CO324 Project 1

Voice Conference (VoCe)

Group 2

E/12/050

E/12/324

Abstract

Peer to peer communication is a strategy used in modern applications. We will design and code a basic peer to peer voice conferencing application similar to Skype in two iterations. The first iteration is capable of handling full duplex communication between two parties .The second iteration handles half duplex communication between multiple clients using UDP multicast. The software is tested for its in several conditions using ‘netem’ a tool emulate network conditions in Linux.

Application Design

The final design of our project is a command line Voice Conferencing application with multi-party Conferencing capabilities. First the application gets user from arguments and then give the real-time voice conferencing capability. The basic mode is normal peer to peer voice conferencing. First user runs application an waits for an incoming call, if a call received then the user should press 'ENTER' key to accept call. After accepting the call both the users can communicate through voice using microphone and speakers. The next mode is multi-party Conferencing which uses a multicast address to multicast the audio streams. User should ask the software to run as multi-party conferencing by passing argument in command line as '-Multicast' and giving an IP address to multicast. The systems use a cache of 1024 packets to store the incoming packets. Message format of the packets are described below. The program also has a threshold value that is to hold the program until the specific number of packets is filled in the cache. It allows us to handle the packet loss, reordering, and delays. This program will use extra 3 Threads for

1. Receiving Packets
2. Recording audio and Send the Packets
3. Play the received packets from the received buffer

How to Use Application

To use this application in following modes are available in current version

1) Normal Voice Calls between two parties.
   The First user (Willing to receive a call) should Type the following commands in command line
   "Java VoCe"
   The Second user (Who is ringing the First user) should type the following
   "java VoCe [IP Address of the peer]"
2) Multi-Party Conferencing
   Anyone who would like to join the conference should type the following in command line

   "java VoCe [Multicast Address of the Group] [-multicast]"

First Iteration

The program uses three threads for sound capturing serialization sending , receiving packets and deserialization and paly buffer to enable full duplex communication.

1. • Initialize communication parameters <ip address>
2. • Record audio sufficient to fill the buffer
3. • Add serial number to packets in the buffer and send packets
4. • Start new thread and receive packets in secound party side and deserialize them
5. • Start new thread and paly received deserialized packets in buffer

Second Iteration

In this iteration also we use three threads for sound capturing serialization sending , receiving packets and deserialization and paly buffer to enable full duplex communication

1. • Initialize communication parameters <Multicast ip> <server/client>
2. • Record audio sufficient to fill the buffer
3. • Add serial number to packets in the buffer and multicast packets
4. • Start new thread and receive multicaseted packets in secound party(who join as client) side and deserialize them and keep it on buffer
5. • Start new thread and paly received deserialized packets in buffer

The Message Format

This is the message format that is used in the VoCe (Voice Conferencing) application. This is 500bytes in default. ; 496 bytes allocated to audio signal and last 4 bytes are allocated for sequence number. This starts from one and go up to 2^{31} . Which is more than enough for a single call, that is this application uses audio quality such that for 1 second it needs only 128 packets of data so with 2^{31} user is able to call for a long period.

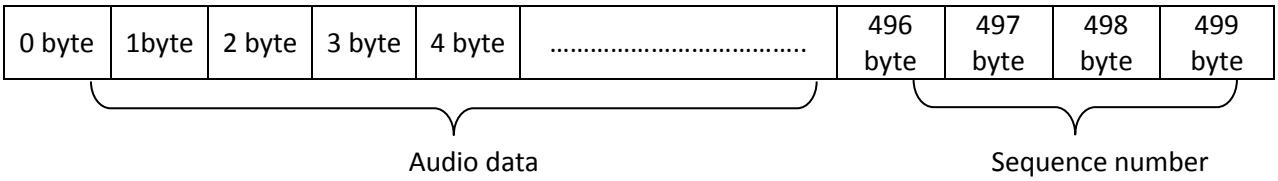

Figure: UDP Packet of VoCe Application

Handling loss and reordering

In multimedia applications loss can be tolerated. Therefore In case of packet loss, application isn't request to resend that packet it paly what is in the buffer. Our implementation has a buffer of 1024 packets. For a weak connections applications waits for some time to fill the buffer. So then the small reordering issues will be solved automatically. That is the application will not play the packets as it receives unless the connection is perfect. So this application stores the receiving packets in the buffer and plays back from the buffer .Application also have a common variable which shows what packet is being played. If the receiving packets serial number is lower than the current playing one it will neglect the packet as that packet is having unallowable reordering. These extreme reordered packets are discarded automatically by the application design.

Concurrency

There is not any concurrency issue in our application unless the static variable which calculates the current playing audio packet and current sending audio packets. These cases are negligible as there are huge number of audio packets are receiving at a second. Concurrency errors will only affect for a packet or two to drop or to playback older packet this will not affect the overall performance of the application.

Special features

We implemented a special feature to automatically calibrate the threshold value. The Threshold value is the number of packets that should be kept in the buffer until the playback. The Application always checks whether there are enough values in the buffer. This technique allows Application to run with reasonable call quality even with 40% packet loss! Also this application is capable of making calls even with 0ms to 1000ms vitiating delay of packets!

So if the connection between the two connections is perfect this buffering will add a delay to the call. But from the use of auto-calibrating according to network connection will detect that the connection is perfect and it will not store a buffer and plays the packets almost real-time.

The basic idea behind this is to give user a reasonable call quality with poor connection. But if the connection is poor it will not be real-time but may add 100ms to 5000ms delay according to connection quality.

Testing and Performance Measurement: Observations and analysis

Following conclusions are obvious as per the results obtained:

1. jitter is the main factor in packet reordering.
2. Delay has no connection with loss rate and packet reordering.
3. Number of lost packets proportional to loss rate

Following qualitative observations too were made during the test.

1. Delay makes no change to the clarity of voice call.

NOTE: complete results are in separate file result.xls in submitted folder.