CS 422 (Park) **Assignment V** **Due: Nov. 15 (Mon.), 1999**

PROBLEM 1 (32 pts)

(a) "Congestion," in road systems, occurs during rush hour when the movement of cars along a stretch of road proceeds at a snail's speed or even comes to a screeching halt. Consider all the major events—a car accident blocking one or more lanes of the road, a number of slow moving cars creating a slowly advancing clog, etc.—that can lead to the traffic being tied down. For each, discuss whether there is a corresponding event in communication network systems that may cause similar effects or consequences. (Recall that the road system analogy of communiation networks only goes so far.)

(b) Continuing with part (a), assume you are a bureacrat in the transportation sector that oversees highway road systems, and all that you care about is how many cars exit on a designated number of exits (e.g., leading to office complexes) per unit time, say, per minute. Let us call this measure of system performance CLPM (cars-leaving-per-minute). Assuming a speed limit of 65mph, does it make sense to speak of a *maximum* CLPM? Since CLPM can only increase when more cars enter into the highway system, let us define a measure called CEPM (cars-entering-per-minute) which measures the rate of influx of new cars into the road system. From your experience and common sense, what would be the typical shape of the graph or function relating CLPM to CEPM, CLPM = f(CEPM), look like? Draw your candidate graph and justify why you think its shape is just so *assuming* CLPM is monotone increasing (or, in general, nondecreasing) in CEPM.

(c) Assume some person comes along claiming that the CLPM/CEPM relation can be "dome shaped" (i.e., unimodal) where CLPM increases as CEPM increases up to some point, after which it decreases. Assuming this person is not suffering from sleep deprivation (e.g., doing assignments for CS4xx) and knows what he/she is talking about, how is this possible? Can the same phenomenon also occur in communication networks? Explain.

(d) CLPM may not be the measure that we—as *users* of the highway system—would like to optimize. Given n users in the system all engaging in commuting, define a user-sensitive cost measure that more accurately reflects a user's decline in comfort level. Your definition must be precise. Can you define a corresponding measure for communication networks? What constitutes the "user" in the communication network set-up in your definition? Assuming your job aspiration is becoming a transportation bureaucrat because you want to increase the welfare of your fellow commuters, what might be a revised criterion that better reflects the performance region that a road system should achieve? From an engineering point-of-view, what can you do to achieve this? Are these actions, in your opinion, "easier" in road systems or in communication networks?

PROBLEM 2 (20 pts)

As a variation of Problem 3, Assignment IV, reimplement the remote execution client/server system using TCP in place of UDP, including the corresponding tests and benchmarks.

PROBLEM 3 (50 pts)

Implement an end-to-end sender/receiver application pair using UDP where the sender process, `udp-send`, on the source host transmits a sequence of UDP packets to the receiver process `udp-receive` on the destination host. `udp-send` takes two command-line arguments

% `udp-send` *interarrival-time session-duration*

where *interarrival-time* is the time interval between successive packets (measured in milliseconds) and *session-duration* is the total session duration or lifetime (i.e., after *session-duration* seconds from start-up `udp-send` is to terminate), measured in seconds. The receiver, `udp-receive`, takes a single command-line argument, *interservice-time*, which is the time duration (or "idle time") between successive servicing of two packets.

The receiver works by responding to two signals—SIGPOLL and SIGALRM—in an asynchronous manner using signal handlers. When a SIGPOLL interrupt occurs, the receiver's SIGPOLL handler fetches the newly arrived UDP packet(s) from the kernel's buffer space and enqueues the packet into its own user space buffer, implemented as a first-come-first-served (FCFS) queue. For tracing purposes, the SIGPOLL handler also takes a time stamp using `gettimeofday()` and records both the time stamp and the current queue length—excluding the newly arrived packet—into a LOG file called *log-queue* for later examination. If the user space buffer, at the time of packet arrival, is full, then the UDP packet is dropped or discarded. This event is then also recorded using a time stamp into a separate LOG file for packet drops called *log-drops*. When a SIGALRM interrupt occurs (signalling that a packet should be serviced), the receiver's SIGALRM handler dequeues a UDP packet from its FCFS queue. Before exiting the handler, the interservice-time parameter is used to set when to wake up next.

