CMPT 434
Computer Networks

Assignment One

Due: Friday January 25th, 6:00pm – late submissions will not be accepted.

Total Marks: 84

Submission Instructions: All assignment submissions must use the Moodle online submission system. Your submission should consist of a single compressed tar file that has been created using "tar -cvzf" on one of the Department's Linux machines. Your tar file should include, for Part A, all your source files, a makefile, a readme file that describes **everything that the marker will need to know** to run and test your code, and a single documentation/design file in either plain text or PDF format. The source files should include a separate C source file for each implemented server and proxy server. All file names should clearly indicate which server or proxy the file is for. With respect to the documentation/design file – think carefully about what the marker needs to know to evaluate what you have done. In addition to a description of what you have done and justification for your design decisions, you should carefully describe any limitations that your solution has. If the marker discovers limitations that you have not described in your documentation, the marker will conclude that your testing was insufficient and you will be docked marks accordingly. Your tar file should also include a separate plain text or PDF file with your answers for Part B.

PART A

In this part you will implement several servers and a client for a simple transfer service for text files, in C. Solutions should work on the Department's Linux machines. Your client should take as command line arguments the host name and port number of the server (or proxy server – see below) that it should connect to, and read commands from `stdin`. Your client should support the following commands:

- **get** *localfilename remotefilename* : retrieves from the server the contents of the text file with name *remotefilename*, and writes these contents to a new text file that the client creates with name *localfilename*
- **put** *localfilename remotefilename* : sends to the server the contents of the text file with name *localfilename*; the server should then write these contents to a new text file that it creates with name *remotefilename*
- **quit** : for termination of the client

For simplicity, your servers and client can be single-threaded, handling only one session and operation at a time. However, you should think about what complexities would arise in your implementations if you did need to support concurrency. Note that you will need to figure out how best to handle error conditions (e.g., if try to create a file that already exists or read

from a non-existent or unreadable file), and how the receiver of file contents will know when it has the entire file.

You must use the proper socket API functions for TCP and UDP servers and clients as described for example in Beej's guide to network programming. Note that with our lab configuration, only port numbers between 30000 and 40000 should be used. Program defensively, checking for possible error conditions.

1. (*20 marks*) Design and implement a server and client using TCP for communication.

2. (*20 marks*) Design and implement a TCP-based "proxy" server. Your proxy should take as command line arguments the host name and port number for your server from question 1, which it will need to connect to. Your client from question 1 should connect to your proxy. Oddly, your proxy should have a particular liking for the characters "c", "m", "p", and "t", and whenever these are encountered in a file, these characters should be duplicated. So, for example, if a file that is being transferred includes a line "already it was impossible to say which was which", your proxy should transform that into "already itt was immppossible tto say which was which".

3. (*20 marks*) Design and implement a UDP-based server. Also implement a proxy server that uses TCP to communicate with your client, and UDP to communicate with your UDP-based server. As in question (1), your proxy should have a particular liking for the characters "c", "m", "p", and "t", and whenever these are encountered in a file, these characters should be duplicated. For simplicity, you can make the assumption that UDP segments are always received correctly and in-order.

PART B

1. (*4 marks*) Consider two nodes connected by a single dedicated link within an OCN (on-chip network), SAN (system/storage area network), LAN (local area network), or WAN (wide-area network), and suppose that we wish to transmit a single 100 byte (including header) packet from one node to the other. Calculate the **total delay** and the **percentage of the total delay that is propagation delay** for each network, assuming that:

   - the link data rate is 4 Gbps (i.e., 4×10^9 bps);
   - there is no queuing delay;
   - the total node processing delay (not overlapped with any other component of delay) is x + (0.5 nanoseconds/byte), where x (the portion of this delay that is independent of packet size) is 0 microseconds for the OCN, 0.3 microseconds for the SAN, 3 microseconds for the LAN, and 30 microseconds for the WAN; and
   - the link distances are 0.5 cm, 5m, 5000m, and 5000 km, for the OCN, SAN, LAN, and WAN, respectively, with the speed of signal propagation in each case equal to 200,000 km/s (approximately 2/3 of the speed of light in a vacuum).

   Be sure to show your work, both for this question and for subsequent questions, so that you can get part marks even if your numerical answers happen to be incorrect.

2. (*4 marks*) Consider a message consisting of *n* packets, each of 1000 bytes (including header), that traverses 4 links in a store-and-forward packet-switched network (i.e., there are 3 intermediate nodes between the source and destination that the message passes through). The propagation delay on each link is 1 ms. The transmission rate on each link is 100 Mbps (i.e., 100 x 10^6 bps). Neglecting processing and queueing delays, and assuming that the packets are sent back-to-back without intervening delays, give the total delay from when the first bit is sent at the source until all packets have been completely received at the destination for the following values of *n*:

   (a) $n = 1$,

   (b) $n = 12$.

3. (*4 marks*) Give the maximum data rate, as measured in Kbps, for transmissions on a channel with bandwidth H = 400 KHz and a signal to noise ratio of 63, for each of the following two cases. (Hint: for each case, use whichever of Nyquist's theorem and Shannon's theorem imposes the tightest constraint.)

   (a) Each transmitted symbol has 2 possible values (i.e., transmits just a single bit).

   (b) Each transmitted symbol has 32 possible values.

4. (*4 marks*) Consider a scenario in which 100 sessions share a link of data rate 100 Mbps (i.e., 100 x 10^6 bps). Suppose that each session alternates between idle periods (when no data is being sent) of average duration 1 second, and busy periods of average duration B.

   (a) Supposing that circuit switching is used (i.e., channel capacity is evenly divided among the sessions, with a session's allocated capacity unused when the session is idle), give the achievable data transmission rate a session can achieve during its busy period.

   (b) Give an estimate of the average achievable data rate of each session (during its busy period) when packet switching is used, for B = 10 seconds, B = 1 second, B = 100 milliseconds, and B = 10 milliseconds, assuming:

      - at each point in time the link capacity is shared approximately equally among all of the sessions currently in their busy period; and
      - when a given session is in its busy period, the number of other busy sessions can be approximated by the total number of other sessions (99) times the fraction of time that each is in its busy period.

5. (*8 marks*) Consider a source and destination pair connected by a network path of capacity 100 Mbps and a round-trip time of 100 ms (measured from when the source transmits the first bit of a packet until the destination's ack is received). Assume a packet size of 1000 bytes, and that the loss probability is negligibly small.

(a) Give the maximum achievable data transfer rate in Mbps, assuming use of the *stop-and-wait* reliable data transfer protocol.

(b) Give the maximum achievable data transfer rate in Mbps, assuming use of a *sliding window* reliable data transfer protocol with maximum sending window sizes of 10 and 1000.

(c) Assuming again use of a sliding window protocol, how big would the maximum sending window have to be to get a maximum achievable data transfer rate equal to the network path capacity of 100 Mbps?

(d) Under what circumstances, given these values for capacity, round-trip time, and packet size, could it be beneficial to have a maximum sending window size larger than that given in your answer to part (c)?