CS 422 (Park) **Assignment III** **Due: Mar. 2 (Wed.), 2005**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 11:59pm 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 or ps format that uses standard fonts (a practical test is to check if the pdf/ps file prints on a CS Department printer without problem). 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. Use* `ps2gif` *to convert a eps/ps plot to gif format (e.g., for inclusion in Word).*

PROBLEM 1 (25 pts)

Read Chapters 8, 9, 10, and 11 from Comer. Solve Problems 9.3, 9.4, 9.8, 10.4, and 11.12.

PROBLEM 2 (30 pts)

The purpose of this problem is to introduce "sniffing" raw data from the "wire" on the private LAN in the lab consisting of two PCs, two laptops, and Ethernet hub. Execute the following command on a PC to capture Ethernet frames from the network:

% `sudo /usr/local/etc/snoopwrap-elbc1 -c10 -o /var/tmp/`*login-ID*

The command will capture 10 packets from the Ethernet LAN and save them in the file `/var/tmp/`*login-ID*, where *login-ID* is your login ID. Generate your own traffic on the private network by doing telnet to communicate with a machine on the private net (`telnet 10.0.0.X`). Submit the log file in octal format. Using the Ethernet header format(s) discussed in class, decode from the octal dump what the values for the fields in the Ethernet header are. Compare these values across the 10 captured frames. Use `/sbin/ifconfig` to compare the Ethernet addresses that you have snooped with those printed out by `ifconfig`. From the value of the type/length field determine whether the Ethernet frames are DIX- or IEEE 802.3-compliant. How long is the payload of the Ethernet frame? Noting that the first four bits of an IP header indicate its version number (4 or 6), locate the version number bits in the Ethernet payload and identify the IP version running on your test machine. Noting that the last 8 bytes of a 20-byte IP header represent the 4-byte IP source address and 4-byte IP destination address, respectively, locate the IP source/destination address bits in the payload and compare their values to those output by `ifconfig`. Relate the payload content to the telnet session that you've been conducting. (The default IP and TCP header sizes are 20 bytes each.) Save your `telnet` interaction using `script`. You need to do some detective work here. When you type your password during `telnet`, is it visible in the octal dump? (Change your password to a temporary password before doing the experiment so that you can safely keep the original one.)

PROBLEM 3 (40 pts)

As a continuation of Problem 2, Assignment II, modify the client code such that the first parameter in the client request is an integer representing a timeout parameter, *mytimeout* (granularity seconds), followed by the UNIX command to be requested. The client, before sending out the service request, sets a timer using `alarm`() to *mytimeout*. If the response from the server does not arrive before the timer expires (i.e., the SIGALRM signal is raised), the request is retransmitted. Use `sigaction`() to register a SIGALRM signal handler that performs the retransmission asynchronously. Your driver code (i.e., `main`()) should use `pause`() to take a nap before it's woken up by the kernel. After 5 tries it should give up and terminate with the message "no response from server". The client should also take a time stamp just before sending out a request using `gettimeofday`() and just after receiving the server's response as well as just after entering the SIGALRM handler (should this happen). Compute the measured RTT and the measured SIGALRM interval (should a timeout be triggered), and print them out at the client side. There is also a change to the server. Modify the message format so that the timeout value is transmitted:

 # mytimeout # process-ID # command # argument-1 *# argument*-2 *# ⋯ # argument-n*

The server process, upon receiving the message request, hands the timeout value along with the command requested to its child process. The child process, after forking, sleeps for *mytimeout* seconds using `sleep`() before executing the

requested command. Test your program by first running the server application in the background, then executing `client.bin` multiple times with `1 date`, `2 date`, `5 date`, and `10 date` as arguments. Use `script` to record your output. What do you observe and why? How accurate are the measured RTT values? How accurate are the measured SIGALRM timer values?

PROBLEM 4 (30 pts)

(a) Suppose that you are using the stop-and-wait protocol to transmit a file of size S bytes from host A to host B on a point-to-point link where bandwidth is B (bps) and one-way link latency is D (msec). The data frame size is F_{data} (bytes) and the ACK frame size is F_{ack} (bytes). What is the ideal timeout value to be used by the sender A in a perfect world? Assuming the link is noise-free (i.e., no bit flips), what is the throughput (in bps) achieved by stop-and-wait? Explain your derivation. For $F_{\text{data}} = 1538$ (bytes), $F_{\text{ack}} = 84$ (bytes), $D = 1$ (msec), $B = 100$ (Mbps), and $S = 100$ (MB), calculate how long it takes for stop-and-wait to complete file transfer. What is the utilization in this instance? What is the average throughput when a data frame encounters bit flips (and therefore is discarded by B's NIC) with probability ε but ACK frames are immune to bit flips? Calculate the average throughput for the previous example with $\varepsilon = 0.001$. By what percentage has throughput declined? What if $\varepsilon = 0.000001$?

(b) Suppose that instead of stop-and-wait you are running a sliding window version of ARQ where k frames are transmitted as a block in unison, but separate ACK frames are returned acknowledging each data frame. The sender A transmits the next block of k data frames only when it has received the ACK (cumulative positive ACK) of the last frame in the previous block. If it does not within a specified timeout, then the entire block is retransmitted. Using the set-up of part (a), what should be the ideal timeout? What is the throughput expression assuming there are no bit flips? For $k = 2, 6, 12$, and 16 calculate the completion time of file transfer using the example parameters in (a), except $F_{\text{data}} = 1500$ (bytes). Use two machines in the lab to clock the actual file transfer completion time for a dummy 100 MB file (you can create your own file or use the one provided by the TAs) when employing `ftp`. Since `ftp` uses TCP which uses a form of sliding window control, estimate what k value `ftp` (i.e., TCP) is using to achieve its completion time. How does `ftp`'s performance fare against the theoretically best possible completion time of $8 \star S \div 100000000$ (sec)? Is there room for improvement? *Note: After finishing your benchmark test, please delete the dummy 100 MB files at the receiver and sender sides.*