CS 422 (Park) **Assignment V** **Due: Nov. 15 (Tue.), 2011**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 11pm 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.*

PROBLEM 1

Read chapters 20–25.

PROBLEM 2 (120 pts)

Modify the TCP-based remote command client/server application of Problem 3(a), Assignment IV, so that it becomes a file transfer client/server application. That is, a client process running on some IP host A sends a file download request to a file server B running on another IP device. Create your own message format that, at a minimum, contains the file name to be transmitted by the server using TCP. As a simplification, the requested file need only be searched in the current working directory of the server process B. If a requested file cannot be found, the server closes the TCP connection to the client, and the client process terminates with a suitable error message. The server should read the port number that it listens on from a configuration file *server-param.dat* at start-up. The client app should be run with three command-line arguments specifying the requested file, the server's IP address and port number. One twist to this problem is that the server should be written to run on Windows (XP and later). The client remains a Linux app. For server development, you may use any of the Windows machines in CS Windows labs (e.g., LWSN B168) using your CS Windows account, ITaP operated Windows PCs (Purdue account), or your own PC/laptop if equipped with a C compiler. Visual C++ is a popular programming environment but not necessary. For testing, use a command-line window in Windows to execute the server app. Writing (or porting Linux/UNIX) networking programming code for Windows entails using the Windows API to access general kernel services and WinSock for socket API. Consult the TA Notes link for additional information and examples. Test your client/server application using a small, large, and non-existent file. The small and large files will be provided in the TA Notes.

PROBLEM 3 (200 pts)

Design, implement, and benchmark a UDP-based file transfer client/server app for Linux that uses sliding window with negative ACKs to improve upon TCP's sliding window that uses positive ACKs. Performance improvement in the form of faster (i.e., shorter) reliable file transfer completion time should be significant in network environments where packet losses are infrequent and files are large. Since there is no standard solution to sliding window with negative ACK, the first part of the problem entails designing your own protocol and explaining why it works correctly. For this part, describe your idea and design choices, provide a detailed description of the protocol—Venkatesan has to be able to make sense of it without becoming prematurely bald—and justify why it works correctly. If there are situations where correctness may be violated—but in your view acceptable in practice—specify the exceptions. Conclude the first part by arguing why your negative ACK sliding window protocol may be faster than TCP and, perhaps, other negative ACK sliding window protocols. In the second part, implement and benchmark your protocol. If your protocol has parameters that you may want to tune (e.g., window sizes), clearly specify them in separate headers files. Instead of static definitions, they may also be read from parameter files at start-up. Note that the number of relevant parameters that impact performance may be many. However, the payload size should be fixed to 1000 bytes. The file name, server IP address and port number specification should follow a format similar to Problem 2.

To affect controlled, repeatable packet losses during benchmarking, use a wrapper function of the `sendto`() system call, `my_sendto`(), that in addition to the arguments of `sendto`(), takes two additional integer arguments, *total_num* and *loss_num*, that are used to specify packet losses that will be emulated by the server's `my_sendto`() call. For example, if *total_num* $= 1000$ and *loss_num* $= 1$, then `my_sendto`() will drop 1 in every 1000 packets by incrementing a counter modulo 1000 and omit calling `sendto`() when the modulo operating yields 0. Refer to the TA Notes for additional details on implementing `my_sendto`() which plays an important role during benchmarking. Use the large files provided in the TA Notes to test correctness of your protocol and its implementation under various packet loss conditions. They include no loss (i.e., $loss\_num = 0$), 1 loss in 10000, 1 loss in 1000, and 1 loss in 100. Note that these are deterministic losses but the TA will also use probabilistic losses during benchmark evaluation of your protocol by linking with a different implementation of `my_sendto`(). Use `diff` to verify that a file has been correctly transferred. Also, run `scp` with the `time` utility to (roughly) gauge how fast TCP transfers the same file.

To enable uniformity and comparison across different negative ACK sliding window protocols, implement "connection set-up" by having the client send 5 duplicate back-to-back request messages to the server so that (hopefully) one will get through. The request messages should be sent using `sendto`(), not `my_sendto`(). Similarly, the server indicates EOF by sending 5 duplicate back-to-back packets of length 1001 bytes which acts as signaling messaging. Use `gettimeofday`() at the client, once just before sending out the first request packet, and once more when receiving the first 1001-byte EOF signaling packet, to compute the file transfer completion time (in seconds with millisecond accuracy). Print the completion time to *stdout*.

During benchmarking follow the guidelines provided in the TA Notes. To prevent potential performance side effects introduced by NFS (both server and client side), it is important to read/write from the local file system of a Linux PC (`/tmp` is the default directory). Finally, the top 3 performers, as deemed by the TA's evaluation of your design and implementation (subject to correctness) will receive 80, 60, and 40 bonus points.