CS 450 – Introduction to Networking – Spring 2014
Homework Assignment 1
File Transfer and Data Bandwidth Analysis Tool

Due: **Monday 17 February. Electronic copy due at 10:30 A.M., Optional paper copy may be handed in during class.** (Note: the paper copy must match the electronic copy exactly.)

Overall Assignment

For this assignment you are to write two programs, a client and a server, that will allow the client to transmit a file to the server and record the effective bandwidth of the transmission. Then you will use the programs to measure the effective bandwidth of sending different sized files over different distances. (Relays will be provided in far-away places to provide for long distance round trips.)

Server Operation and Details

- The syntax for running the server shall be:

  ```
  myServer [ port ] ...
  ```

  - port is the port to listen to for new connections. Default value is 54321.
  - You may support additional arguments to appear after the ones shown here, provided they are well documented in your README file.

- The server shall (ultimately) perform the following tasks:
  1. Create a socket to listen for new connection requests.
  2. Use **select( )** to determine which socket(s) have data available.
     - If new connection requests are present, establish new connections and add the new sockets to the list of sockets to listen to. This will also involve increasing the number of current active connections.
     - If new data is available from any connected sockets, read the data and deal with it accordingly.
       - Each transmission will begin with a defined header, containing further information and the number of bytes of data that follow.
       - After reading in the header, the server should read in the number of bytes of data specified, either writing them out to a (newly created) named file, or discarding them, depending on the saveFile field in the header.
       - When all the data is received and processed, then the server should send back an acknowledgement reporting how many bytes of data were successfully received, by setting packet type to 2 and nbytes to the number of bytes received. No data should actually follow this header.
       - If the persistent field was false, close the connection after the acknowledgement has been sent. Otherwise keep listening.
  3. Repeat step 2 forever, or until something stops the program.
  4. Close all sockets when exiting.
- For the basic assignment the only information sent from the server to the clients are headers containing acknowledgements of completed file transfers. (However see optional enhancements.)

Client Operation and Details

- The syntax for running the client shall be:

```
myClient [ server ] [ port ] [ relay ] . . .
```

  - server is the name or IP address of the computer running the server program. Default is localhost, 127.0.0.1
  - port is the port on the server that is listening for TCP connection requests, default 54321.
  - relay is the remote relay to use, or "none". Default is "none".
  - If any argument is omitted, then all following arguments must also be omitted.
  - You may support additional arguments to appear after the ones shown here, provided they are well documented in your README file.

- The client shall (ultimately) perform the following tasks:

  1. Ask the user for the name of a file.
  2. Memory map the file to an address using mmap, to save actually reading it.
  3. Note the time, for timing purposes.
  4. Contact the server to establish a new TCP connection, using socket( ) and connect( ).
  5. Create a header (see below), and write it to the connected socket with send( ) or write( ).
  6. Write the file data to the socket.
  7. Read an acknowledgement, and report the time and overall transfer rate.
  8. If the user has additional files to transfer, repeat as necessary. If a persistent connection is used, then the overhead of establishing the connection can be saved, but a mechanism for specifically closing it at the end must be used. (One idea: Transmit a file of 0 bytes, with persistent set to 0. Or add another package type or field for close requests.)

Socket Details

- The server needs to perform the following tasks regarding sockets:
  1. Call **socket( )** to create a socket for accepting TCP connection requests. The type should be **AF_INET** (or **PF_INET**) and **SOCK_STREAM.**
  2. Call **bind( )** to assign a name (port number, sockaddr structure) to this socket.
     - See ip(7) for details on the format of IP addresses.
     - gethostbyname(3) returns IP addresses as char *. I.e. "200.100.50.10" has 14 chars.
     - inet_aton(3) fills in in_addr structs based on char * IP addresses.
     - gethostname(2) will get the name of the current host. (Then call gethostbyname( ))
  3. Call **listen( )** to mark this socket as ready to accept connection requests.
  4. Call **accept( )** to create new sockets based on connection requests received on the socket created in step 2. All further communications with this client will be through the new socket created by accept( ).
  5. Use **recv( )** and **send( )** or **read( )** and **write( )** to communicate with this client via the newly created socket.
  6. **Note:** You can use **select( )** to determine which socket(s) have data available to be read. Note that the data structure sent to select gets modified, so you need to set it up again before each call to select. **poll( )** is another alternative to consider.
  7. Call **close( )** when necessary to close down the socket.

- The client needs to perform the following tasks regarding sockets:

    1. Call **gethostbyname( )** to lookup the IP address of the server. (See above, under bind( ))
    2. Call **socket( )** to create a socket for communications with the server.
    3. Call **connect( )** to request a connection from the server.
        - Connect will need to know the name used by the server in the bind( ) step listed above.
        - Connect will fail if the client tries to connect before the server is ready. Check the return value, and use a busy loop (with a **sleep(1)** call in it) if connect fails with errno equal to ENOENT.
    4. Use **recv( )** and **send( )** or **read( )** and **write( )** to communicate with the server.

