CS 536 Fall 2019

Lab 6: Congestion Control for Audio Streaming [270 pts]

Due: 12/02/2019 (Tue), 11:59 PM

Objective

The aim of this lab is to implement feedback congestion control for pseudo-real-time multimedia streaming. To not get sidetracked by video/audio encoding standards and issues, we will use audio streaming which allows us to focus on the networking components. The same methods apply to video streaming.

Reading

Read chapters 7, 8 and 9 from Peterson & Davie.

Problem (270 pts)

Client/server session initiation

Design, implement and benchmark a UDP-based pseudo real-time audio streaming application. In the session initiation phase, TCP is used to exchange session set-up information. In the audio streaming phase, UDP is used to stream feedback controlled audio data. The sender (i.e., server) transmits packets containing audio payload at rate lambda. The rate may change over time when congestion control actions are undertaken upon receiving feedback control packets from the receiver (i.e., client). The client uses a TCP connection at a server port (tcp-port) to transmit a filename specifying an audio file of type AU with extension ".au". For testing, we will use one of two audio files available at

/homes/park/pub/pp.au [short]
/homes/park/pub/kline-jarrett.au [long]

Copy the files to the /tmp file system as in Problem 1, lab3, on the server side so that NFS related performance effects are removed. The audio streaming server, a concurrent server, verifies that the file exists, binds to an unused port number (server-udp-port), and responds by sending the character '2' followed server-udp-port followed by the size of the file (assume 4-byte unsigned integer) to the client. server-udp-port is used by the client to send feedback control messages. Otherwise character '0' is returned. The client, upon receiving '2', binds to an unused port number (client-udp-port) and sends it to the server. client-udp-port is used by the server to transmit audio packets. Upon receiving any other character, the client prints an error message to stdout and terminates. The server, upon receiving a port number from the client, moves on to transmitting the content of the requested audio file using feedback congestion control. When audio data streaming is completed, the server transmits character '5' five times using TCP, dumps the log data in memory to logfile1 (in the current directory) and terminates. The client terminates when it receives character '5'. Before termination, the client dumps log data in memory to logfile2 and outputs to stdout the number of bytes received divided by the size of the file. If the client does not receive '5' on the TCP control channel, it waits indefinitely and lets the user decide what to do.

Streaming server structure

The server process blocks on accept() and forks a child process that handles TCP interaction with the client and UDP audio streaming using ports numbers communicated during the TCP-based set-up phase. The parent process goes back to waiting for further client requests. The server app, streamerd, takes command-line arguments

% streamerd tcp-port payload-size init-lambda mode logfile1

where tcp-port specifies the TCP port number on which the server listens for client requests. The role of the other arguments are discussed below. The child/worker process schedules transmission of audio packets using nanosleep(). Between successive transmission of audio packets, nanosleep() is called to pace transmission using sleep parameter 1/lambda where lambda is the time-varying sending rate (pps). Note that since lambda is in unit of pps (packet-per-second), 1/lambda specifies the time spacing between successive audio packets. Although the problem description will use rate lambda as the control variable so as to be consistent with our discussion in class, when implementing sender-side control at the server lambda must be translated to a time interval before calling nanosleep(). Hence increasing lambda leads to a decrease of sleep interval 1/lambda, and vice versa. The server uses a SIGIO/SIGPOLL handler to perform asynchronous receipt and processing of feedback control packets from the client.

The second command-line argument payload-size (in bytes) specifies the size of the UDP payload, excluding a 4-byte sequence number inscribed at the start of the payload. payload-size is the unit by which an audio file will be segmented and transmitted to the client as UDP payload. Considering that our lab machines are connected via Ethernet switches, we will impose an upper bound on payload-size as 1488 bytes. The third argument, initial-lambda (pps), is the initial audio transmission rate lambda. The fourth agument, mode, specifies the four congestion control modes: 0 (method A), 1 (method B), 2 (method C), and 3 (method D). logfile1 is the filename of a file to record time-varying lambda at the server. At start-up, streamerd reads feedback control parameters a, delta, epsilon, and beta (all of type float) from a data file control-param.dat that are relevant dependent on which congestion control mode is used. The server is typically coded as a multithreaded application, 2 or 3 threads. Use fork() to create a child process that mainly handles UDP audio packets (also referred to as data plane) and the parent who uses TCP to handle management packets (referred to as control plane).

