COMP 3331/9331: Computer Networks and Applications

Practice Programming Lab

Aim

The goal of this lab is to provide students with hands-on practice in socket programming. Note that, the lab is not marked. You are not required to submit your completed programs. If you are unable to complete the program during the lab duration, you can always continue with your development on your own. Some optional enhancements (some of which are difficult) are listed towards the end, for students who are keen for the challenge. The skills learnt in completing the practice programs will be useful for the programming assignments.

In this lab, you will study a simple Internet ping server (written in the Java language), and implement a corresponding client. The functionality provided by these programs are similar to the standard ping programs available in modern operating systems, except that they use UDP rather than Internet Control Message Protocol (ICMP) to communicate with each other. Note that, we will study ICMP later in the course. (Java does not provide a straightforward means to interact with ICMP)

The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

Ping Server (provided)

You are given the complete code for the Ping server (PingServer.java – download from the lab exercises page). The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the server simply sends the encapsulated data back to the client.

Your job is to write the corresponding Ping client. You can use either C or Java to write your client. You should read through the server code thoroughly, as it will help you with the development of your client program.

Packet Loss

UDP provides applications with an unreliable transport service, because messages may get lost in the network due to router queue overflows or other reasons. In contrast, TCP provides applications with a reliable transport service and takes care of any lost packets by retransmitting them until they are successfully received. Applications using UDP for communication must therefore implement any reliability they need separately in the application level (each application can implement a different policy, according to its specific needs). Because packet loss is rare or even non-existent in typical campus networks, the server in this lab injects artificial loss to simulate the effects of network packet loss. The server has a parameter LOSS_RATE that determines which percentage

of packets should be lost. The server also has another parameter AVERAGE_DELAY that is used to simulate the delay incurred by a packet inthe Internet. You should set AVERAGE_DELAY to a positive value when testing your client and server on the same machine, or when machines are close by on the network. You can set AVERAGE_DELAY to 0 to find out the true round trip times of your packets.

Compiling and Running Server

To compile the server, do the following:
>javac PingServer.java

To run the server, do the following:
>java PingServer port

where *port* is the port number the server listens on. Remember that you have to pick a port number greater than 1024, because only processes running with root (administrator) privilege can bind to ports less than 1024. If you get a message that the port is in use, try a different port number. Other students may be using the port number that you have chosen.

Note: if you get a class not found error when running the above command, then you may need to tell Java to look in the current directory in order to resolve class references. In this case, the commands will be as follows:
>java -classpath . PingServer port

Your Task: Implementing Ping Client

You should write the client so that it sends 10 ping requests to the server, separated by approximately one second. Each message contains a payload of data that includes the keyword PING, a sequence number, and a timestamp. After sending each packet, the client waits up to one second to receive a reply. If one seconds goes by without a reply from the server, then the client assumes that its packet or the server's reply packet has been lost in the network.

You should write the client so that it starts with the following command:

>java PingClient host port OR
> PingClient host port (if using C)

where *host* is the IP address of the computer the server is running on and *port* is the port number it is listening to. In your lab you will be running the client and server on the same machine. So just use 127.0.0.1 (i.e., localhost) for *host* when running your client. In practice, you can run the client and server on different machines.

The client should send 10 pings to the server. Because UDP is an unreliable protocol, some of the packets sent to the server may be lost, or some of the packets sent from server to client may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should have the client wait up to one second for a reply; if no reply is received, then the client should assume that the packet was lost during transmission across the network. It is important that you choose a reasonably large

value, which is greater than the expected RTT (Note that the server artificially delays the response using the AVERAGE_DELAY parameter). In order to achieve this your socket will need to be non-blocking (i.e. it must not just wait indefinitely for a response from the server). If you are using Java, you will need to research the API for DatagramSocket to find out how to set the timeout value on a datagram socket (Check: http://java.sun.com/javase/6/docs/api/java/net/Socket.html). If you are using C, you can find several options here: http://rhoden.id.au/doc/sockets2.html. Note that, the *fcntl()* function is the simplest way to achieve this.

Note that, your client should not send all 10 ping messages back-to-back, but rather sequentially. The client should send one ping and then wait either for the reply from the server or a timeout before transmitting the next ping. Upon receiving a reply from the server, your client should compute the RTT, i.e. the difference between when the packet was sent and the reply was received. There are functions in Java and C that will allow you to read the system time in milliseconds. The RTT value should be printed to the standard output (similar to the output printed by ping; have a look at the output of ping for yourself). An example output could be:

ping to 127.0.0.1, seq = 1, rtt = 120 ms

Of course, you should exclude pings for which no reply was received from the server.

Message Format

The ping messages in this lab are formatted in a simple way. Each message contains a sequence of characters terminated by a carriage return (CR) character (\r) and a line feed (LF) character (\n). The message contains the following string:

PING sequence_number time CRLF

where *sequence_number* starts at 0 and progresses to 9 for each successive ping message sent by the client, *time* is the time when the client sent the message, and CRLF represent the carriage return and line feed characters that terminate the line.

Hint: Cut and paste PingServer, rename the code PingClient, and then modify the code, if you are using Java.

Optional Exercises

When you are finished writing the client, you may wish to try one of the following exercises:

1) Currently the program calculates the round-trip time for each packet and prints them out individually. Modify this to correspond to the way the standard ping program works. You will need to report the minimum, maximum, and average RTTs. (easy)

2) The basic program sends a new ping immediately when it receives a reply. Modify the program so that it sends exactly 1 ping per second, similar to how the standard ping program works. Hint: Use the *Timer* and *TimerTask* classes in *java.util*. (difficult)

3) Develop two new classes *ReliableUdpSender and ReliableUdpReceiver*, which are used to send and receive data reliably over UDP. To do this, you will need to devise a protocol (such as TCP) in which the recipient of data sends acknowledgements back to the sender to indicate that the data has arrived. You can simplify the problem by only providing one-way transport of application data from sender to recipient. Because your experiments may be in a network environment with little or no loss of IP packets, you should simulate packet loss. (difficult)

NOTE: You are not required to submit your programs. The lab is not marked.