2-1

client.py

```
from socket import *
import sys

addr = sys.argv[1]
port = sys.argv[2]
file = sys.argv[3]

ClientSocket = socket(AF_INET, SOCK_STREAM)
ClientSocket.connect((addr, int(port)))
head = 'GET /%s HTTP/1.1\r\n' % file
ClientSocket.send(head.encode())
ClientSocket.send(file.encode())
data = ClientSocket.recv(1024)
raw_data = ClientSocket.recv(1024)
data1 = ''
print(data.decode())
while raw_data :
    data1 = data1 + raw_data.decode()
    raw_data = ClientSocket.recv(1024)
data1 = data1 + raw_data.decode()
print(data1)
ClientSocket.close()
```

new_web.py

```
from socket import *
import threading
import sys

def web_server(connectionSocket,addr):
    try:
        print('connected from',addr)
        # Receives the request message from the client
        message =  connectionSocket.recv(1024)
        # Extract the path of the requested object from the message
        # The path is the second part of HTTP header, identified by [1]
        filename = message.split()[1]

        # Because the extracted path of the HTTP request includes
        # a character '/', we read the path from the second character
        f = open(filename[1:])

        # Store the entire contenet of the requested file in a temporary buffer
        outputdata = f.read()

        # Send the HTTP response header line to the connection socket
        header = ' HTTP/1.1 200 OK\n\n'
        connectionSocket.send(header.encode())
```

```
        # Send the content of the requested file to the connection socket
        for i in range(0, len(outputdata)):
            connectionSocket.send(outputdata[i].encode())

        # Close the client connection socket
        connectionSocket.close()

    except IOError:
        # Send HTTP response message for file not found
        header = 'HTTP/1.1 404 Not Found\n\n'
        connectionSocket.send(header.encode())

        # Close the client connection socket
        connectionSocket.close()



serverSocket = socket(AF_INET, SOCK_STREAM)

# Prepare a sever socket
host = 'localhost'
port = 12315
serverSocket.bind((host,port))
serverSocket.listen(10)

while True:
    # Establish the connection
    print (' The server is ready to receive')

    # Set up a new connection from the client
    connectionSocket, addr = serverSocket.accept()

    thread = threading.Thread(target=web_server,args=(connectionSocket,addr))
    thread.start()


serverSocket.close()

# Terminate the program after sending the corresponding data
sys.exit()
```

2-2

```
import time
from socket import *
import sys

server_address = sys.argv[1]
server_port = sys.argv[2]

clientSocket = socket(AF_INET,SOCK_DGRAM)
clientSocket.settimeout(1)
RTTs = []
RTT_count = 0
```

```
lost_num = 0
for i in range(1,11):
    send_time = time.time()
    send = ('Ping %d %s' % (i,send_time)).encode()
    try:
        clientSocket.sendto(send,(server_address,int(server_port)))
        recv, recv_address = clientSocket.recvfrom(1024)
        RTT = time.time() - send_time
        RTTs.append(RTT)
        RTT_count = RTT_count + RTT
        print('Reply from %s: %s RTT = %fms' % (recv_address[0], recv, round(RTT
* 1000)))
    except:
        print('Request timed out')
        lost_num = lost_num + 1

clientSocket.close()
print('丢包率:%d%% 最小RTT:%dms 最大RTT:%dms 平均RTT:%dms' % (lost_num * 10,
round(1000*min(RTTs)), round(1000*max(RTTs)), round(1000*RTT_count/(10-
lost_num))))
```

