VoCe: Voice Conference

CO324 project 1

February 11, 2016

1 Problem description

Peer to peer communication is a strategy used in applications such as Bittorrent, and Skype. You will design and code a basic peer to peer voice conferencing application similar to Skype in two iterations.

In the first iteration you will implement voice communication between two parties. You will extend the application to support multi-party call conferencing in the second iteration. Finally, you will measure the performance of you application under real-world conditions using a network emulator.

1.1 Iteration 1 requirements

The sample code provided demonstrates the use of Java audio APIs to record audio play back audio on the same machine. Extend this code to transmit and receive full-duplex audio over a network between two parties. You will need to use separate threads for recording and transmission, and reception and playback.

Your application should take the peer's hostname or IP as a command line argument. Assume the peer is directly reachable by their IP addresses (no NAT is involved.) You are not required to implement a GUI.

1.2 Iteration 2 requirements

In iteration 2 you will implement multi-party conferencing using UDP multicast. Assume that only one party is speaking at a time, so there is no need to handle mixing multiple audio streams.

Your enhanced application should take a multicast group address as a command line argument. Assume all participants are directly reachable by their IP addresses (no NAT) and, that multicast functionality is available. However, not all participants need be on the same (wireless) LAN or subnet.

2 Design and testing

Before writing any code, plan the structure of your application for ease of modification and testing.

Begin by define your application message framing atop UDP. You must have appropriate headers for handling packet loss and reordering. Separate logic that handles packet loss and reordering and message serialisation/deserialisation code into individual classes.

You are required to write unit tests for at least your serialisation/deserialisation code. Additional unit tests will earn you bonus marks.

3 Performance measurement

To test performance of your application under packet loss and reordering conditions use the *netem* Linux network emulator introduced in the course labs. First, modify your code to print statistics on the number of packets lost or received out of order every minute.

Collect the statistics by varying UDP packet size, and the netem loss and delay parameters. Include test results, observations and results in your project report.

4 Marking criteria

You submission will be marked according to the following criteria.

1. Correct implementation of each iteration's requirements: $2 \times 30\%$.
2. Good design and unit tests: 20%.
3. Performance measurements and analysis: 20%.

In addition to code submissions for each iteration, you must write a report which describes the message format, application design, testing and performance measurements. Limit your report to 8 pages (11pt Roman font.)

Bonus marks will be awarded for creative features except a GUI. However, first ensure that the basic functionality works exactly as defined before attempting to add bonus features!

Late penalties will apply as per course and department policy. **Remember that plagiarism gets you zero!**

5 References

- Java Sound tutorial
- Getting started with JUnit
- Multicast howto
- Multicast example in Java
- Introduction to network emulation
- How to define netem rules