRAM)

- demo2:
  - socket(family=AF_INET,type=**SOCK_RAW**,proto=IPPROTO_ICMP)
  - socket(PF_PACKET, SOCK_RAW, htons(0x0800))
  - socket(family=AF_INET,type=**SOCK_RAW**,proto=IPPROTO_UDP)

https://sock-raw.org/papers/sock_raw

setsockopt

- socket.**setsockopt**(**level, optname, value: int**)
- socket.setsockopt(level, optname, value: buffer)
- socket.setsockopt(level, optname, None, optlen: int)
  - Set the value of the given socket option (see the Unix manual page setsockopt(2)). The needed symbolic constants are defined in the socket module (SO_* etc.). The value can be an integer, None or a bytes-like object representing a buffer. In the later case it is up to the caller to ensure that the bytestring contains the proper bits (see the optional built-in module struct for a way to encode C structures as bytestrings). When value is set to None, optlen argument is required. It’ s equivalent to call setsockopt() C function with optval=NULL and optlen=optlen.
  - demo: self._sock.**setsockopt**(socket.IPPROTO_IP,socket.IP_TTL,ttl)
    - means set the value of TTL filed of IP Header as ‘ttl’



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Example Internet Datagram Header

send, sendto

- socket.**send**(bytes[, flags])
  - Send data to the socket. The socket must be connected to a remote socket. The optional flags argument has the same meaning as for recv() above. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data.
    - demo: sock.**send**(request)

- socket.**sendto**(bytes, address)
  - Send data to the socket. The socket should not be connected to a remote socket, since the destination socket is specified by address. Return the number of bytes sent.
    - demo: **sendto**(udp_pkt,(dName,dPort))

receive, receivefrom

- socket.**recv**(bufsize[, flags])
  - Receive data from the socket. The return value is a bytes object representing the data received. The maximum amount of data to be received at once is specified by bufsize. See the Unix manual page recv(2) for the meaning of the optional argument flags; it defaults to zero.
    - demo: reply = sock.recv(1024)

- socket.**recvfrom**(bufsize[, flags])
  - Receive data from the socket. The return value is a pair (bytes, address) where bytes is a bytes object representing the data received and address is the address of the socket sending the data. See the Unix manual page recv(2) for the meaning of the optional argument flags; it defaults to zero.
    - demo: message, cAddress = sSocket.recvfrom(2048)

package/unpack packet - struct

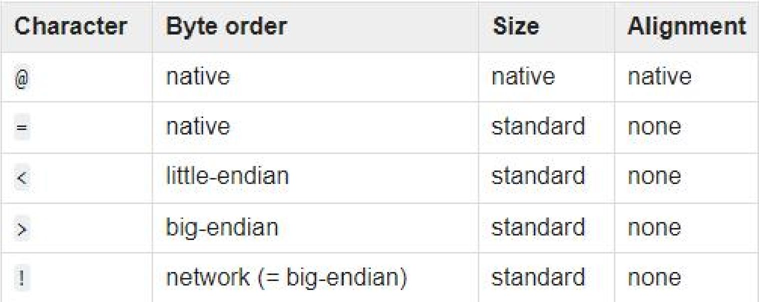
Module **struct** performs conversions between Python values and C structs represented as Python **bytes** objects. This can be used in handling binary data stored in files or from network connections, among other sources. It uses **Format Strings** as compact descriptions of the layout of the C structs and the intended conversion to/from Python values.

- **struct.pack(format, v1, v2, ...)**
  - Return a bytes object containing the values v1, v2, … packed according to the format string format. The arguments must match the values required by the format exactly.
- **struct.unpack(format, buffer)**
  - Unpack from the buffer buffer (presumably packed by pack(format, ...)) according to the format string format. The result is a tuple even if it contains exactly one item. The buffer's size in bytes must match the size required by the format, as reflected by calcsize().

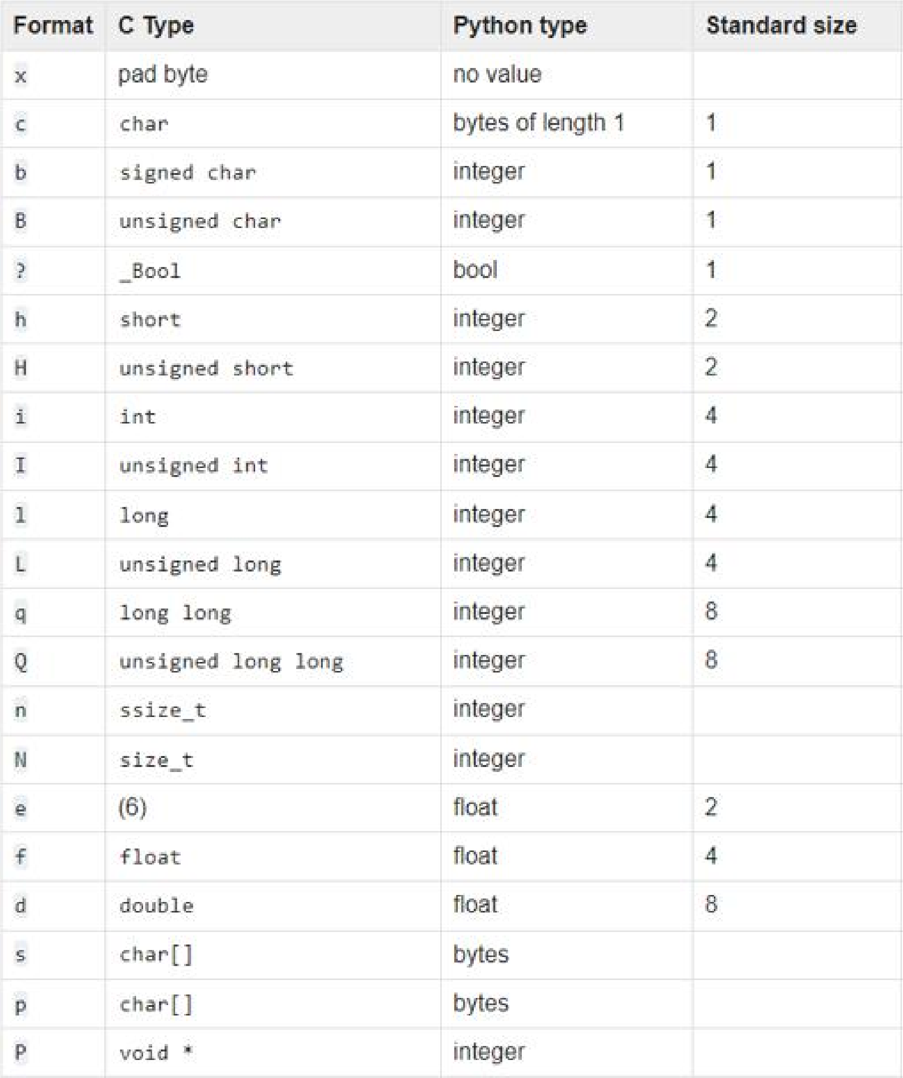
https://docs.python.org/3.9/library/struct.html

struct-format(1)

‘Format’ strings are the mechanism used to **specify the expected layout when packing and unpacking data**. They are built up from Format Characters, which specify the type of data being packed/unpacked. In addition, there are special characters for controlling the **Byte Order**, **Size**, and **Alignment**.



Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none



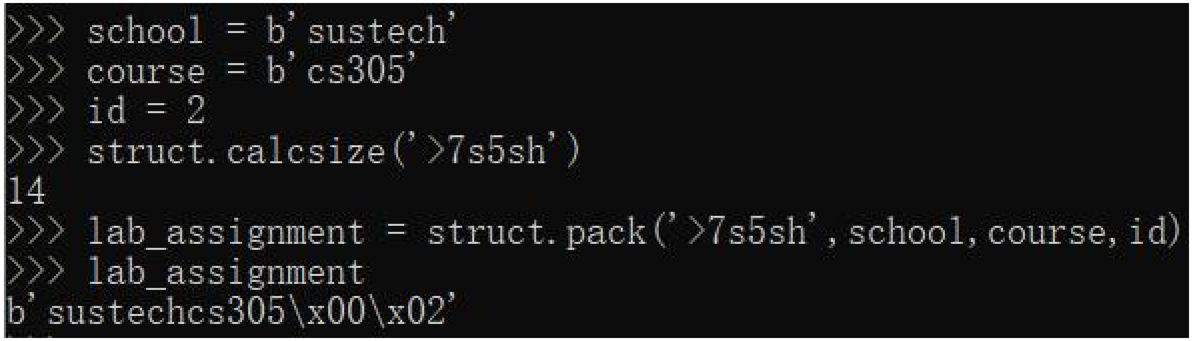
Format	C Type	Python type	Standard size
x	pad byte	no value	
c	char	bytes of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
l	long	integer	4
L	unsigned long	integer	4
q	long long	integer	8
Q	unsigned long long	integer	8
n	ssize_t	integer	
N	size_t	integer	
e	(6)	float	2
f	float	float	4
d	double	float	8
s	char[]	bytes	
p	char[]	bytes	
P	void *	integer	

SUSTech
Southern University
of Science and Technology

https://docs.python.org/3.9/library/struct.html

Struct-format(2)

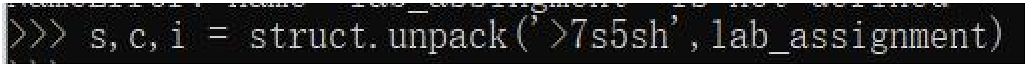
Pack the ‘school’, ‘course’, and ‘id’ to bytes, ‘school’ is a byte[] with 7 items, ‘course’ is a byte[] with 5 items, ‘id’ is expected to treat as a short number whose width is 2 bytes.



```
>>> school = b'sustech'
>>> course = b'cs305'
>>> id = 2
>>> struct.calcsize('>7s5sh')
14
>>> lab_assignment = struct.pack('>7s5sh',school,course,id)
>>> lab_assignment
b'sustechcs305\x00\x02'
```

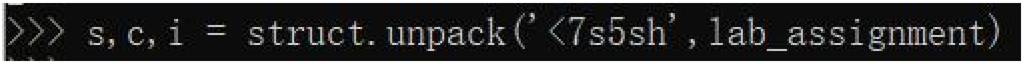
Unpack the ‘lab_assignment’ to get the information about school, course and id, which of the following way is(are) correct?

A.



```
>>> s,c,i = struct.unpack('>7s5sh',lab_assignment)
```

B.



```
>>> s,c,i = struct.unpack('<7s5sh',lab_assignment)
```

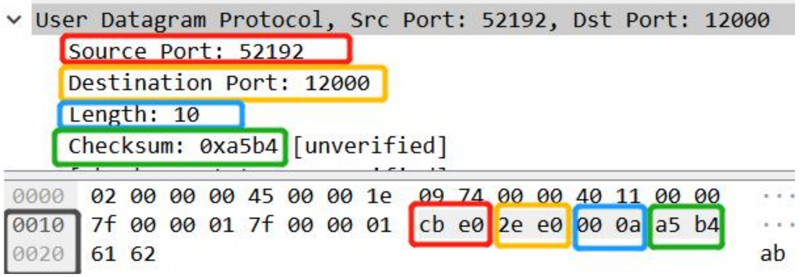
struct-demo1

Generate a UDP packet as the packet captured on the right hand.

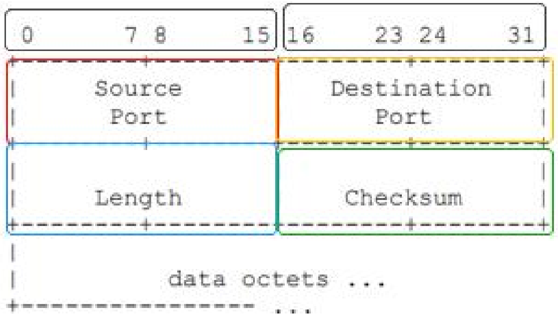
- step1: generate a UDP header

  Which one(s) is(are) correct:

  - A:
    - struct.pack('<4H',52192,12000,10,0xa5b4)
  - B:
    - struct.pack('>4H',52192,12000,10,0xa5b4)
  - C:
    - struct.pack('!4H',52192,12000,10,0xa5b4)
  - D:
    - struct.pack('>HHHH',52192,12000,10,0xa5b4)



```
User Datagram Protocol, Src Port: 52192, Dst Port: 12000
    Source Port: 52192
    Destination Port: 12000
    Length: 10
    Checksum: 0xa5b4 [unverified]

0000  02 00 00 00 45 00 00 1e  09 74 00 00 40 11 00 00   ...
0010  7f 00 00 01 7f 00 00 01  cb e0 2e e0 00 0a a5 b4   ...
0020  61 62                                              ab
```



```
 0      7 8     15 16    23 24     31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

Q1: Could using 'h' to replace 'H' in the format description here? Why?

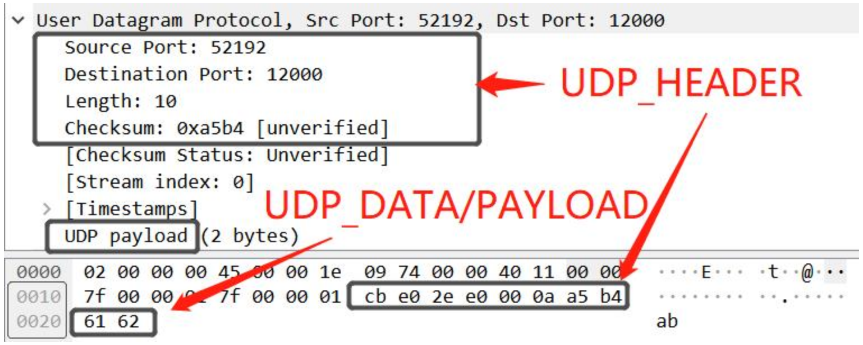
Q2: while receive UDP packet 'udp_pkt', how to get the source port and the checksum fileds from the 'udp_pkt'?

struct-demo2

Generate a UDP packet as the packet captured on the right hand.

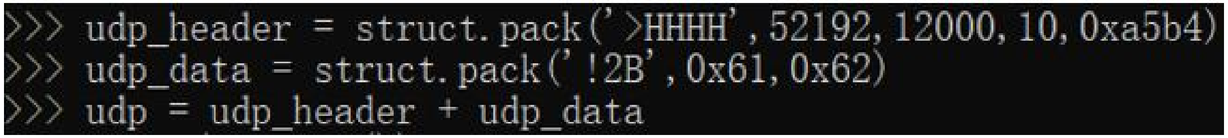
- step2: generate a UDP packet

  A Udp packet consists of UDP header and udp payload.

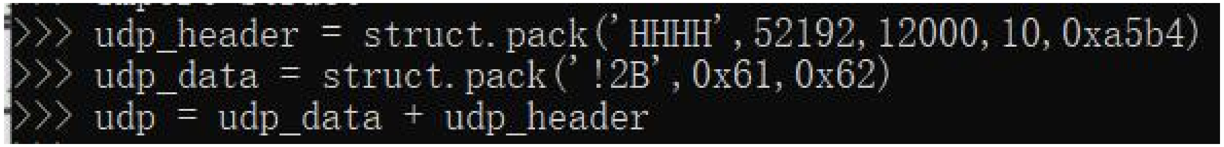


