CS 536 (Park) **Assignment IV** **Due: Nov. 2 (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 3.1, 3.2, 3.4, and 4.1 from P & D.

PROBLEM 2 (20 pts)

Read "Measured capacity of an Ethernet: Myths and Reality" by D. Boggs, J. Mogul, and C. Kent. *Proc. ACM SIGCOMM '88*, pages 222-234, 1988. The paper (technical report version) can found at

`http://www.research.compaq.com/wrl/techreports/abstracts/88.4.html`

Give a half-page summary of its pertinent points. Give a half-page critique of its conclusions, methodology, and assumptions used to obtain the results. Do the issues discussed in the paper also apply to IEEE 802.11 WLANs? Provide a half-page discussion of your thoughts and reasoning.

PROBLEM 3 (40 pts)

As a continuation of Problem 4, Assignment III, modify the concurrent client/server application so that TCP is used in place of UDP. The retransmission mechanism remains in place. Since TCP performs ARQ, we have an ARQ-over-ARQ protocol at hand. Perform the same test as in Assignment III with the following additional constraints: (1) in your server code, make sure that the TCP implementation uses the same random seed, and hence is subject to the same sequence of request drop events, as UDP; (2) use `time` when executing the client so that you can determine the client's completion time (focus on the wall time). Re-run your UDP-based application with `time`. Compare the completion time results of the two runs and discuss your findings. As before, the interaction should be logged using `script` and the output submitted.

PROBLEM 4 ($80 = 50 + 30$ pts)

In the following, two versions of UDP-based traffic generators but a single version of receiver will be implemented and benchmarked.

(a) In the first version, implement a UDP-based CBR sender/receiver application where the sender, `cbr_send_v1`, transmits at maximum possible rate. The sender takes the command-line arguments

% `cbr_send_v1` *IP-address port-number payload-size packet-count*

where *IP-address* and *port-number* are the coordinates of the receiver, *payload-size* is the size of the UDP payload in bytes, and *packet-count* is the total number of packets to be transmitted. `cbr_send_v1` is essentially a `for`-loop with loop count *packet-count* where no artificial pauses are injected between successive calls to `sendto`(). The sender takes a time stamp using `gettimeofday`() before entering the `for`-loop and after exiting it, then prints out the completion time (i.e., the difference between the two time stamps) in seconds, the application bit rate (bps) which is *packet-count* $\times$ *payload-size* $\times$ 8 $\div$ completion time, the physical bit rate which includes the UDP header (8 bytes) and Ethernet header overhead, and the packet-per-second (pps) values of the application and physical data rates.

The receiver, `cbr_recv`, blocks on `recvfrom`() in an infinite loop. It is manually terminated at the end of a run when the sender has transmitted the last packet. For each arriving packet, it takes a time stamp and increments a

cumulative packet counter and a windowed packet counter. It logs the time stamp, cumulative packet counter, and payload size into main memory (arrays of 50000 entries should be allocated at initialization for this purpose; you can assume that no more than 50000 packets will be sent, i.e., unless the sender code is buggy). It is important to use main memory for logging so as to avoid disk I/O. The time stamp values should be relative to the first time stamp taken when the first packet arrives. The main memory log is flushed at the end of the run to disk. To achieve this, register a SIGINT signal handler—the signal generated by typing CTRL-C—that performs the graceful clean-up before termination. The receiver takes three command-line arguments

% `cbr_recv` *port-number time-window log-file*

where *port-number* is the port number on which it waits for packets, *time-window* is a time window in msec, and *log-file* is the name of the log file to which the measurements will be flushed. The receiver uses *time-window* to calculate "instantaneous" data rates in real-time. It accomplishes this task by registering a signal handler for SIGALRM that is invoked every *time-window* msec via `ualarm`(), upon which it calculates the instantaneous (i.e., time windowed) data rate—application pps—by dividing the time windowed counter by *time-window*. This value is then output to *stdout* (i.e., terminal) for real-time monitoring. Before terminating, the receiver outputs the overall application pps as in the sender.

Perform four benchmark tests with *packet-count* 50000, *time-window* 500msec, and payload sizes 600, 1200, 1800, and 2400 bytes. Use `script` to log the terminal interaction. Using `gnuplot`, show, in a single plot with two graphs, the application pps data rates calculated at the sender and receiver as a function of payload size (the x-axis is the payload size and the y-axis the data rate in pps). Create a corresponding plot for the sender and receiver physical data rate in bps. What trend do you observe? Discuss your findings. For the payload size 1200 run, process the log file by aggregating the packet counts over 500msec time windows. That is, the new processed data file looks like

0.0 1000
0.5 1002
1.0 999
1.5 1001
...

where "0.0 1000" would indicate that there were a total of 1000 packets logged during the first 500msec time window, "0.5 1002" would mean that there were 1002 new packets during the second 500msec time window, and so forth. Make the aggregation time window a command-line parameter of your data processing script. Using `gnuplot`, show the time series plot of the processed log file that depicts the overall time dynamics. Do the logged time dynamics plots agree with the overall data rates shown in the first plot? Are there noticeable fluctuations over time? How do the logged values compare with the real-time computed time windowed values? Discuss your findings. Re-run the 1200 payload case with *time-window* 1000 msec. How does the larger time window impact the real-time instantaneous data rates?

(b) A rate-limited variation of the CBR generator in part (a), implement a new sender, `cbr_send_v2`, that transmits paced CBR traffic specified by the user. The sender takes two additional command-line arguments

% `cbr_send_v1` *IP-address port-number payload-size packet-count packet-spacing burst-size*

where *packet-spacing* (in msec) denotes the time interval between successive packet bursts and *burst-size* denotes the number of packets transmitted "simultaneously"—i.e., back-to-back without wait times—at each packet burst. Thus burst events occur periodically *packet-spacing* apart, and at each burst event, *burst-size* number of packets are transmitted in bulk. Perform benchmark tests with *packet-count* 50000, *time-window* 1000 msec, *payload-size* 1200 bytes, *burst-size* 1, and *packet-spacing* values 50, 30, 10, and 1 msec. Analogous to the first plot in part (a)—but with *packet-spacing* in place of *payload-size* on the x-axis—show the application data rates at the sender and receiver sides. What trend do you observe as *packet-spacing* decreases from 50 downto 1 msec? Explicate your findings. Repeat the benchmarks with *burst-size* 10 and 20. How are your results different? Discuss your findings. For the *burst-size* 20 and *packet-spacing* 10 msec run, perform the log file aggregation with time window 500 msec as in (a). Plot the time series. Do you observe fluctuations? What about aggregation time window 5 msec?