Homework 2

Total points : 120 | Maximum points : 100

Due: Tuesday 11:59 PM, October 10,2023

1. (12 points) Identify the OSI layer responsible for each of the following functionalities:
    a. Providing node-to-node communications with reliable service.
    b. Determining the best path to route packets.
    c. Providing end-to-end communications with reliable service.

2. (8 points) Suppose that you have two browser applications open and active at the same time and that both applications are accessing the same server to retrieve HTTP documents at the same time. How does the server know how to tell the difference between the two applications (to send the correct document(s) to the correct application)?

3. (10 points) Suppose that we want to distribute a file with a size of $F = 20$ Gbits to $N = 100$ clients/peers. The server supports an upload rate of $u_s = 30$ Mbps while each client/peer has a download rate of $d_i = 2$ Mbps and an upload rate of u, where $u = 300$ Kbps or 700 Kbps. Show your calculations.
    a. Calculate the minimum distribution time for a client-server distribution using the two values of u given above.

    b. Calculate the minimum distribution time for a peer-to-peer distribution using the two values of u given above.

4. (10 points) TCP/IP over Ethernet supports basic frames with a total size of up to 1518 bytes (including both the message payload and headers). Suppose that an application protocol wants to send an L-byte message across the network to its peer over TCP/IP. The TCP segment adds 20 bytes of header to the payload, while IP packet adds an additional 20 bytes of header to the segment. If the Ethernet frame adds 18 bytes of header to the packet, calculate the size of L the application needs so that exactly 95% of the transmitted bits in the physical layer carry the message payload (i.e., the data itself, not the header). Show your calculations.

5. (80 points) UDP Pinger programming assignment:

In this assignment, you will write two complete python programs to support a client/server model using Linux sockets for a UDP "ping" utility, similar to the ping utility already available on ECS coding machines.

- **Server**
  - The server program will be called with one command-line argument, the port number being used, such as python3 pingsvr.py 8001. If the user calls the server program with too few or too many arguments, you will print out a usage statement and terminate the program. You will run this command on **ecs-coding1.csus.edu.**
  - The server will set up a UDP socket on the Internet (i.e., INET) domain and then wait in an infinite loop listening for incoming UDP packets, specifically PING messages from a client. Your server should be able to support multiple clients at the same time (though no extra work is expected to support this requirement).
  - When a PING message comes in from a client and if the packet is not lost, the server will print the client message to the terminal and then send a PONG message back to the client. If the packet is determined to be lost, the server will print an appropriate message to the terminal and simply "eat" the message by not responding to the client.
  - The server will remain "always on" until a user enters Ctrl-C(^C) to send an interrupt signal to the program to terminate.

- **Client**
  - The client program will be called with two command-line arguments, the hostname of the server and the port number being used, such as python3 pingcli.py ecs-coding1.csus.edu 8001. If the user calls the client program with too few or too many arguments, you will print out a usage statement and terminate the program. You will run this command on **ecs-coding2.csus.edu.**

- The client will send 10 automated PING messages to the server using a UDPsocket, where automated means the message is built in the code, not entered from the keyboard. Because UDP is an unreliable protocol, a packet sentfrom the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a PING message. You should get the client to wait up to one second for a reply – if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network.
- Specifically, for each of the 10 PING messages, your client program should:
  - send the PING message using the UDP socket and print a status message.
  - if the response message is received from the server, calculate and print the round trip time (RTT) in milliseconds for each message; otherwise, print a status message that it timed out.
- After all of the PING messages have been sent (and responses received or timed out), the client program should report the following and then terminate:
  - the number of messages sent, the number of messages received, and the message loss rate (as a percentage);
  - the minimum, maximum, and average RTTs for all of the PING messages in milliseconds.

Your program should run on the INET domain using SOCK_DGRAM (i.e., UDP) sockets so that the server and the client execute on a different ECS machine.

You will also need to make sure you are able to handle any error cases.

SAMPLE OUTPUT (user input shown in **bold**)**:**

==> SERVER on ecs-coding1

$ **python3 pingsvr.py**

usage: python3 pingsvr.py <port>

$ **python3 pingsvr.py 8001**

UDP Ping Server is listening on port 8001

Received PING from ('130.86.188.34', 59007)

Sent PONG to ('130.86.188.34', 59007)

Received PING from ('130.86.188.34', 59007)

Sent PONG to ('130.86.188.34', 59007)

Received PING from ('130.86.188.34', 59007)

Sent PONG to ('130.86.188.34', 59007)

Received PING from ('130.86.188.34', 59007)

Sent PONG to ('130.86.188.34', 59007)
Received PING from ('130.86.188.34', 59007)
Sent PONG to ('130.86.188.34', 59007)
Packet from ('130.86.188.34', 59007) was lost.
Received PING from ('130.86.188.34', 59007)
Sent PONG to ('130.86.188.34', 59007)
Packet from ('130.86.188.34', 59007) was lost.
Received PING from ('130.86.188.34', 59007)
Sent PONG to ('130.86.188.34', 59007)
Received PING from ('130.86.188.34', 59007)
Sent PONG to ('130.86.188.34', 59007)
^C

==> CLIENT on ecs-coding2
$ **python3 pingcli.py**
usage : python3 pingcli <hostname> <port>
$ **python3 pingcli.py ecs-coding1.csus.edu**
usage : python3 pingcli <hostname> <port>
$ **python3 pingcli.py ecs-coding1.csus.edu 8001**
Received PONG from ('130.86.188.33', 8001) (RTT: 0.28 ms)
Received PONG from ('130.86.188.33', 8001) (RTT: 0.14 ms)
Received PONG from ('130.86.188.33', 8001) (RTT: 0.16 ms)
Received PONG from ('130.86.188.33', 8001) (RTT: 0.10 ms)
Received PONG from ('130.86.188.33', 8001) (RTT: 0.13 ms)
Request timed out for sequence 6
Received PONG from ('130.86.188.33', 8001) (RTT: 0.26 ms)
Request timed out for sequence 8
Received PONG from ('130.86.188.33', 8001) (RTT: 0.26 ms)
Received PONG from ('130.86.188.33', 8001) (RTT: 0.12 ms)

Ping statistics:
Sent = 10, Received = 8, Loss rate = 20.00%
Minimum RTT = 0.10 ms, Maximum RTT = 0.28 ms, Average RTT = 0.18 ms

Code Requirements:

- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.
- Your programs should be named "pingsvr.py" and "pingcli.py", without the quotes, for the server and client code, respectively.
- Your program will be graded based largely on whether it works correctly on the ECS machines (e.g., ecs-coding1, ecs-coding2, …, ecs-coding3), so you should make sure that your program compiles and runs on a ECS machine.
- Please pay attention to the SAMPLE OUTPUT for how this program is expected to work. If you have any questions about this, please contact your instructor, assigned to this course to ensure you understand these directions.

Submission Files :

- You will submit the server code, client code and a pdf file consisting of answers to questions 1-4.

Note: The assignment is graded out of 120 points. Your grade will not exceed 100.