```
User Datagram Protocol, Src Port: 52192, Dst Port: 12000
    Source Port: 52192
    Destination Port: 12000
    Length: 10
    Checksum: 0xa5b4 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
    UDP payload (2 bytes)
0000  02 00 00 00 45 00 00 1e  09 74 00 00 40 11 00 00   ····E··· ·t··@··
0010  7f 00 00 01 7f 00 00 01  cb e0 2e e0 00 0a a5 b4   ········ ··.·····
0020  61 62                                              ab
```

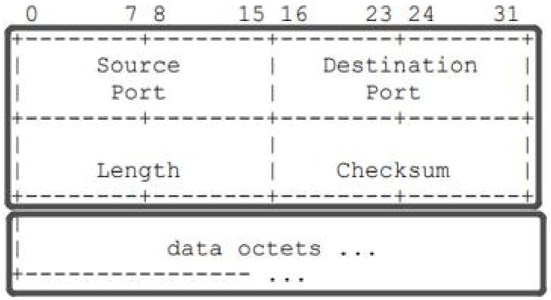
For bytes in python, Does “a+b” equal to “b+a” ?



```
>>> udp_header = struct.pack('>HHHH',52192,12000,10,0xa5b4)
>>> udp_data = struct.pack('!2B',0x61,0x62)
>>> udp = udp_header + udp_data
```



```
>>> udp_header = struct.pack('HHHH',52192,12000,10,0xa5b4)
>>> udp_data = struct.pack('!2B',0x61,0x62)
>>> udp = udp_data + udp_header
```

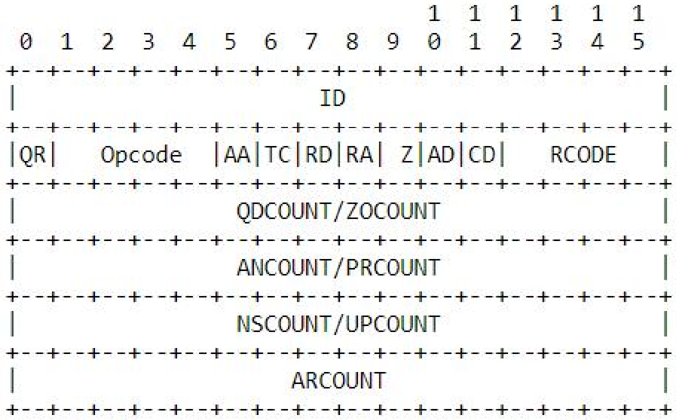


```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

User Datagram Header Format

struct-demo3

```
class DNSHeader:
  Struct = struct.Struct('!6H’)  #

  def __init__(self):
    self.__dict__ = {
      field: None
      for field in ('ID', 'QR', 'OpCode’, 'AA', 'TC', 'RD', 'RA', 'Z’,
      'RCode', 'QDCount', 'ANCount', 'NSCount', 'ARCount’)}

  def parse_header(self, data):
    self.ID, misc, self.QDCount, self.ANcount,  self.NScount, self.ARcount = DNSHeader.Struct.unpack_from(data)
    self.QR = (misc & 0x8000) != 0
    self.OpCode = (misc & 0x7800) >> 11
    self.AA = (misc & 0x0400) != 0
    self.TC = (misc & 0x200) != 0
    self.RD = (misc & 0x100) != 0
    self.RA = (misc & 0x80) != 0
    self.Z = (misc & 0x70) >> 4 # Never used
    self.RCode = misc & 0xF

  def __str__(self):
    return '<DNSHeader {}>'.format(str(self.__dict__))
```

