The programs in problems 1 and 2 can be used to create unidirectional pipes across a TCP/IP network. When paired they can be used to transfer files, etc.

## Problem 1 -- tcp_send

Write a program to copy a stream of bytes from standard input to a TCP port. It should have the following syntax:

```
tcp_send hostname port <input_stream
```

Accept *hostname* as either a text host name (e.g. `foo.bar.com`) or a decimal IP address (`192.168.10.20`). If a decimal IP address is supplied, your program should NOT attempt DNS resolution of the name. Open a TCP connection to the specified host and port number and read from stdin until EOF, relaying the bytes to the remote socket. Properly report and handle errors in looking up the hostname, making the connection, short writes, and read and write failures. Print to stderr at the conclusion of the program the number of bytes sent and the transfer rate in MB/s (the transfer rate is based on the time actually spent transferring the bulk data, not including connection setup). Your program should be able to handle any arbitrary byte stream just as cat does.

**Be certain that all data make it to the far end** before returning a successful exit code (0). Look up the `SO_LONGER` socket option and be prepared to discuss why it is important in this application.

## Problem 2 -- tcp_recv

Write a program having the following syntax:

```
tcp_recv port >output_file
```

Listen on the TCP port *port*. Note that ports below 1024 are not available to ordinary users on UNIX systems. The port you choose may also be in use by another program at the time you run yours, in which case you'll have to pick another number. Accept the first incoming connection, then copy all received data to stdout. Properly report and handle any errors. Print to stderr the number of bytes received and the data rate in MB/s (as defined in problem 1). Also report the remote endpoint, i.e. the IP address, reverse DNS lookup of IP address (if available) and port. Your program must be able to handle any arbitrary byte stream, just as cat does.

*Testing note:* Apparently you can test your tcp_send and tcp_recv against each other. You can also use the standard UNIX utility netcat to create either active-mode (connect) or server-mode (listen) sockets and use that for testing.