CS 422 (Park) **Assignment III** **Due: Feb. 28 (Mon.), 2011**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 4pm on the due date. You may use any number of word processing software (e.g., Framemaker, Word, LaTeX), but the final output must be in pdf format that uses standard fonts (a practical test is to check if the pdf file prints on a CS Department printer). For experiments and programming assignments that involve output to terminal, please use* `script` *to record the output and submit the output file. Use* `gnuplot` *to plot graphs.*

PROBLEM 1 (20 pts)

For students who have the 5th edition (Comer): Read chapters 12, 13, 14 and 15. Solve Problems 12.14, 13.12, 14.8 and 15.4. For students who have the 4th edition: Read chapters 7, 8, 9 and 10. Solve Problems 7.7, 8.7 (talking to an electrical engineer is not needed), 9.4 and 10.4.

PROBLEM 2 (30 pts)

(a) Given a wired or wireless network medium of bandwidth 0–B Hz, Shannon's channel coding theorem gives an upper bound on the achievable reliable throughput assuming the sender has a power budget of P_S and the network medium is subject to white noise of power P_N. For example, if $B = 2$ GHz, then reliable throughput is at most

$$2 \log \left(1 + P_S/P_N\right) \text{ Mbps.}$$

Noting that OFDM allows us to squeeze in a large number of sub-carriers within the 0–2 GHz frequency range over which parallel bit transmissions can be affected—e.g., with 100 sub-carriers we may potentially achieve a 200-fold increase in Mbps—why is Shannon's formula still valid and does not lead to a contradiction?

(b) Ignoring the "1+ " component inside the logarithm of Shannon's formula, give a qualitative explanation of why Shannon's formula should intuitively capture the reliable throughput of a network medium of bandwidth B, signal power P_S and noise power P_N.

PROBLEM 3 (30 pts)

(a) Stop-and-wait's throughput formula, frame size / RTT, holds true when there are no frame errors, and RTT includes the transmission times at the sender to send a data frame and at the receiver to transmit an ACK frame. Suppose stop-and-wait were used to download a file, *bigfile.txt* (around 47 MB), from `www.cs.purdue.edu` using a data frame size of 1500 B and ACK frame size of 40 B from a PC in LWSN B148. Using `ping`, approximate the RTT applicable to the stop-and-wait protocol above. Find out the configured link speed of the Ethernet interface of the PC (`ifconfig -a` will list all interfaces; consult `dmesg` to check link speed) and, assuming the link speed at the web server is the same, estimate the RTT of stop-and-wait using the link speed and link length assumption of 100 feet. How do the `ping` and formula based estimates of stop-and-wait RTT compare? What are the main factors that may contribute to their quantitative difference? As always, please use `script` to log the measurement experiments.

(b) Using a web browser, download `www.cs.purdue.edu/~park/bigfile.txt` and use a watch to time the approximate completion time of the download operation. Compute stop-and-wait file download completion time estimates of *bigfile.txt* using the two RTT estimates from (a). How do the three values compare? What are the main factors that may cause the measured completion time to differ from the RTT based estimates? Rank them by your assessment of their relative importance. Noting that the web browser uses HTTP which, in turn, uses TCP—a sliding window implementation of ARQ that transmits multiple data frames per RTT—can this help narrow the observed performance gaps? Discuss your reasoning.

PROBLEM 4 (60 pts)

As a continuation of Problem 6, Assignment II, modify the server side such the server's child process, before calling `exec`(), sleeps for R seconds which is given as a command-line argument when the server is started. Modify the client

side so that when no response is forthcoming from the server within S seconds after the request has been sent, a duplicate request is retransmitted. This retransmission is repeated every S seconds (in millisecond granularity) until the server's response is received or the number of attempts has exceeded Q. S and Q should be initialized by reading from a configuration file, *client-config.cfg*, when the client is started. For example, S may be specified as 2.755 meaning 2 seconds and 755 milliseconds. To implement request retransmission at client side, use the signal SIGALRM that is set using `ualarm`() with S as part of the argument. The alarm should be set right after transmitting a request. Register a signal handler, `my_retransmit_req`(), using the `sigaction`() system call. `my_retransmit_req`() is a callback function that you are registering with the kernel such that it is invoked when SIGALRM is triggered. Your signal handler should retransmit the request, check that the number of attempts has not exceeded Q, then call `ualarm`() to set a new alarm before returning. When a response is received before the timer expires, then the timer should be killed so that a request is not retransmitted. If Q has been exceeded, the client should output an appropriate error message before exiting. Test your client/server application with $R = 7.5$, $Q = 4$, and $S = 10.2$, 7.6, 7.55, 7.45, 7.4, and 0.515. Use `script` the record the interaction and output.

PROBLEM 5 (30 pts)

As a continuation of Problem 4, what happens with your client if a response from the server arrives while the client is executing `my_retransmit_req`() because the timer has expired? On a single CPU host (note our client and server processes run on the same host), is this even possible? Explain your reasoning. What about the case where, while the client is executing `read`() and is in the midst of receiving the server response (i.e., copying from FIFO to user space buffer), SIGALRM is raised? Is this possible on a single CPU host? Keep in mind that `read`() is referred to as a slow system call since the return from kernel mode may take a while even when it is not blocked (default mode if there is no data in the FIFO to read). You also need to consider atomicity of FIFO operations. Discuss how you may code your client so that correctness under the preceding race condition scenarios is assured.

PROBLEM 6 (50 pts)

Modify the concurrent client/server application of Problem 6, Assignment II, so that the server becomes a file server. That is, the client sends the full pathname of a file, *file-pathname*, starting at the root directory "/" instead of sending a command. If the file does not exist, an error status (the string "*file-pathname* not found") is returned. If a full pathname is not specified, then the requested filename is searched in the current working directory which is part of the server's run-time environment. The server, when transmitting the content of a file, must decide the size of individual `write`() system calls through which segments of the file are written to FIFO until EOF is encountered. Denote the segment size K. The client must do the same for `read`() system calls. For simplicity, let the client's segment size be K. The server and client must agree on a protocol for distinguishing signaling messages from file content when communicating through the client's FIFO. That includes the server indicating when a file transmission is complete. The server should measure the wall clock time elapsed between sending the first segment of a file and its last by calling `getttimeofday`() which is output to *stdout*. The same goes for the client to measure the time between the first and last `read`() system calls. The client should also write the content received to a file using `write`() (same segment size as `read`()). Test the client/server file server application on `/u/u3/park/pub/cs422/medfile.txt` for K values 100, 200, 500, 1000, 2000, 4000, 8000, 16000, 32000 bytes. Log the results using `script`. Plot the server- and client-side completion times as a function of K using `gnuplot`. What behavior do your observe? Is there an optimum K value? Are there any differences between the server- and client-side measurements? Discuss your results. Repeat the test and analysis for `/u/u3/park/pub/cs422/smallfile.txt`.