Overview. This week we'll focus on simulation of “real” conditions in a network. In particular, this week’s project is to write a “ping”-like client application that uses a simulated echo server (which I will provide).

The server, called “UDPPingServer.java”, may be downloaded from the course Web site. It receives UDP packets containing a single line of text. The text consists of the word PING, followed by an integer (a sequence number, used purely for seeing what’s going on), followed by a time in nanoseconds. The three items are terminated by a ‘\n’. For instance, the packet might contain: PING 7 112347838272737\n, meaning that this is the eighth ping request in a row (sequence numbers begin at 0) and that the time when the client sent it is 112347838272737 (time is in nanoseconds and is measured from some unspecified fixed point).

The server simply returns the text, unchanged. A random number generator is used to introduce simulated delay in returning the reply. Furthermore, a random number generator is used to simulate packet loss — occasionally the server will not respond to a PING request.

Your client will need to send a number of PING requests to the server, spaced one second apart, and then calculate the average round-trip-time, or RTT, to the server and back. Here is the basic structure of the client. I have left out details about things like counting the number of successful RTT calculations, finding the average RTT, setting socket timeouts, etc.:

```java
create socket
for i = 0 to N-1 // N = number of trials; should be >= 20
  if no simulated packet loss,
    send PING + i + current time in nanoseconds
  else
    print error message (e.g., "Request i lost")
    continue
  end if
wait for response
if response received within one second
  note current time and calculate RTT
  print RTT for this ping request
else // socket timed out
  print error (e.g., "no response received")
end if
wait 1 second
end for
calculate and print average RTT for successful pings
```

Handed out on 21 September 2009

Handout #6
A Few Details

You can capture the current time using the function “System.nanoTime()”. Note that this returns a long value. Example:

```java
long startTime = System.nanoTime();
... event to be timed ...
long elapsedTime = System.nanoTime() - startTime;
```

You can introduce a one second delay using:

```java
Thread.sleep(1000); // assumes a "throws Exception" elsewhere
```

The only exception you need to explicitly catch is the SocketTimeoutException (when the ping server fails to respond, you should wait no longer than one second for a reply). However, you now know how to catch other exceptions as well, so use them as you see fit.

The next page shows some sample sessions.
Server:

$ java UDPPingServer 12345
Received from /141.195.226.27:4462:
  PING 0 346942913243941
  Reply sent.

Received from /141.195.226.27:4462:
  PING 1 346944158370258
  ERROR -- Reply gets lost.

Received from /141.195.226.27:4462:
  PING 2 346945158789089
  Reply sent.

  ...

Received from /141.195.226.27:4462:
  PING 7 346950584998162
  ERROR -- Reply gets lost.

Received from /141.195.226.27:4462:
  PING 9 346951589214401
  Reply sent.

  ...

Received from /141.195.226.27:4462:
  PING 19 346960038015704
  ERROR -- Reply gets lost.

Client:

$ java UDPPingClient aldenv27 12345
Sent to aldenv27/141.195.226.27:12345:
  PING 0 346942913243941
  Packet acknowledged; RTT = 244.582314 ms

Sent to aldenv27/141.195.226.27:12345:
  PING 1 346944158370258
  ERROR -- Socket timed out; no response

Sent to aldenv27/141.195.226.27:12345:
  PING 2 346945158789089
  Packet acknowledged; RTT = 36.774211 ms

  ...

Sent to aldenv27/141.195.226.27:12345:
  PING 7 346950584998162
  ERROR -- Socket timed out; no response

  ERROR -- Ping request 8 gets lost.

Sent to aldenv27/141.195.226.27:12345:
  PING 9 346951589214401
  Packet acknowledged; RTT = 37.904051 ms

  ...

Sent to aldenv27/141.195.226.27:12345:
  PING 19 346960038015704
  ERROR -- Socket timed out; no response

Average RTT: 109.9225123ms
(based on 10 RTTs)