2-3

```
from socket import *
import base64
from email.mime.multipart import MIMEMultipart
from email.header import Header
from email.mime.text import MIMEText

# Mail content
subject = "I love computer networks!"
contenttype = "text/plain"
msg = "I love computer networks!"
endmsg = "\r\n.\r\n"


# Choose a mail server (e.g. Google mail server) and call it mailserver
mailserver = 'smtp.163.com'

# Sender and reciever
fromaddress = 'timelovercc@163.com'
toaddress = 'timelovercc@163.com'

# Auth information (Encode with base64)
username = 'dGltZWxvdmVyY2NAMTYzLmNvbQ=='
password =  ''


message = MIMEMultipart()
message['From'] = fromaddress
message['To'] = toaddress
message['Subject'] = Header(subject, 'utf-8')
message.attach(MIMEText("正文",'plain','utf-8'))
```

```
att1 = MIMEText(open('test.txt', 'rb').read(), 'base64', 'utf-8')
att1["Content-Type"] = 'application/octet-stream'
att1["Content-Disposition"] = 'attachment; filename="test.txt"'
message.attach(att1)

att2 = MIMEText(open('pic.JPG', 'rb').read(), 'base64', 'utf-8')
att2["Content-Type"] = 'application/octet-stream'
att2["Content-Disposition"] = 'attachment; filename="pic.JPG"'
message.attach(att2)



# Create socket called clientSocket and establish a TCP connection with
mailserver
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((mailserver, 25))
recv_tcp = clientSocket.recv(1024) .decode()
print(recv_tcp)

# Send HELO command and print server response.
helo = 'HELO Alice\r\n'
clientSocket.send(helo.encode())
recv_helo = clientSocket.recv(1024).decode()
print(recv_helo)

# Send AUTH LOGIN command and print server response.
clientSocket.sendall('AUTH LOGIN\r\n'.encode())
recv = clientSocket.recv(1024).decode()
print(recv)

clientSocket.sendall((username + '\r\n').encode())
recv = clientSocket.recv(1024).decode()
print(recv)

clientSocket.sendall((password + '\r\n').encode())
recv = clientSocket.recv(1024).decode()
print(recv)

# Send MAIL FROM command and print server response.
mail_from =  'MAIL FROM: <' + fromaddress + '>\r\n'
clientSocket.send(mail_from.encode())
recv_from = clientSocket.recv(1024).decode()
print(recv_from)

# Send RCPT TO command and print server response.
rcpt_to = 'RCPT TO: <' + toaddress+ '>\r\n'
clientSocket.send(rcpt_to.encode())
recv_to = clientSocket.recv(1024).decode()
print(recv_to)

# Send DATA command and print server response.
data = 'DATA\r\n'
clientSocket.send(data.encode())
recv_data = clientSocket.recv(1024).decode()
print(recv_data)

# Send message data.
```

```
msg_data = 'from: ' + fromaddress + '\r\n'
msg_data += 'to: ' + toaddress + '\r\n'
msg_data += 'subject: ' + subject + '\r\n'
msg_data += 'content-type: text/plain\t\n'
msg_data += '\r\n' + message.as_string()
clientSocket.send(message.as_string().encode())


# Message ends with a single period and print server response.
endmsg = '\r\n.\r\n'
clientSocket.send(endmsg.encode())
recv_end = clientSocket.recv(1024).decode()
print(recv_end)

# Send QUIT command and print server response.
quit_msg = 'QUIT\r\n'
clientSocket.send(quit_msg.encode())
recv_quit = clientSocket.recv(1024).decode()
print(recv_quit)

# Close connection
clientSocket.close()
```

2-4

```
from socket import *
import threading
import os

# Define thread process
def Server(tcpClisock, addr):

    BUFSIZE = 1024
    print('Received a connection from:', addr)
    data =  tcpClisock.recv(4096).decode()
    print(data)

    if len(data):
        # Extract the filename from the received message
        getFile = data.split()[1]
        print('getFile:',getFile)

        # Form a legal filename
        filename =  getFile.partition("/")[-1].partition("/")[-1].partition("/")
[-1]
        print('filename:',filename)

        # Check wether the file exist in the cache
        if os.path.exists(filename):
            print('File exist')
            # ProxyServer finds a cache hit and generates a response message
            f = open(filename,"r")
            CACHE_PAGE = f.read()
            # ProxyServer sends the cache to the client
            tcpClisock.send(CACHE_PAGE.encode())
```

```
            print('Send the cache to the client')
            tcpClisock.close()
        else:
            print('File not exist')
            # Handling for file not found in cache
            # Create a socket on the ProxyServer
            c =  socket(AF_INET, SOCK_STREAM)
            try:
                # Connect to the WebServer socket to port 80
                hostn = getFile.partition("/")[2].partition("/")[0]
                c.connect((hostn,80))
                print('Connect to',hostn)

                # Some information in client request must be replaced before it
can be sent to the server
                modified_data = data.replace('/'+hostn,'')
                modified_data = modified_data.replace(host+':'+str(port),hostn)
                # Send the modified client request to the server
                c.sendall(modified_data.encode())

                # Read the response into buffer
                buff = c.recv(4096)
                print("buff:\n")
                print(buff.decode())
                print('recvbuff len:',len(buff))

                # Send the response in the buffer to client socket
                tcpClisock.send(buff)
                print('Send to client\r\n')
                # Create a new file to save the response in the cache for the
requested file
                tmpFile = open(filename,"w")
                tmpFile.writelines(buff.decode())
                tmpFile.close()
                i=0
                with open(filename,"r",encoding="utf-8") as f:
                    lines = f.readlines()
                with open(filename,"w",encoding="utf-8") as f_w:
                    for line in lines:
                        i+=1
                        if i >9:
                            f_w.write(line)
            except:
                print("Illegal request")
            tcpClisock.close()

# Main process of  ProxyServer
if __name__ == '__main__':

    # Create a server socket, bind it to a port and start listening
    tcpSersock = socket(AF_INET, SOCK_STREAM)
    host = 'localhost'
    port = 8088
    tcpSersock.bind((host,port))
    tcpSersock.listen(5)

    print("Ready to serve......\n")
    while True:
```

```
        tcpClisock, addr = tcpSersock.accept()
        thread = threading.Thread(target=Server, args=(tcpClisock, addr))
        thread.start()
    tcpSersock.close()
```