Streaming client structure

The client, playaudio, receives incoming audio packets on client-udp-port and writes them to the app's user space audio buffer after stripping off 4-byte sequence numbers. Use nanosleep() to sleep at fixed playback interval mu which is the inverse of the playback rate gamma discussed in class. mu determines at which time an audio sample is taken out from the app's audio buffer and played back by writing to the client's codec and audio device. By using audio data in .au format that the Linux device driver of /dev/audio accepts, we are able to by-pass audio encoding/decoding issues that are tangential to the network protocol tasks at hand. We may say that SIGIO/SIGPOLL drives the producer side of the audio buffer in the client whereas nanosleep() drives the consumer side. Using semaphores protect the shared producer/consumer audio buffer so that it is not corrupted by concurrent access. Describe your method in Lab6Answers.pdf.

The client process, playaudio, uses command-line arguments

% playaudio tcp-ip tcp-port audiofile block-size gamma buf-size target-buf logfile2

where tcp-ip is the server's IP address, tcp-port is the server's TCP port number, audiofile is the name of an audio file, block-size (in bytes) is the unit by which content is read from the audio buffer and sent to the client's audio device, gamma (blocks per second) is the fixed playback rate at which buffered audio is written to the client's audio device, buf-size is the maximum audio buffer space (bytes), target-buf (bytes) is the target buffer level (i.e., Q*), and logfile2 is the name of a file where time-varying audio buffer state at the client is monitored to evaluate app performance. The client commences playback after prefetching has reached Q*.

The client, upon receiving an audio data packet, sends a feedback packet to the server at port server-udp-port indicating its buffer state. The UDP payload of the feedback packet contains a 4-byte integer specifying buffer occupancy (i.e., the number of bytes excluding the newly arrived audio data in the app's audio buffer) followed by a 4-byte integer specifying target-buf, followed by a 4-byte integer specifying gamma. As in the server, code the client as a 2-process app whose tasks are separated by data vs. control plane chores.

Performance measurement and evaluation

To evaluate how well the system performs, the sender logs the current sending rate lambda, along with the time stamp from gettimeofday(), whenever a packet is transmitted. The receiver, upon receiving an audio packet from the sender or when dequeueing an audio packet for playback, logs the current time stamp and buffer occupancy for off-line diagnosis. Measurement logs should be written to memory and flushed to disk at the end of the run. This avoids overhead and slow-down stemming from disk I/O to update logfile1 and logfile2 during streaming operation.

Benchmark the application between two lab machines using the two audio files provided. Set payload size to 1488 B, init-lambda to correspond to initial packet-spacing of 90 msec, gamma such that 1/gamma is approximately 313 msec, buf-size to 48 KB, and target-buf to 24 KB. Benchmark each of the four congestion control methods. Plot the time series measurement logs using gnuplot, MatLab, or Mathematica. gnuplot is very easy to use and produces professional quality graphical output in various formats for inclusion in documents. At the sender plot lambda in unit of packet rate (pps) as a function of time at granularity 1 sec. At the receiver, plot audio buffer occupancy against time at 1 sec time granularity. There is significant degree of freedom in the selection of congestion control parameters which is up to you to determine. Compare subject audio quality perception with the numerical performance results. Discuss your results and findings in Lab6Answers.pdf. Extend your benchmark by testing with 3 concurrent clients. Check if performance is impacted by additional load at the server. Submit your code and Lab6Answers.pdf in v1/.

Note: This problem may be tackled as a group effort involving up to 3 people. If going the group effort route, please specify in Lab6Answers.pdf on its title page who did what work including programming the various client/server components, performance evaluation, and write-up. Before turnin, send an email to the TAs specifying who the group members are and who will perform turnin. Whether you implement lab6 as an individual effort or make it a group effort is up to you. Keep in mind trade-offs: group effort incurs coordination overhead which can slow down execution, especially for a 2-week lab assignment. Benefits include collaborative problem solving and some parallel speed-up if efficiently executed. Regarding late days, for a group to use k (= 1, 2, 3) late days, every member of the group must have k late days left. For lab6 whose scope is a typical 2-week assignment, it is encouraged to complete it before the Thanksgiving Break.

Bonus Problem (30 pts)

Add a new control law, method E, given by mode value 4. The aim is improve upon the performance of method D. Describe your control law and its rationale in Lab6Answers.pdf. Try to be creative, keeping in mind that intuition which we continually develop and rely on, can be both friend and foe. Implement the control and show performance results that demonstrate improvement over method D. Submit the revised code in v2/. If the main problem is tackled as a group effort, the bonus problem must be solved as a group effort as well.

Turn-in Instructions

Electronic turn-in instructions:

We will use turnin to manage lab assignment submissions. Go to the parent directory of the directory lab6/ where you deposited the submissions and type the command

turnin -v -c cs536 -p lab6 lab6

Back to the CS 536 web page