VoCe - Voice Conference CO324 Assignment

Aslam M.M.M. (E/15/021)

Hisni Mohammed M.H. (E/15/131)

Suhail S. (E/15/348)

Abstract

Peer to peer communication is a strategy used in modern applications. A basic peer to peer voice conferencing application similar to Skype is implemented in two parts. The first implementation is capable of handling full duplex communication between two parties. The second implementation handles half duplex communication between multiple clients using UDP multicast. The software is tested under real-world conditions using a network emulator.

Application Design

Our project is a command line Voice Conferencing application with multi-party Conferencing capabilities. First application gets user arguments from command line and then give the real-time voice conferencing capability.

The first implementation is normal peer to peer voice conferencing between two persons. First, one user runs the application with IP address of other person as command line argument and other person also runs the application with this persons IP address. then application automatically accepts the packet and after that both the users can communicate through voice using microphone and speakers.

In this program the transmission happens through a simple DatagramSocket in the. This program uses two separate threads. One for recording and sending packets and other one is for receiving and playing the packets.

The second implementation is multi-party Conferencing which uses a multicast address to multicast the audio streams. User should run the application with a multicast IP address by passing argument in command line. Then other persons also join the group by running the application in the same way with same multicast IP address.

We assumed that only one person is speaking at a time. Person wants to speak needs to press ‘T’ to talk and need to press ‘E’ once finished speaking.

The systems use a cache of 512 packets to store the incoming packets. 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 4 Threads each for

1. Recording audio and Send the Packets
2. Receiving Packets and add to Buffer
3. Play the received packets from the buffer
4. Interacting with user & printing statistics

How to Use Application

1. Voice Calls between two parties.

   User should compile and run the VoCe.java file

   java VoCe <Destination_IP>

   One user should use other persons IP address as Destination IP and other person should use this persons IP address.

2. Multi-Party Conferencing

   User should compile and run the VoCe.java file

   java VoCe <MulticastIP>

   Any persons can join the group conference call by running the application with same multicast IP address. IP address should be in the range 224.0.0.0 to 239.255.255.255

First Implementation

The program uses two threads for sound capturing & sending and receiving packets & paly to enable full duplex communication. Here DatagramSocket is used for transmission.

1. Initialize communication parameters <ip address>
2. Start the Sender thread and keep record audio sufficient to fill the buffer and once the buffer is filled send packets and keeps doing it in infinite loop
3. Start new thread and receive packets paly the received packets in buffer and keeps doing it in infinite loop.

Second Implentation

This program uses four threads for sound capturing serialization sending, receiving packets and deserialization, paly buffer to enable full duplex communication and interact with user to record voice and send or not. Here MulticastSocket is used for transmission.

1. Initialize communication parameters <Multicast ip>
2. Start 4 Threads
3. Receiver thread waits in infinite loop for receive packets, once packet is received add the packet to buffer.
4. Sender thread sits on infinite loop, and according to users state it will record and send packets or will do nothing.
5. Interaction thread also runs in an infinite loop and gets user inputs and mage the states.
6. Player also runs in an infinite loop and check the buffer once the buffer is fill with number of packets equal to threshold the thread will play the packets.

The program stays in two state. One is Record and Send while listening state. And the second is listening only state. If the user enters 'T' app is in state one and if user enter 'E' app sits on the second state. Since in both state app keeps listening to packets user can observe the voice and can/need to determine the state of the app. Here we assumed only one person is talking at a time therefor user needs to determine the app state.

Sender uses a sequence number and User ID which is the hash value of local IP address. DataPacket is created using these parameters and along with the voice. Since DataPacket class implements the serializable interface this object is serializable. Receiver deserialized the received packet and process the packet according to the sequence number and user ID.

DataPacket Class

Data Packet class implements the java Serializable class. Therefore, DataPacket is serializable using ByteArrayInputStream and ByteArrayOutputStream. DataPacket object contains Byte array of Voice, sequence of number and ID of the user who created the packet. Extra 103 bytes are occupied for these functionalities.

```
public DataPacket( byte [] voice, long sequenceNo, int id ){
                this.voiceBuffer = voice;
           this.sequenceNo = sequenceNo;
           this.id = id;
}


public static DataPacket ByteArrayToObject(byte [] data) throws IOExce
ption{

     ObjectInputStream  ois = null;
     try{
           ByteArrayInputStream b = new ByteArrayInputStream(data);
            ois = new ObjectInputStream(b);
            return (DataPacket)ois.readObject();
     }catch(Exception e){
           e.printStackTrace();
            return null;
      }finally{
           ois.close();
      }
}
```

```
public static byte[] ObjectToByteArray( DataPacket packet ) throws IOExceptio
n{
      ObjectOutputStream  oos = null;
      try{
            ByteArrayOutputStream buffer = new ByteArrayOutputStream();
            oos = new ObjectOutputStream(buffer);
            oos.writeObject(packet);
            oos.flush();
            return buffer.toByteArray();
      }catch(Exception e){
            e.printStackTrace();
            return null;
      }finally{
            oos.close();
      }
}
```

ByteArrayToObject() method is used to convert a byte array to DataPacket object and ObjectToByteArray() method is used to convert a DataPacket object.

Other class 'Sender', 'Receiver' and 'Player' are extends the java Thread class. Multicast socket is used for sending and receiving packets. Sender Multicast socket is created without binding to any port. But receiving socket is bind to particular fixed port. Therefore, all packets can send to this port.

Handling loss and reordering

In multimedia applications losses can be tolerated. Therefore, the application is not handling the packets losses. 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 512 packets. For weak connections, applications wait 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 uses a threshold value of 32. So packets are played if buffer is filled with at least packets received is equal to threshold value.
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 no concurrency issue in our application unless the static variable which calculates the current playing audio packet and current sending audio packets. 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. Since there are huge number of audio packets are receiving at a second these cases are negligible.

Reference

1. https://docs.oracle.com/javase/tutorial/sound/sampled-overview.html
2. https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/ip_multicast/White_papers/mcst_ovr.html#wp1008683
3. https://github.com/junit-team/junit4/wiki/Getting-started
4. https://calomel.org/network_loss_emulation.html