5-1

```
from socket import *
import os
import sys, getopt
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8

def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = string[count] * 256 + string[count+1]
        csum = csum + thisVal
        csum = csum & 0xffffffff
        count = count + 2

    if countTo < len(string):
        csum = csum + string[len(string) - 1]
        csum = csum & 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def receiveOnePong(mySocket, destAddr, ID, sequence, timeout):
    timeLeft = timeout

    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []: # Timeout
            return "Request timed out."
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

        #Fill in start

        #Fetch the ICMP header from the IP packet
```

```
        header = recPacket[20: 28]
        type,code,checksum,id,sequence = struct.unpack("!BBHHH", header)
        if type == 0 and id == ID:
            bytes0 = struct.calcsize("!d")
            timeSent = struct.unpack("!d", recPacket[28: 28 + bytes0])[0]
            delay = timeReceived - timeSent
            ttl = ord(struct.unpack("!c", recPacket[8:9])[0])
            return (delay, ttl, bytes0)

        #Fill in end
        if(type == 3 and code == 0):
                return "目的网络不可达"
        if(type == 3 and code == 1):
            return "目的主机不可达"
        if(type == 3 and code == 2):
            return "目的协议不可达"
        if(type == 3 and code == 3):
            return "目的端口不可达"
        if(type == 3 and code == 6):
            return "目的网络未知"
        if(type == 3 and code == 7):
            return "目的主机未知"
        if(type == 4 and code == 0):
            return "源抑制"
        if(type == 9 and code == 6):
            return "路由器通告"
        if(type == 10 and code == 0):
            return "路由器发现"
        if(type == 11 and code == 0):
            return "TTL过期"
        if(type == 12 and code == 0):
            return "IP首部损坏"

        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return "Request timed out."
        return "Request error."


def sendOnePing(mySocket, destAddr, ID, sequence):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)

    myChecksum = 0
    # Make a dummy header with a 0 checksum
    # struct -- Interpret strings as packed binary data
    header = struct.pack("!BBHHH", ICMP_ECHO_REQUEST, 0, myChecksum, ID,
sequence)
    data = struct.pack("!d", time.time())

    # Calculate the checksum on the data and the dummy header.
    myChecksum = checksum(header + data)

    # Get the right checksum, and put in the header
    if sys.platform == 'darwin':
        # Convert 16-bit integers from host to network byte order
        myChecksum = htons(myChecksum) & 0xffff
    else:
```

```
        myChecksum = htons(myChecksum)

    header = struct.pack("!BBHHH", ICMP_ECHO_REQUEST, 0, myChecksum, ID,
sequence)
    packet = header + data

    mySocket.sendto(packet, (destAddr, 1)) # AF_INET address must be tuple, not
str
    # Both LISTS and TUPLES consist of a number of objects
    # which can be referenced by their position number within the object.

def doOnePing(destAddr, ID, sequence, timeout):
    icmp = getprotobyname("icmp")

    # SOCK_RAW is a powerful socket type. For more details:
    #http://sock-raw.org/papers/sock_raw
    #Fill in start

    #Create Socket here
    mySocket = socket(AF_INET, SOCK_RAW, icmp)

    #Fill in end

    sendOnePing(mySocket, destAddr, ID, sequence)
    rtt = receiveOnePong(mySocket, destAddr, ID, sequence, timeout)

    mySocket.close()
    return rtt

def ping(dest, count):
    # timeout=1 means: If one second goes by without a reply from the server,
    # the client assumes that either the client's ping or the server's pong is
lost
    timeout = 1

    myID = os.getpid() & 0xFFFF  # Return the current process i
    loss = 0
    delay_set = []

    # Send ping requests to a server separated by approximately one second
    for i in range(count) :
        result = doOnePing(dest, myID, i, timeout)

        if(type(result) == str):
            print(result)
            loss = loss + 1
            continue
        delay = int(result[0]*1000)
        ttl = result[1]
        bytes = result[2]
        delay_set.append(delay)
        print("Reply from host_ipaddr "+ dest + ": bytes= "+ str(bytes)+" time=
"+str(delay)+"ms TTL= "+ str(ttl))

        #Fill in end

        time.sleep(1)# one second
```

```
    #Fill in start
    #Print Ping statistics
    import numpy as np
    print("Packet: sent = " + str(count) + " received = " + str(count-loss) + "
lost = " + str(loss) + " loss rate = " + str(100*float(loss)/count) + "%")
    if delay_set != []:
        print("max rtt : " + str(max(delay_set))+"ms "+"min rtt : " +
str(min(delay_set))+"ms "+"average rtt : " + str(np.mean(delay_set))+"ms ")
    else:
        print("rtt does not exist.")
    #Fill in end

    return

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print('IcmpPing.py dest_host [-n <count>]')
        sys.exit()
    host = sys.argv[1]
    try:
        dest = gethostbyname(host)
    except:
        print('Can not find the host "%s". Please check your input, then try
again.'%(host))
        exit()

    count = 4
    try:
        opts, args = getopt.getopt(sys.argv[2:], "n:")
    except getopt.GetoptError:
        print('IcmpPing.py dest_host [-n <count>]')
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-n':
            count = int(arg)

    print("Pinging " + host + " [" + dest + "] using Python:")
    print("")
    ping(dest, count)
```