# Redes de Computadores

## Lab #2

## Exercise (Cont.)

### UDP File Transfer

Implement a client/server application capable of transferring files between two computers...

- A client program (**sender**) sends a file to a server (**receiver**) content to the server. Since UDP datagrams are limited in size, the contents of file will need to be split into multiple datagrams.
- Design the protocol so that:
  1. The transfer starts with a datagram containing the name of the file to be transferred;
  2. The contents of the file will be sent in blocks of at most 1024 bytes;
  3. The client and server can determine when the transfer has finished.

## Experiments

1. Validate your file transfer protocol...
   - Transfer a file smaller than 1024 bytes;
   - Transfer a file exactly 1024 bytes;
   - Transfer a file larger than 1024 bytes.

   Does the transfer finish on both client and server and the resulting files match the source?
2. File transfer reliability...
   - Running the client and server on the same machine, transfer a large JPEG image file, sending the datagrams as fast as possible;
   - Running the client and server on different machines, transfer a large JPEG image file, sending the datagrams as fast as possible;
   - Note that both computers will need to on the same network and the respective firewalls disabled or configured appropriatelly.

   Can you view the image copy correctly? Or, does it appear corrupted or cannot be opened at all?
3. Try the previous experiment but insert a short delay, see below, before sending each datagram packet.
   - Experiment with different delays, starting with just 1 ms to see if it makes a difference.

   Does slowing down the transfer rate improve the chances of success?

## Next Steps

By now, it should be obvious that UDP does not ensure message delivery. It is a best-effort protocol after all.

**Adapt your UDP File Transfer solution so that it can deal with lost datagrams...**

Slowing down the transfer rate exposes the reason why datagrams are lost, but it is not the the actual solution.

The problem is that an one-way protocol does not work when the underlying network is not reliable. The sender and receiver need to work in concert...

The receiver needs to send some feedback to the sender so that lost messages can be re-sent. One simple approach is to have the receiver ***acknowledge*** each message...If the confirmation of a given message does not arrive within the reasonable time period, it needs to be resent...

## Socket timeouts

By default, socket operations are blocking, ie., socket.receive() will block until a datagram is received.

To change the behavior of socket and abort the socket.receive() after some ***timeout***, one can use the socket.setSOTimeout() method on the socket. From then on, receive() operations on that socket either return normally or throw an exception if the timeout has elapsed and no datagram was received...

### Wide Area Networks

In Wide Area Networks it's normal to experience a small but persistent rate pf UDP datagram loss. The class MyDatagramSocket can be used instead of the standard DatagramSocket class to simulate the behavior of UDP in a wide area network, in the local machine.

## Java Tips

### File I/O

- java.io.File - represents a file or folder in the filesystem. Can be used to query file attributes, including its length.
- java.io.FileOutputStream - for writing a file sequentially as a stream of bytes;
- java.io.FileInputStream - for reading a file sequentially as a stream of bytes. The stream read operation returns -1 when ***oef*** is reached.

### Sleeping

- Thread.sleep( ms ) - blocks the program the given time in milliseconds; Besides Facebook, it's a great time waster...

### Timing Events/Operations

- The difference between two System.currentTimeInMillis() measurements provides the elapsed time in milliseconds;
- For more precision, System.nanoTime() measures time in nanoseconds (1 nanosecond = 1e-9 seconds)