**HW1: Computer Networks**

**Problem 1. [CS4473 and CS5473] (60 points)**

Suppose that multiplication of two numbers is a computationally expensive calculation and you have developed a killer app for multiplication which runs on a server equipped with a powerful CPU. Users can send two numbers to the server and receive their product back using UDP. The multiplication is done “in the cloud”, instead of locally on their own computer. In this programming problem, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout.

You are given the complete code for the Ping server below. Your task is to write the Ping client.

Server Code

Please download the server code from Canvas. You need to compile and run the server code before running your client program. You do not need to modify the server code.

The server sits in an infinite loop listening for incoming UDP packets. The message should contain 3 numbers. The first two numbers should be multiplied. To detect communication errors, the third number is provided by the users as the sum of the first two numbers. If the server finds that the sum is incorrect, the server will send a message to the client to re-transmit the original message.

UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Although UDP uses checksum to verify the integrity of a message, it does not do anything to recover an error. Let us implement the functionalities to recover from message losses and errors in the application layer. Because such events are rare or even non-existent in typical networks, the server in this lab injects artificial loss and modification to simulate the effects of unreliable communication. A message has a 25% chance to be lost and a 25% chance to have a modified number out of the three submitted numbers.

You should study this code carefully, as it will help you write your client.

Client Code

You need to implement the following client program:

1. Get the server IP address, and the server port number, and the two numbers to be multiplied from the user as command line arguments in this order.
2. Compute the sum of the two numbers
3. Send the three numbers to the server using UDP
4. Because UDP is an unreliable protocol, a packet sent from 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 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. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket. After timeout, the client should re-send the message.
5. If the server responds with an error message for the incorrect sum, the client should also re-send-the message.
6. The client should repeat step #4 and step #5 until a valid response is received from the server. Please print out the result. For every request, calculate and print the round trip time (RTT), in seconds, of each packet, if the server responses. Otherwise, print “Request timed out”.

During development, you should run the UDP\_Multiplier\_Server.py on your machine, and test your client by sending packets to *localhost* (or, 127.0.0.1). We have also kept the server program running at two hosts in cloud with IP addresses 34.68.41.35 and 35.186.147.8. Test your client by sending packets to these two hosts also. Finally, compare the three RRTs obtained, i.e. sending packets to the server program running on your machine and the two hosts in the cloud.

Message Formats

The client message is one line, consisting of ASCII characters in the following format:

Multiply *number1 number2 sum time*

where *number1*and *number2* needs to be multiplied, *sum* is the sum of these two numbers, and *time* is the time when the client sends the message.

The service message is either a string “*Incorrect sum*” or a number for the product of the two numbers.

What to Hand in

You will hand in the complete client code and screenshots at the client verifying that your ping program works as required.

**Problem 2. [CS4473 and CS5473] (20 points)**

Please think about the transmission delays covered in Chapter 1 and let us consider sending real-time voice from Host A to Host B over a packet-switched network (VoIP). Host A converts analog voice to a digital 64 kbps bit stream on the fly. Host A then groups the bits into 56-byte packets. There is one link between Hosts A and B; its transmission rate is 2 Mbps and its propagation delay is 10 msec. As soon as Host A gathers a packet, it sends it to Host B. As soon as Host B receives an entire packet, it instantaneously converts the packet’s bits to an analog signal. How much time elapses from the time a bit is created (from the original analog signal at Host A) until the bit is decoded (as part of the analog signal at Host B)?

**Problem 3. [CS4473 and CS5473] (20 points)**

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

* In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?
* If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?
* If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

**Problem 4. [CS5473 only] (20 points)**

UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?