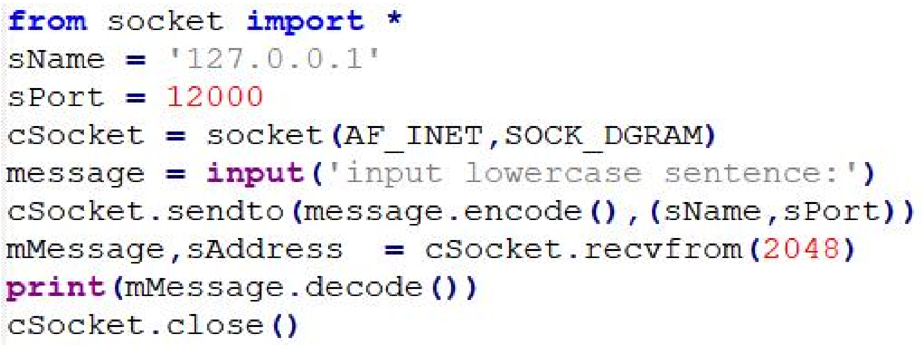


```
                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      ID                       |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |QR|   Opcode  |AA|TC|RD|RA| Z|AD|CD|   RCODE   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                QDCOUNT/ZOCOUNT                |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                ANCOUNT/PRCOUNT                |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                NSCOUNT/UPCOUNT                |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ARCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

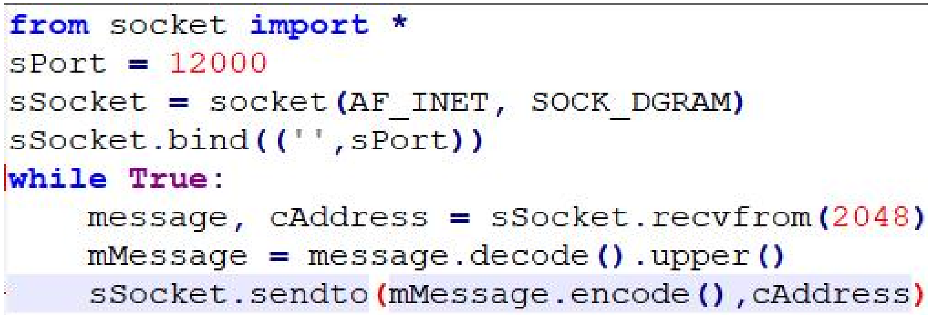
SUSTech
Southern University
of Science and Technology

DGRAM SOCKET(1)

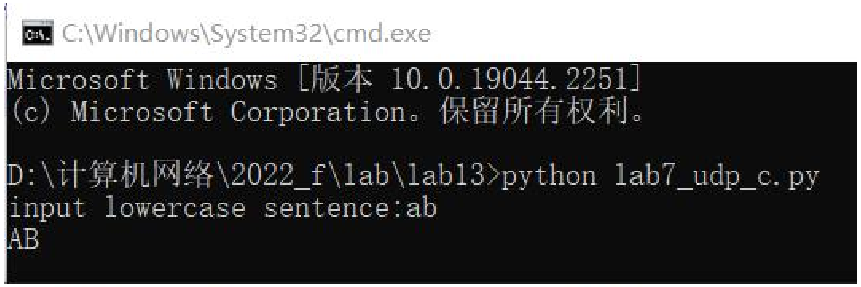
- Implement a echo server and client based on UDP by DGRAM socket.



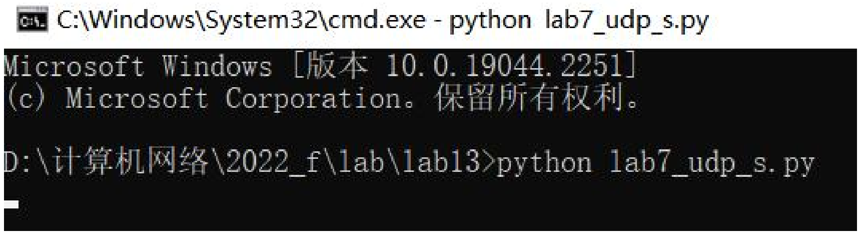
```python
from socket import *
sName = '127.0.0.1'
sPort = 12000
cSocket = socket(AF_INET,SOCK_DGRAM)
message = input('input lowercase sentence:')
cSocket.sendto(message.encode(),(sName,sPort))
mMessage,sAddress  = cSocket.recvfrom(2048)
print(mMessage.decode())
cSocket.close()
```



```python
from socket import *
sPort = 12000
sSocket = socket(AF_INET, SOCK_DGRAM)
sSocket.bind(('',sPort))
while True:
    message, cAddress = sSocket.recvfrom(2048)
    mMessage = message.decode().upper()
    sSocket.sendto(mMessage.encode(),cAddress)
```

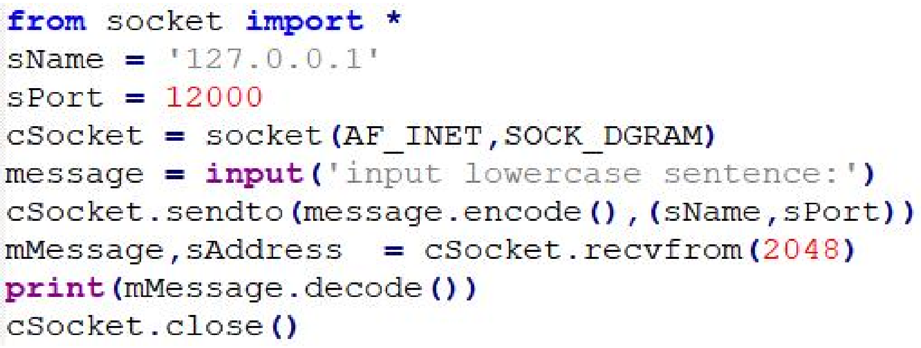


```
Microsoft Windows [版本 10.0.19044.2251]
(c) Microsoft Corporation。保留所有权利。

D:\计算机网络\2022_f\lab\lab13>python lab7_udp_c.py
input lowercase sentence:ab
AB
```

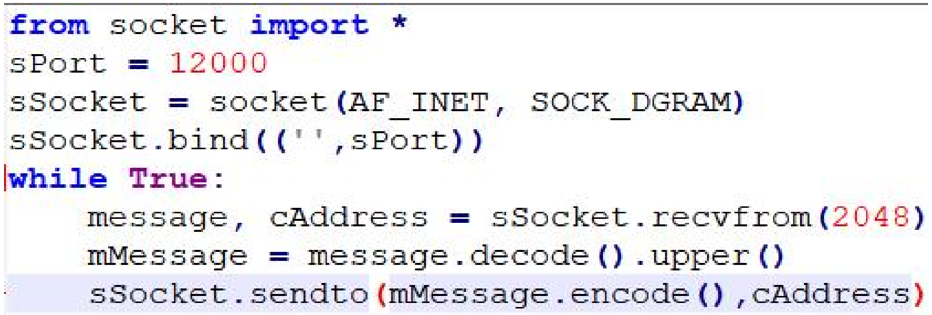


