Real-Time and Embedded Systems: Programming Assignment

Dr. Colin Perkins

1st March 2005

Background

Many companies are now marketing voice-over-IP phones. These look like a traditional desk telephone, but connect to an Ethernet LAN instead of a phone line. They can make calls to other voice-over-IP devices using RTP over UDP/IP as a network protocol, or can use a central gateway server to make calls to the traditional circuit switched telephone network.

Internally these phones run a cut-down version of a general-purpose operating system, such as Linux, on a low power processor with limited memory. Software is loaded from flash memory, and the phone boots directly into the telephony application after configuring its network interface using DHCP. When a call is made, the phone will perform an initial negotiation stage using a protocol such as SIP or H.323, then start transmitting speech data. The G.729 speech compression algorithm is often used, and produces acceptable quality speech at a data rate of 8kbps. Running the compression can take a significant portion of the system processor power. Speech packets are produced every 10ms, with reception quality reports sent every 5 seconds $\pm$ 2.5 seconds on average.

Unfortunately, Linux has a history of problems due to "system call latency" where, in some circumstances, tasks may be delayed for tens of milliseconds when the operating system blocks during access to hardware devices. It is possible that this may disrupt the operation of a voice over IP system if implemented using Linux, since the inter-packet spacing is of the same order as the reported system call latency. The aim in this assignment is to test the real-time behaviour of Linux, to see if these concerns are justified.

Assignment

Write a test program, `sender`, to simulate a voice-over-IP sender. When started with the IP address of another host as a command line parameter, this program will perform two functions:

- Every 10ms, it should send a datagram packet to UDP port 5004 of the host specified on the command line. These packets must contain 22 bytes of data, to simulate a voice-over-IP system using the G.729 speech compression algorithm. The contents of the 22 bytes 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 test, the values of the other data bytes are unimportant.
- It will listen for packets on UDP port 5005. When it receives a packet on this port, which will have the format described below, it should should retrieve and display the statistic on the fraction of packets lost.

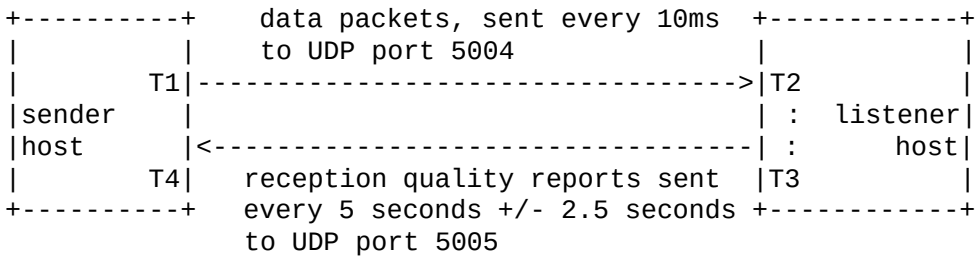
Write another program, `listener`, that listens to the packets being sent by the `sender` program and generates reception quality reports:

- 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 packets lost in transit. The transmission and reception timestamps should be stored in a file for later analysis.

- At intervals chosen randomly in the range 2.5 to 7.5 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 IP address of the sender host should be specified on the command line). These packets simulate the RTP control protocol, used for reception quality reporting in voice over IP systems. They should be 70 bytes in length, and should report the fraction of the packets sent on port 5004 that have been lost since the last Reception Quality Report was sent. For the purpose of this test, the remainder of their contents are unimportant.

The two test programs should be written in C using the sockets API for UDP/IP communication. Each program should be multithreaded, with one thread handling sending/receiving on port 5004, and another thread handling port 5005.




```
T1: thread sending data packets
T2: thread receiving data packets
T3: thread sending reception quality reports
T4: thread receiving reception quality reports

Threads T2 and T3 need to communicate, to pass reception quality
information from T2 to T3.
```

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 a timing graph – similar to that shown in lecture 15 – of send time versus arrival time for the system, over a period of approximately one minute. 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.

Comment on the data presented in your two timing graphs. Do the Linux systems you tested have sufficiently accurate timing to be suitable for use in a voice-over-IP system? You may need to zoom in on a small part of the graph to observe the timing properties. If packets were to be sent at a higher rate (smaller inter-packet spacing) to reduce end-to-end latency, would these systems still be usable? What is the minimum inter-packet spacing that the hosts can support while maintaining reasonable timing behaviour? Would use of the POSIX real-time scheduling extensions improve the performance of the system?

This assignment is due at 5:00PM on Monday, 11th April 2005. The marks will be assigned 40% for the design and implementation of the `sender` program, 40% for the design and implementation of the `listener` program, and 20% for the timing graphs and associated discussion. The code will be marked for correctness of the threading and networking code, and does not need to have a "pretty" user interface. You should submit the source code to the two programs, any design notes you have made, the two timing graphs, and a discussion of the timing properties of the system, including answers to the questions posed.