CS 536 (Park) **Assignment VI** **Due: Nov. 23 (Tue.), 2004**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 5pm 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

Read Sections 5.1, 5.2, 5.4, and 6.3 from P & D.

PROBLEM 2 (10 pts)

Read "Congestion avoidance and control" by Van Jacobson. In *Proc. ACM SIGCOMM '88*, pp. 314–329, 1988. The paper (a version thereof) can be found at

`http://www-nrg.ee.lbl.gov/papers/congavoid.pdf`

Give a one-page summary and critique.

PROBLEM 3 (15 pts: for the plan, not the project)

Write a 3-page project plan as follows: (i) any relevant background, (ii) what you aim to do, (iii) why you find it interesting and worthwhile, (iv) specific output—in DARPA jargon, "deliverable"—that you will produce, (v) the specific steps you plan to take to achieve the target outputs, and (vi) what you expect to find (your clear/fuzzy gut instinct although you may be proved wrong). Your project has the following components: idea, implementation, benchmarking, measurement, and interpretation of the results. These need to be written up, in a professional manner, in a 15–20 page report (11 pt font, A4 letter) including plots and (any) references. The project plan is due on Nov. 15 (Mon.), 2004, in hardcopy, submitted at the beginning of class. I will check it and return my comments Nov. 17, in class. It will either be a pass or revision required. What I will look for is the soundness of the plan, its scope and feasibility. Scope is of paramount importance: you do not want to propose something overly grandious that cannot be feasibly tackled in the given time frame (project is due Dec. 18, 2004). View it as an assignment, albeit solving a single "heavy-weight" problem in place of 3–4 smaller problems that a typical assignment contains. Project ideas will be posted on the course web page. Your project need not be confined to this list. You may form a group of up to 3 people. The pros are: divide-and-conquer, share in the fruits of the joint effort. The cons are: coordination overhead, share in the fruits of the joint effort. (In a group effort, each group member gets the same project grade.)

PROBLEM 4 (40 pts)

As a continuation of Problem 4, Assignment IV, extend `cbr_send_v2`, call it `cbr_send_v3`, such that it uses the first 4 bytes of the payload as a sequence number field to keep track of sent packets. Modify the receiver `cbr_recv` so that it records the sequence number—along with the other information logged—in main memory that is then dumped to disk at the end of the run. The receiver should also count, in real-time, the number of dropped packets during the specified *time-window* argument and output to *stdout* both the number of dropped packets and loss rate (dropped packets divided by the number of packets sent during the time window). Lastly, add a termination check to the receiver so that upon receiving a packet with the sequence number (unsigned integer) all 1s it goes into termination mode as with SIGINT. The sender, after transmitting *packet-count* number of packets, sends 10 of these special termination packets using a 100 msec spacing, to signal the receiver that it should terminate. The SIGINT based termination mode remains so that manual intervention with graceful shutdown can be affected.

Benchmark the CBR sender/receiver application on two Xinu machines with *payload-size* 1 KB, *packet-count* 100000 (you need to increase the initial array size allocated), *packet-spacing* 10 msec, and *burst-size* 2. Using `gnuplot`

draw the time series plots in Problem 4, Assignment IV, under aggregation time window 500 msec that includes the packet loss plot. Benchmark the application by porting the Linux code to your Linux PC at home/dorm/undisclosed location—only if you have cable/DSL broadband connection (or faster)—and repeat the experiment by sending from the Linux PC to one of the Xinu machines. Perform the benchmark experiment in reverse direction by sending from a Xinu machine to your Linux PC. If you have a Linux PC but do not have broadband connection (but have free local calling telephone service), then use your 56 Kbps modem for the experiment with *packet-spacing* 100 msec and *payload-size* 200 B. In this case, for comparative purposes, you need to additionally run a corresponding benchmark test between two Xinu machines at this spec. If you don't have a Linux PC at your residence or you don't have free local calling, then use one of Purdue's public Linux/UNIX accounts to carry out the "remote" benchmark experiment. Using your measurement data and plots, discuss your findings.