```
Microsoft Windows [版本 10.0.19044.2251]
(c) Microsoft Corporation。保留所有权利。

D:\计算机网络\2022_f\lab\lab13>python lab7_udp_s.py
_
```

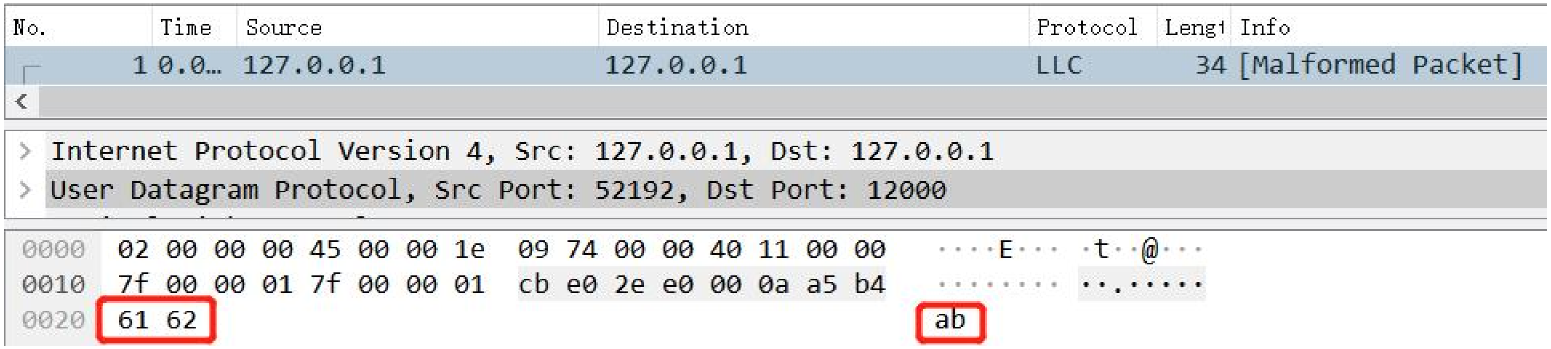
DGRAM SOCKET(2)



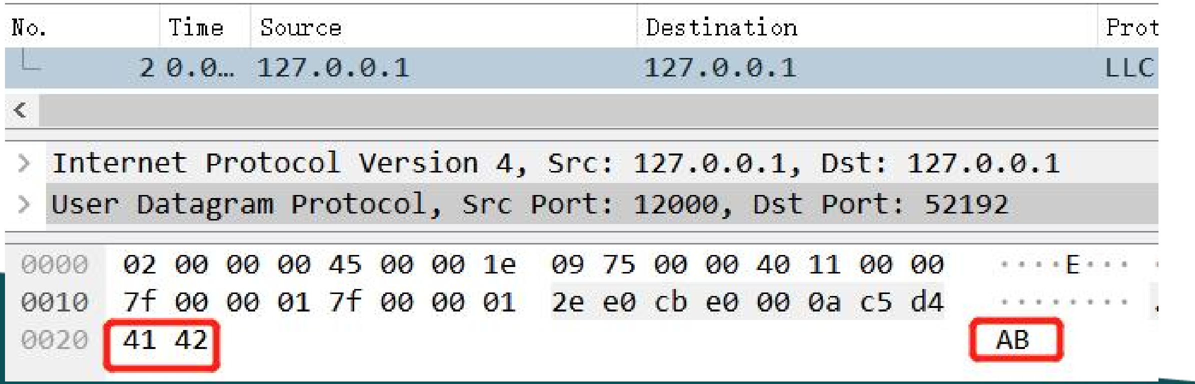
```
from socket import *
sName = '127.0.0.1'
sPort = 12000
cSocket = socket(AF_INET,SOCK_DGRAM)
message = input('input lowercase sentence:')
cSocket.sendto(message.encode(),(sName,sPort))
mMessage,sAddress  = cSocket.recvfrom(2048)
print(mMessage.decode())
cSocket.close()
```



```
from socket import *
sPort = 12000
sSocket = socket(AF_INET, SOCK_DGRAM)
sSocket.bind(('',sPort))
while True:
    message, cAddress = sSocket.recvfrom(2048)
    mMessage = message.decode().upper()
    sSocket.sendto(mMessage.encode(),cAddress)
```



```
No.    Time   Source       Destination   Protocol  Lengt Info
     1 0.0…   127.0.0.1    127.0.0.1     LLC          34 [Malformed Packet]

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 52192, Dst Port: 12000

0000  02 00 00 00 45 00 00 1e  09 74 00 00 40 11 00 00   ····E··· ·t··@···
0010  7f 00 00 01 7f 00 00 01  cb e0 2e e0 00 0a a5 b4   ········ ··.·····
0020  61 62                                              ab
```



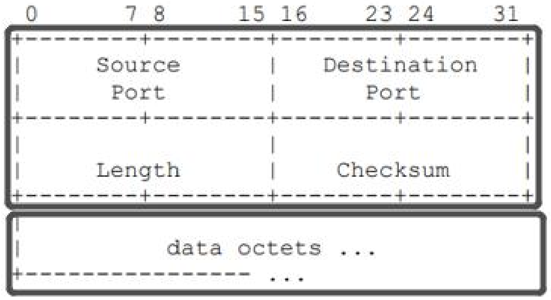
```
No.    Time   Source       Destination   Prot
     2 0.0…   127.0.0.1    127.0.0.1     LLC

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 12000, Dst Port: 52192

0000  02 00 00 00 45 00 00 1e  09 75 00 00 40 11 00 00   ····E···
0010  7f 00 00 01 7f 00 00 01  2e e0 cb e0 00 0a c5 d4   ········
0020  41 42                                              AB
```

SUSTech
Southern University
of Science and Technology

RAW SOCKET(1)

Implement a **sender** and receiver based on UDP by raw socket.

sender:



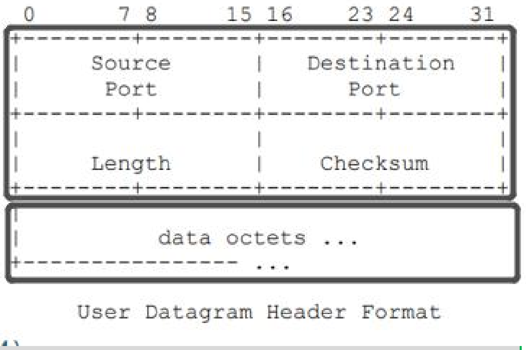
```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

User Datagram Header Format

```
from socket import *
from struct import *
dName = '127.0.0.1'
sPort = 12007
dPort = 12008
cSocket = socket(AF_INET, SOCK_RAW, IPPROTO_UDP)
cSocket.bind(('127.0.0.1',sPort))
message = input('input tx data:')
checksum = 0x00;
length = 8+len(message);

udp_head = pack('!4H',sPort,dPort,length,checksum)
udp_pkt = udp_head + message.encode()
print("tx pkt:".format(udp_pkt.hex()))

cSocket.sendto(udp_pkt,(dName,dPort))

cSocket.close()
```

RAW SOCKET(2)

Implement a sender and **receiver** based on UDP by **raw** socket.



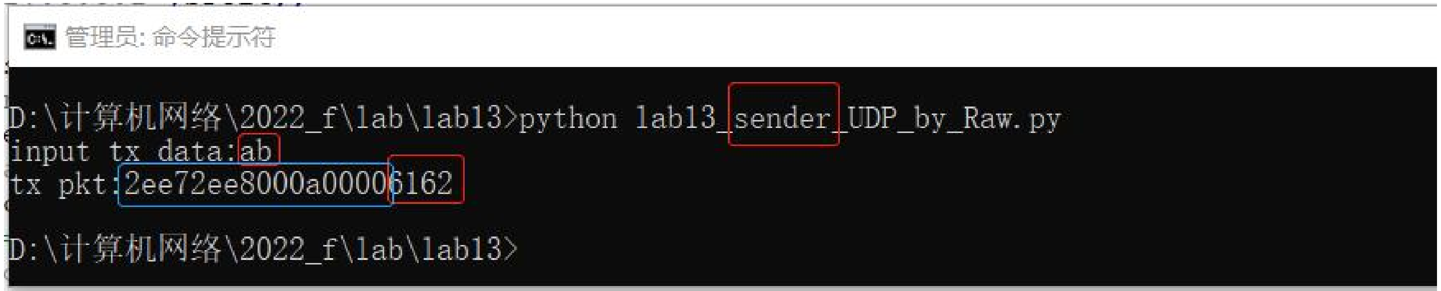
```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...

     User Datagram Header Format
```

receiver:

```
from socket import *
from struct import *
sPort = 12008
sSocket = socket(AF_INET, SOCK_RAW, IPPROTO_UDP)
sSocket.bind(('127.0.0.1',sPort))

message, cAddress = sSocket.recvfrom(2048)
print("rx pkt_len: {},message:{}".format( len(message), message.hex() ) )

rHeader = message[20:28]
print("rx udp-header:%s"%rHeader.hex())

rsrc,rdst,rlen,rchecks = unpack('!4H',rHeader)
print("rsrc:{},rdst:{},rlen:{},rchecks:{},rdata:{}".format(rsrc,rdst,rlen,rchecks,message[28:]))

sSocket.close()
```