Test your application using the following set-up. Use a fixed MTU (for UDP packet payload) size of 1kB (containing an all zero bit pattern) and a user space buffer capacity of 500 MTUs (i.e., 500kB). For interservice-time 200ms, run six experiments with interarrival-times 350ms, 300ms, 250ms, 200ms, 150ms, 100ms for a fixed session duration of 30s. For each run, plot two time series (or graphs) using the log files *log-queue* and *log-drops*, one showing the queue length as a function of time and the other showing packet drops as a function of time. Use your favorite graph plotting software (e.g., `gnuplot`). Give your interpretation of the queueing behavior and packet drop behavior as the interarrival time is increased.

Note: An important programming consideration here is the management of the two interrupts such that they do not get in "each other's way." For example, while the SIGALRM signal handler is active, the SIGPOLL signal should not be able to interrupt the SIGALRM signal handler—and vice versa—as this can lead to a corrupted state of the user space buffer which is manipulated by both. As part of the hand-in, explain in half a page of write-up how your program manages the orderly access to the user space FIFO queue by the two signal handlers.

PROBLEM 4 (50 pts)

Implement a "greedy" file transport protocol called `ftp-greedy` which achieves fast (or not-so-fast), reliable file transport on top of UDP using a form of selective ARQ. Given a file F of size $|F|$ bytes to be transferred, `ftp-greedy` segments F into $\lceil |F|/M \rceil$ blocks of size $M < |F|$ (except for possibly the last block which can be of size less than M), sending each block encapsulated in an UDP packet. Before encapulation, each block is assigned a header where a sequence number i identifying the block is enscribed. The sequence number is represented using a single byte, thus giving a range of 0–255. If more than 255 blocks are needed, $i \bmod 255$ is used to achieve wrap-around. `ftp-greedy`, on the sender side, is to be invoked using

% `ftp-greedy` *filename IP-address*

where *filename* is the name of the file to be transferred and *IP-address* is the IP address in dotted decimal form. On the receiver side, `ftp-greedy` is executed without command-line arguments. This puts `ftp-greedy` in "receive mode." When the transfer of a file is complete, `ftp-greedy` (on the receiver side) prints out a message to *stdout* saying that the file in question has been (reliably) received after which it terminates. For the receiver to know the filename in which to store the received file, the sender, in block 0, sends the filename. To indicate that no more blocks are to come (i.e., the last block i has been sent), the sender sends an empty block with sequence number $i + 1 \bmod 255$ at the very end.

Two parameters (e.g., hardcoded using #define) influence the sender's behavior, M and S. M is the block size and S is the sender's interpacket waiting time (in msec). That is, S measures the number of milliseconds the sender waits between successive sending of packets (an event triggered by SIGALRM). The smaller S, the more "greedier" `ftp-greedy`. The receiver uses negative ACKs to achieve reliable communication using retransmission. As soon as the receiver sees a "jump," i.e., a hole in its received packet sequence,

$$\cdots i-2, i-1, i, i+2$$

it transmits a NACK packet to the sender demanding that packet $i+1$ be retransmitted. At the same time, it starts a timer to expire after T msec. If the missing packet is received prior to the timer expiration, the timer is cancelled. If not, a NACK is sent again. Set T initially to 3 msec; each time a NACK is sent again because of timer expiration, increment T by 2 msec. Each time a retransmitted packet arrives before timer expiration, decrement T by 1 msec. T should never be allowed to go below 3 msec or above 15 msec. The receiver can maintain a buffer size of K packets to hold packets arriving while the missing $i+1$ packet is being processed. You can maintain the buffer anyway you want, but it can never exceed the K packet bound.

Test your application by creating a $|F| = 100$ KB dummy file containing the ASCII characters '0', '1', '2', ..., '9', '0', '1', ... Measure the file transmission completion time (wall time) at the receiver from the moment that the first packet is received to the moment that the last packet is received. The last packet may be marked by sending a termination packet with empty payload. Perform the measurements for $S = 10, 30, 50, 70, 100, 200$ msec with $M = 1000$ B. Draw the performance curve (completion time) as a function of S. For the S corresponding to the smallest completion time, call it S^*, test your application for $M = 500, 1500, 2500$ B. Draw the performance curve as a function of M. Give a 1/2-page written interpretation of your results. Hand in the source code, scripts, and your summarized results.