Hampton Terry
Kevin Tang
1/24/2013
CSE461 - Project 1

1. *Why might the UDP ping client time out?*
   Suppose your UDP ping client experiences a timeout while waiting for a response. List all distinct possible causes of the timeout.

   Packet is sent from the client, but is dropped before reaching the server.

   Packet reaches the server but something happens so that the server does not respond. Could be caused by an error in the in the server in dealing with the packet such as an overflow error caused by the packet or a bad header.

   Server receives the packet and responds but the packet is lost on the way back.

   The server sends a response and the response reaches the client, but the response has a bad header or some other error.

2. *What is an appropriate response to a timeout?*
   Suppose your ping client is asked to make 100 trials UDP pings. The client code experiences a timeout on trial number 5, and no other timeouts.
   a. There's an argument that when the timeout occurs, the client code should give up, not perform the remaining 95 trials, and return "I don't know" as the estimated ping time. What is that argument?

Essentially, a timeout means that the result of the trial is failure. So when trying to take the average response times of the trials, the failure must be accounted for. It might not make sense to return only the results of the successful trials, therefore if there is a timeout, a result should not be returned.

   b. There's also an argument that the code should go ahead and perform the remaining 95 trials, returning the average of the 99 successful trials and an indication that one trial failed. (This response is interpreted as "When the ping is successful, the average latency is xx msec., but it's successful only Y% of the time," which isn't quite the original goal but is still something worth knowing.) Briefly explain that

argument.

This argument also makes sense because it gives the times for the trials as well as a failure percentage and this way it can provide information about the average latency for just the successful trials, but also provide a success percentage to help someone interpret the data more correctly.

c. There's no argument that you should just keep running trials until you have 100 completed without timeout, and then simply return the mean of those 100 trials. Briefly explain why not.

First, this completely ignores that some trials will timeout. No information is given in this case. A trial with a lower average latency on success but a success but has a timeout percentage of 50% really does not mean that much.
Second, this system could be artificially manipulated to produce a specific set of results. If for example, some connection has a latency of 20-30 ms, and this version of ping is run with a timeout of 22 ms. This ping would produce the result that the average ping would be somewhere between 20 and 22 ms.
Third is the consideration that the trials always fail. Will this ping continually send packets until it reaches 100 trials even when it never gets a response back?

d. Very briefly, what does your UDP ping code implementation do when a timeout occurs? (This is just a question of what you actually do. You don't have to have done "the best imaginable thing." In fact, the project discourages you from trying to do the best imaginable thing because it's much too complicated for this project.)

The ping code for our UDP ping does something similar to B but not quite. We do not go back and do trials if they have resulted in timeout, and we just continue on and throw away the timeout trials. It however, does print out that a timeout has occurred and therefore gives some indication that some error occurred.

3. The UDP ping protocol we asked you to implement doesn't actually work, in some abstract sense at least. Explain how it can fail.
   Hint: Consider the case where the caller's timeout is relatively short, say, about the same as the time it will take to receive the server's response in the normal case.

The way UDP ping is implemented, it only uses one socket to handle all the clients pinging the server. If the server is in the middle of processing

one request when another arrives, the second client's request will have to wait until the server is finished processing the previous requests. This is a problem because it could cause ping requests to timeout or return slower and inaccurate latencies simply because the server is busy processing other requests.

4. In contrast, the TCP ping protocol does work, not because TCP is reliable and UDP isn't, but because our ping implementation using TCP creates a new connection for each ping attempt. Explain how to adapt that solution to the UDP case. (I.e., how could you change the UDP implementation so that it solved the problem in a manner analogous to the TCP implementation?)

We would modify our UDP implementation to create a new connection for each request. This will allow multiple connections to be processed at the same time, and each ping request will return after the correct time period.

5. The solution developed in the preceding question is "heavyweight," in the sense of generating a lot of seemingly unnecessary overhead. Explain how to fix the UDP problem by changing the format of the echo header, but without modifying how you use UDP sockets.

Rather than creating a new socket for each connection, we will have one socket for receiving all packets coming from all connections. For each connection, we will fork another thread to handle the request. This allows all requests to be processed at the same time, without the overhead of creating a lot of sockets. We will modify the header of each packet to include an unique identifier for each packet(possible the clients IP) so the server knows where to send the packet to. We will also add to the header a sequence number that is unique to each connection. This allows clients who are sending multiple packets to match their received packets with their request packets.