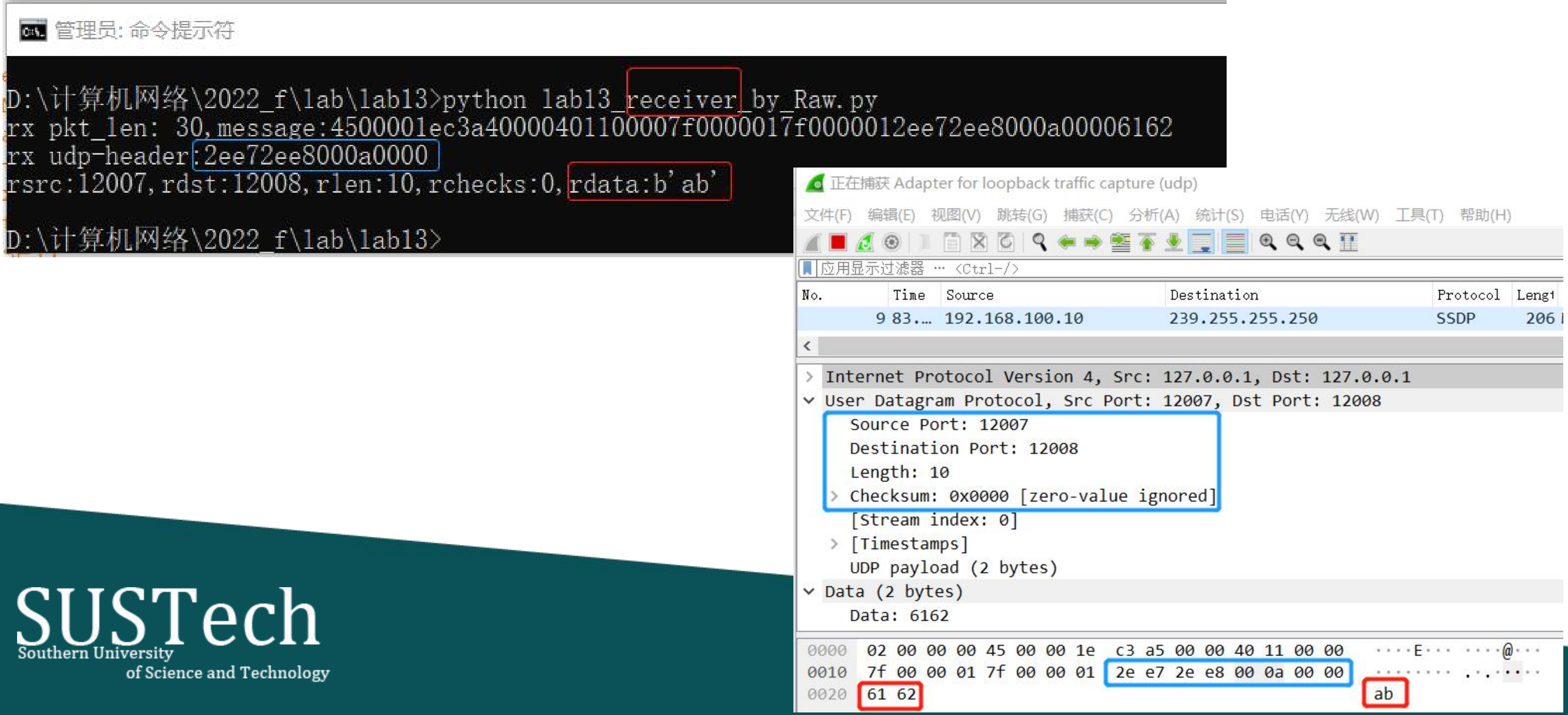
RAW SOCKET(3)

Test as an administrator in the command line tool of Windows



管理员: 命令提示符

```
D:\计算机网络\2022_f\lab\lab13>python lab13_sender_UDP_by_Raw.py
input tx data:ab
tx pkt:2ee72ee8000a00006162

D:\计算机网络\2022_f\lab\lab13>
```



管理员: 命令提示符

```
D:\计算机网络\2022_f\lab\lab13>python lab13_receiver_by_Raw.py
rx pkt_len: 30,message:4500001ec3a40000401100007f0000017f0000012ee72ee8000a00006162
rx udp-header:2ee72ee8000a0000
rsrc:12007,rdst:12008,rlen:10,rchecks:0,rdata:b'ab'

D:\计算机网络\2022_f\lab\lab13>
```

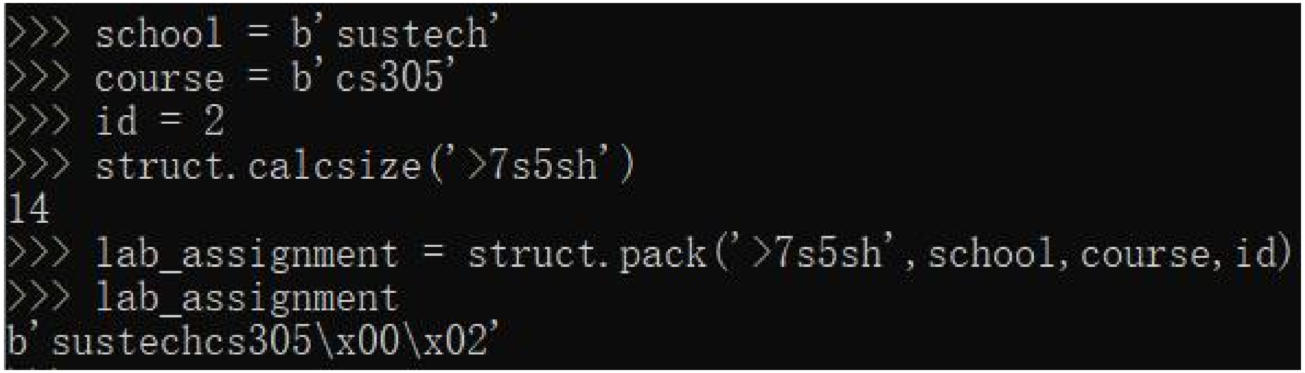
正在捕获 Adapter for loopback traffic capture (udp)
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>

```
No.   Time   Source            Destination        Protocol  Lengt
    9 83.…   192.168.100.10    239.255.255.250    SSDP      206
```

