Real-Time and Embedded Systems: Programming Assignment

Dr. Colin Perkins

20th February 2008

Introduction

Many companies are now marketing voice-over-IP (VoIP) phones. These look like a traditional telephone, but connect to an Ethernet or Wi-Fi network instead of a phone line. They make calls to other VoIP terminals using RTP over UDP/IP as a network protocol (discussed in lecture 16), or can use a central gateway server to make calls to the traditional circuit switched telephone network.

Internally, these VoIP phones run a cut-down version of a general-purpose operating system, such as Linux, on a low speed processor with limited memory. Software is loaded from flash memory, and the phone boots directly into the telephony application after configuring its network interfaces using DHCP. Calls are setup using the Session Initiation Protocol (SIP), then the phone sends and receives real-time speech data. The G.729 speech compression algorithm is often used, and produces an 80 bit frame of compressed speech every 10 ms. On a typical system, running the speech compression algorithm can take up to 30% of the system processor power, while decompression takes 20%. The speech data is transmitted in RTP packets sent over UDP/IP every 20 ms. These RTP packets comprise two frames of speech data and an RTP header, for a total of 32 bytes (a data rate of 12.5 kilobits per second). In addition to the RTP packets containing the speech data, reception quality reports are sent every 5 seconds, to allow the sender to adapt the transmission to match network conditions. Received speech packets are decompressed and played to the user as they arrive. The system also runs a graphical user interface.

More advanced systems support video conferencing, in addition to voice. These systems work in a conceptually similar manner to VoIP phones, and generate an RTP video stream in addition to the audio stream. The video stream is often much higher data rate than the audio, and can generate hundreds, possibly thousands, of RTP packets per second depending on the video quality desired.

Real-Time Behaviour of Linux

Linux has a history of problems where, in some circumstances, jobs may be delayed for tens of milliseconds due to the operating system blocking for access to hardware devices. This can potentially disrupt the operation of video conferencing applications, since the inter-packet spacing is of the same order as the reported blocking times. The aim of this assignment is to test the real-time behaviour of Linux, to see if these concerns are justified.

Write a test program, `sender`, to simulate a video conferencing system sending data. When started with the IP address of another host as a command line parameter, this program should perform the following tasks:

- Every 20ms, `sender` should send a datagram packet to UDP port 5004 of the host specified on the command line. These packets must contain 32 bytes of data, to simulate audio data. The packets should include a sequence number that increases by one with each packet sent, and a timestamp indicating the time at which the packet was sent. For the purposes of this assignment, the values of the other data bytes are unimportant.
- Every 40ms, `sender` should generate a frame of video data. Video codecs send periodic key frames, once per second (every 25th frame) for this exercise, with smaller intermediate frames derived from the contents

of the previous key frame. The `sender` should generate key frames that are 10 UDP packets in size, and intermediate frames that are two packets in size. The UDP packets representing the video data must be sent to port 5006, and must be evenly spaced over the 40ms frame interval, to avoid congesting the network. Each packet is 1500 bytes in size, and should contain a sequence number and timestamp, similar to the audio packets. For the purpose of this assignment, the contents of the other bytes are unimportant.

- The `sender` will listen for packets on UDP ports 5005 and 5007. When a packet, which will have the format described below, is received on either of these ports it should should retrieve and display the statistic on the fraction of packets lost.

Write another program, `listener`, that listens to the packets being sent by the `sender` program and generates reception quality reports:

- For each packet it receives on UDP port 5004, the `listener` program should record the value of the sequence number and timestamp contained in that packet, and also the arrival time of the packet. The sequence numbers should be used to calculate the number of audio packets lost in transit. The transmission and reception timestamps should be stored in a file for later analysis.
- Every five seconds, the `listener` should send a datagram packet (a "Reception Quality Report") back to UDP port 5005 of the host running the `sender` program (the address of the sender is specified on the command line). This datagram should be 70 bytes in length, and should report the fraction of the audio packets sent on port 5004 lost since the last report was sent.
- The `listener` program should also listen on port 5006, and send reports to port 5007, to report the reception quality of the video stream, in an analogous manner.

The two test programs should be written in C using the pthreads and sockets APIs, and should run on Linux. The code you develop does not need to have a "pretty" user interface. You should submit a printout of the source code to the two programs, along with any design notes you have made.

Run the `sender` and `listener` programs, sending data from one host in the lab to another host in the lab. Using the data captured by the listener program, plot timing graphs (similar to that shown in lecture 15) of send time versus arrival time, over a period of approximately one minute, for both audio and video. Repeat on both a lightly loaded host and on a heavily loaded host (for example, when the receiver machine is idle, and when it is compiling a large program). You should use a program such as gnuplot or Microsoft Excel to plot the graphs, producing plots to illustrate any interesting aspects of the timing behaviour of the system.

Discuss the data presented in your two timing graphs. Do the systems you tested have sufficiently accurate timing to be suitable for use in a video conferencing system? How often does your system miss its deadlines? You may need to zoom in on a small part of the graph, or instrument your code to plot additional statistics, to observe the detailed timing properties.

If packets were to be sent at a higher rate (smaller inter-packet spacing) would these systems still be usable? For example, consider the behaviour of a system sending high quality video, where both the key frames and the intermediate frames comprise more packets. Experiment with the system to determine what is the minimum inter-packet spacing the hosts can support while maintaining reasonable timing behaviour. Document your experiments, and experimental design process, and discuss how you might improve the performance of the system.

Submission

This assignment is worth 15% of the mark for this module: 5% for the code, and 10% for design notes, your experimental design and results, and discussion of system performance issues. Completed assignments must be submitted by 5:00pm on Friday, 14th March 2008 via the locked box outside the Teaching Office. You must include a pink declaration of authorship form. Late submissions will be awarded zero marks unless accompanied by a valid special circumstances form.