Evolutionary Development

It is recommended that the program be developed in the following order, making sure each step is working before beginning development on the next step:

1. Develop initially with the client and server running on the same computer, or on two adjacent computers in the computer lab.
2. Develop simple server and client programs that create sockets and establish basic communications. Verify that they can exchange data without any concern for the content at this point.
3. Implement the file transfer, and let the server assume there will only be a single client. **Select( )** is not needed in this case.
4. Modify the server to handle multiple connections. At this point the server will need to implement **select( )** to determine which connection(s), if any, have incoming data to process.
5. Work out any remaining bugs so that the client-server system efficiently provides accurate file transfer and data transfer rate reporting.
6. Run the program over a range of different distances using files of varying sizes, from a single byte to a file size that takes about a minute to transfer the file and get the acknowledgement back, using the provided relays as necessary.
7. Optional Enhancements – See below.

Data Header Format and Contents

A defined data header will be provided on the web site, and may be subject to slight changes as needed. As of this writing, version 2 of the header file defines the following fields, <u>**which must be in this order**</u>:

- int version; // Set this to 2 to match this documentation
- long int UIN; // The relays (see below) will drop connections without recognized UINs.
- int HW_number; // e.g. 1 for HW1, etc.
- int transactionNumber; // To identify multiple parts of one transaction.
- char ACCC[10], // your ACCC user ID name, e.g. astudent42
- filename[20]; // the name of the file being transmitted. i.e. the name where the server should store it.
- unsigned int From_IP, To_IP; // Ultimate destination and source, not the relay
- int packetType; // 1 = file transfer; 2 = acknowledgement
- unsigned long nbytes; // Number of bytes of data to follow the header
- int relayCommand; // Interpreted by relay. 1 = close connection
- int persistent; // non-zero: Hold connection open after this file
- int saveFile; // non-zero: Save incoming file to disk. Else discard it.
-
- char reserved[1000]; // Reduce if student-defined fields are added. Keep total size constant.

Remote Relays

In order to provide for long distance data transfers, a set of relays will be provided at known locations.

- Each relay will listen for new TCP connections at port 54321.
- When sending data to the relay for ultimate delivery to a server:
    - The To_IP field in the CS450 defined header should be the IP address of the server, i.e. the ultimate destination, not the IP of the relay.
    - The address specified in the connect( ) call should be the address of the relay, not the server.
    - (When not using a relay, the two addresses are both the same, that of the server.)
- The relay will make a connection to the server as soon as the first CS450 header comes through with the To_IP address of the server provided. After that, all further transmissions on the same connection should be directed to the same destination.
- Once the connection is fully established, the relay will read whatever data is sent by the client and relay it to the server, and vice versa. For this assignment the relay will not modify or disturb the data in any way, though it may log the transmissions that pass through it.
- Any data arriving at the relay without a valid header, or without an approved UIN number, will be dropped.
- If either the client or server closes their connection to the relay, the relay should recognize that fact, and will close the connection to the other end as well. Alternatively, if the relay sees a packet with the relayCommand set to 1, then it will close both sides of this connection.

Required Output

- All programs should print your name and CS account ID as a minimum when they first start.
- Beyond that, this program should perform the work described above.
- It may be appropriate to report some summary statistics when the program closes, such as the total amount of data transferred, average response rate, etc. More details on this may be provided in class and/or on the class web site.

Analysis

Once the program is up an working, use it to transfer files of various sizes between clients and servers at different distances away. (I.e. using different relays, or no relays at all.) Choose file sizes from 1 byte (overhead only) to files big enough to take approximately one minute to transfer. Report your findings in both a table and a plot. For this assignment your data analysis may be included as part of your readme file (see below) or as a separate document.)

Other Details:

- A makefile is required, that will allow the TA to easily build both your client and server. As always, you are free to develop wherever you wish, but the TA will be grading the programs based on their performance on the virtual computers provided by Amazon AWS for this class.
- MOSS will be used to check submissions not only against other submissions from this semester, but also against past submissions from previous semesters and any similar programs found on the web, so be sure to do your own work.

What to Hand In:

1. Your code, **including a makefile, and a readme file,** should be handed in electronically using Blackboard.
2. The intended audience for the readme file is a general end user, who might want to use this program to perform some work. They do not get to see the inner workings of the code, and have not read the homework assignment. You can assume, however, that they are familiar with the problem domain (e.g. computer networking.)
3. A secondary purpose of the readme file is to make it as easy as possible for the grader to understand your program. If there is anything special the grader should know about your program, be sure to document it in the readme file. In particular, if you add any additional command-line parameters or special fields to the data packets or do any of the optional enhancements, then you need to document what they are and anything special the TA needs to do to run your program and understand the results.
4. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
5. Make sure that your name appears at the beginning of each of your files. Your program should also print this information when it runs.
6. The readme file must specifically provide direction on how to run the program, i.e. what command line arguments are required and whether or not input needs to be redirected. It must also specifically document any optional information or command line arguments your program takes, as well as any differences between the printed and electronic version of your program.
7. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
8. A printed copy of your program, along with any supporting documents you wish to provide, (such as hand-drawn sketches or diagrams) may be provided **to the TA** on the day the assignment is due or the following school day. The hard copies must match the electronic copy exactly.
9. Make sure that your **name and your ACCC account name** appear at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

It is course policy that students may go above and beyond what is called for in the base assignment if they wish. These optional enhancements will not raise any student's score above 100 for any given assignment, but they may make up for points lost due to other reasons. Note that all optional enhancements need to be clearly documented in your readme files. The following are some ideas, or you may come up with some of your own:

- Provide additional functionality, such as bi-directional file transfer, directory(tree) transfer, data compression, or encryption. See the manual pages for ftp( ) and other file-transfer programs for ideas.
- Anything else you can think of – Check with the TA for acceptability.