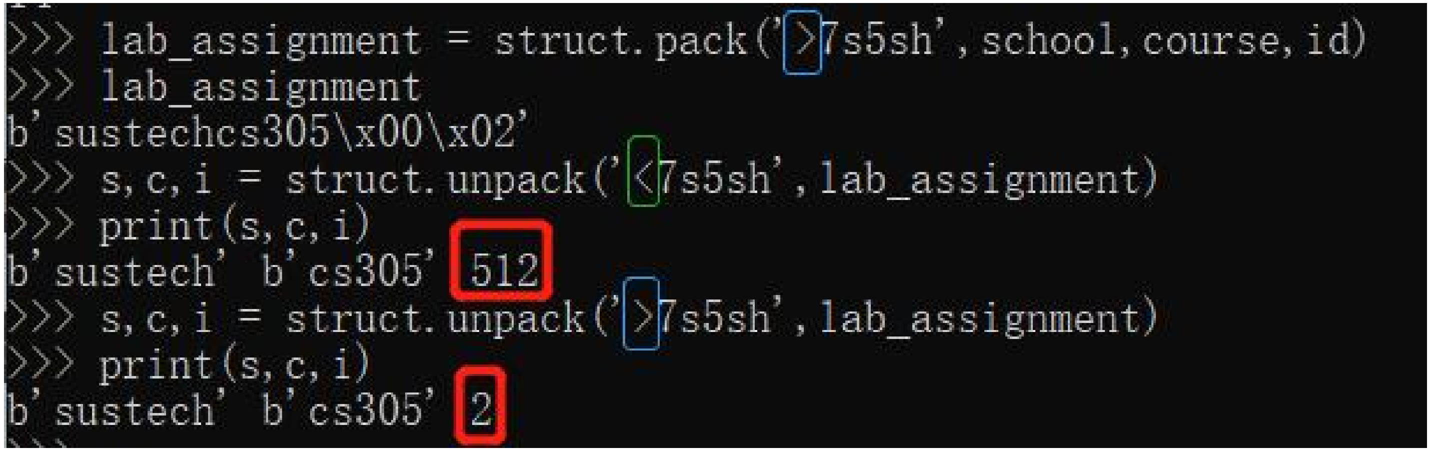
```
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 12007, Dst Port: 12008
    Source Port: 12007
    Destination Port: 12008
    Length: 10
  > Checksum: 0x0000 [zero-value ignored]
    [Stream index: 0]
  > [Timestamps]
    UDP payload (2 bytes)
v Data (2 bytes)
    Data: 6162
```

```
0000  02 00 00 00 45 00 00 1e  c3 a5 00 00 40 11 00 00   ····E··· ····@···
0010  7f 00 00 01 7f 00 00 01  2e e7 2e e8 00 0a 00 00   ········ .·.·····
0020  61 62                                              ab
```

TIPs(1)

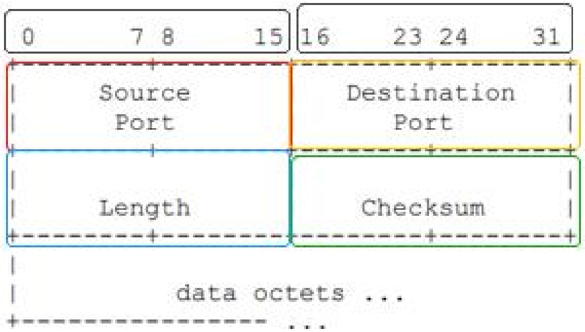


```
>>> school = b'sustech'
>>> course = b'cs305'
>>> id = 2
>>> struct.calcsize('>7s5sh')
14
>>> lab_assignment = struct.pack('>7s5sh', school, course, id)
>>> lab_assignment
b'sustechcs305\x00\x02'
```

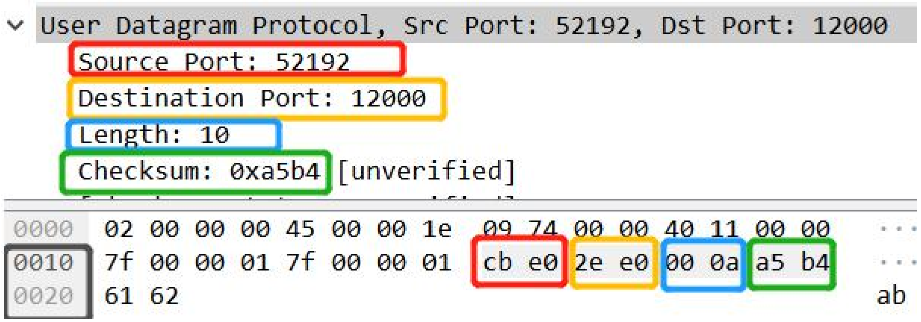


```
>>> lab_assignment = struct.pack('>7s5sh', school, course, id)
>>> lab_assignment
b'sustechcs305\x00\x02'
>>> s, c, i = struct.unpack('<7s5sh', lab_assignment)
>>> print(s, c, i)
b'sustech' b'cs305' 512
>>> s, c, i = struct.unpack('>7s5sh', lab_assignment)
>>> print(s, c, i)
b'sustech' b'cs305' 2
```