PROBLEM 5 (60 pts)

Design, implement and benchmark a UDP-based peer-to-peer (P2P) pseudo real-time audio streaming application. Your application can be built on top of the CBR sender/receiver application, with suitable modifications. The sender, `my_audio_send`, takes as command-line arguments

% `my_audio_send` *IP-address port-number audio-file packet-size packet-spacing mode*

where *audio-file* is a stored audio file that will be streamed to the receiver (i.e., client)—unless otherwise indicated, assume the file format is binary—*packet-size* (in bytes) is the size of the UDP payload (excluding 4-byte sequence number) at which unit the audio file will be segmented and transported, *packet-spacing* (msec) is the initial packet spacing used in the CBR transmission of the audio file, and *mode* specifies the congestion control mode: 0 (method A), 1 (method B), 2 (method C), and 3 (method D). The receiver, `my_audio_play`, has command-line arguments

% `my_audio_play` *port-number time-window log-file packet-size pb-del pb-sp buf-sz target-buf*

where *pb-del* is the initial playback delay (sec)—time delay from the arrival of the first audio packet—*pb-sp* (msec) is the time interval at which buffered audio is written to `/dev/audio` for actual playback (triggered by SIGALRM), *buf-sz* is the total allocated buffer space (bytes), and *target-buf* (bytes) is the target buffer level (i.e., Q^*). In the receiver's code structure, attention needs to be paid to the shared audio buffer—the SIGIO handler will write to the buffer whereas the SIGALRM handler for audio playback will read from the buffer—so that it does not get corrupted due to unruly access (e.g., semaphores may be used to achieve orderly access). Also, when audio packets, upon arriving, find the audio buffer full, they will be dropped. In addition to the information logged in Problem 4 of this assignment and Problem 4 of Assignment IV, log the current buffer occupancy at the time of new packet arrival and buffered audio reads (in units of *packet-size*) as well as any buffer overflow events. To affect feedback congestion control, the receiver transmits a 12 byte feedback packet containing Q^* (i.e., *target-buf*), $Q(t)$, and γ (in terms of time interval *pb-sp*, not rate). Depending on the *mode* value, the sender will utilize the relevant information to institute the selected congestion control.

Benchmark the application between two Xinu machines where *audio-file* will be provided (see TA notes), packet size is 1 KB, initial *packet-spacing* at the sender side is 100 msec, *pb-del* is 5 seconds, *pb-sp* is 50 msec, *buf-sz* is 75 KB, and *target-buf* is 50 KB. Perform one benchmark run per congestion control method. Plot the time series results, including those of queue length evolution (i.e., $Q(t)$) and buffer overflow, and discuss your findings. Note that there is some degree of freedom left in the selection of the congestion control parameters. Run the receiver on designated Xinu machines (see TA notes) where audio has been enabled (by default, all Xinu machines have their audio drive disabled to prevent a 70s disco atmosphere in the Xinu lab).

PROBLEM 6 (30 pts)

Design a "greedy" variant of TCP congestion control, call it *TCP-greedy*, that aims to exploit the cooperative nature of other TCP flows sharing a bottleneck link to monopolize bandwidth. The basic idea is simple: assuming there are other flows competing for shared bandwidth, *TCP-greedy*, upon detecting possible packet loss, will institute a congestion control action (could include backoff) that increases its bandwidth share by exploiting the fact that other TCP flows will back off. The subtlety of the problem is: when other TCP flows have already backed off as much as is possible (crawling along), or TCP-greedy is the only flow traversing the link, then continued aggressiveness will lead to self-congestion, i.e., shooting oneself in the foot. The trick is to know when to be greedy, and when not to be (that is not to say that TCP-greedy is condoned). Describe a detailed design of your TCP-greedy—to the extent that it may be straightforwardly implemented by anyone familiar with TCP reading your document—and argue why you think that your version will achieve its objective.