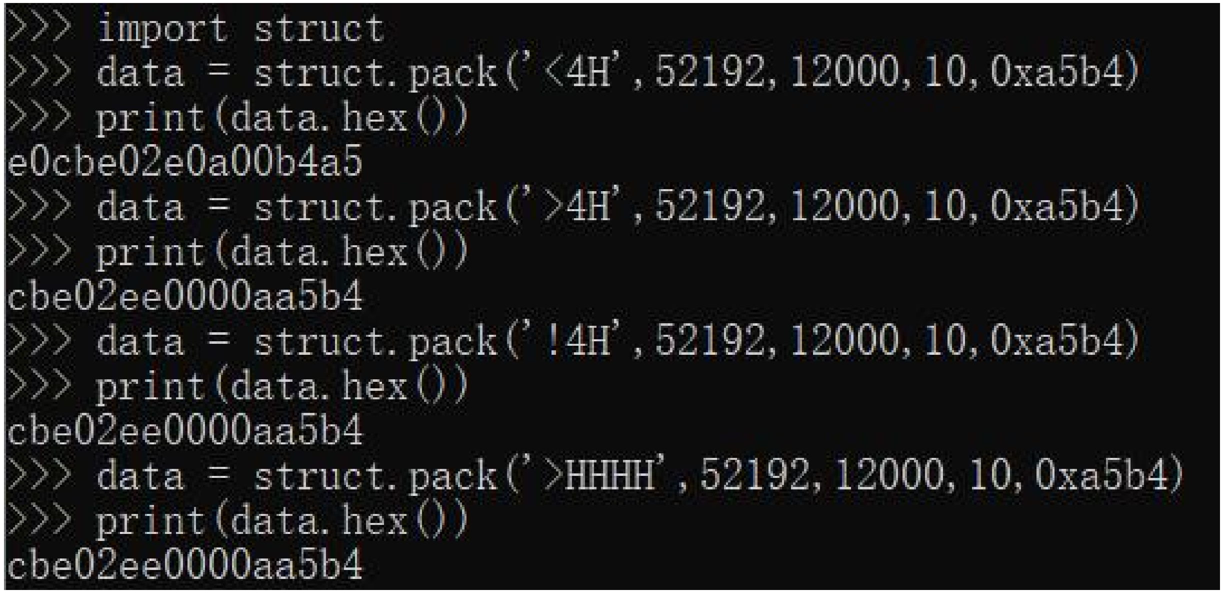
TIPs(2)



```
 0      7 8     15 16    23 24     31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

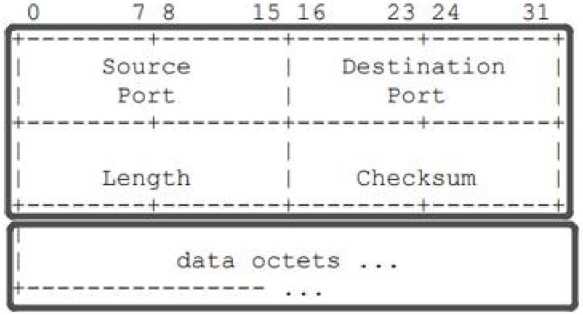


```
v User Datagram Protocol, Src Port: 52192, Dst Port: 12000
    Source Port: 52192
    Destination Port: 12000
    Length: 10
    Checksum: 0xa5b4 [unverified]

0000  02 00 00 00 45 00 00 1e  09 74 00 00 40 11 00 00   ...
0010  7f 00 00 01 7f 00 00 01  cb e0 2e e0 00 0a a5 b4   ...
0020  61 62                                              ab
```



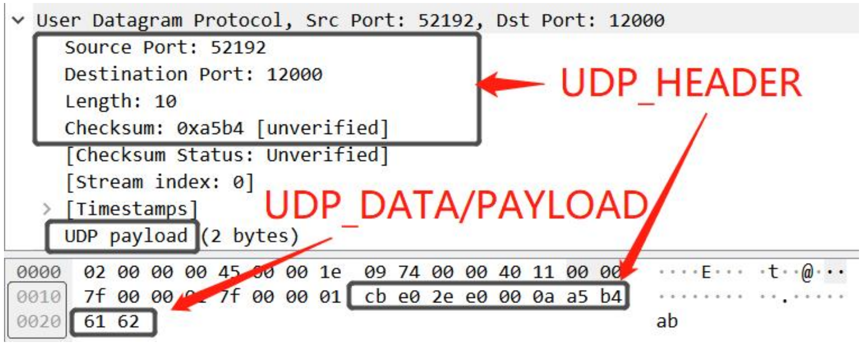
```
>>> import struct
>>> data = struct.pack(’<4H’,52192,12000,10,0xa5b4)
>>> print(data.hex())
e0cbe02e0a00b4a5
>>> data = struct.pack(’>4H’,52192,12000,10,0xa5b4)
>>> print(data.hex())
cbe02ee0000aa5b4
>>> data = struct.pack(’!4H’,52192,12000,10,0xa5b4)
>>> print(data.hex())
cbe02ee0000aa5b4
>>> data = struct.pack(’>HHHH’,52192,12000,10,0xa5b4)
>>> print(data.hex())
cbe02ee0000aa5b4
```

TIPs(3)

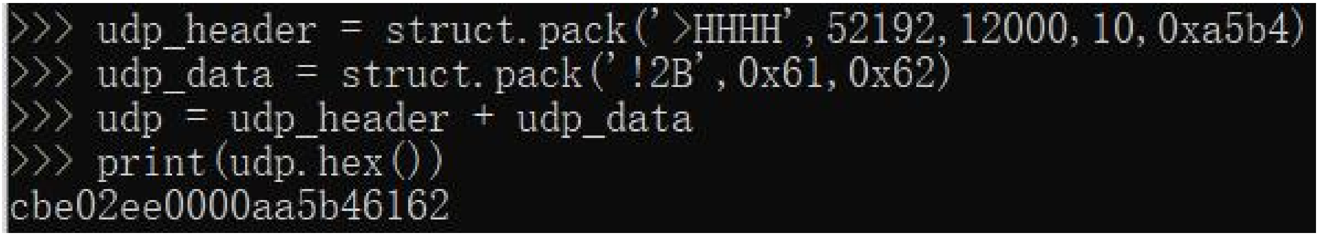


```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

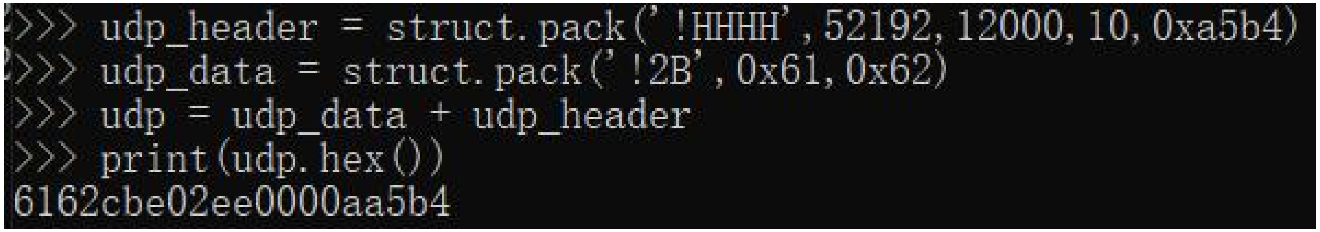
User Datagram Header Format



```
∨ User Datagram Protocol, Src Port: 52192, Dst Port: 12000
    Source Port: 52192
    Destination Port: 12000
    Length: 10
    Checksum: 0xa5b4 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
    UDP payload (2 bytes)

0000  02 00 00 00 45 00 00 1e  09 74 00 00 40 11 00 00   ····E··· ·t··@···
0010  7f 00 00 01 7f 00 00 01  cb e0 2e e0 00 0a a5 b4   ········ ··.·····
0020  61 62                                              ab
```



```
>>> udp_header = struct.pack('>HHHH',52192,12000,10,0xa5b4)
>>> udp_data = struct.pack('!2B',0x61,0x62)
>>> udp = udp_header + udp_data
>>> print(udp.hex())
cbe02ee0000aa5b46162
```



```
>>> udp_header = struct.pack('!HHHH',52192,12000,10,0xa5b4)
>>> udp_data = struct.pack('!2B',0x61,0x62)
>>> udp = udp_data + udp_header
>>> print(udp.hex())
6162cbe02ee0000aa5b4
```