VoCe: VOICE CONFERENCE

(1st ITERATION)

E/12/058, E/12/188

GROUP 14

SEMESTER 5

05/03/2016

Introduction

In this project we are designing and implementing a program with basic peer to peer voice conferencing application similar to Skype in two iterations.

In the first iteration we implemented a voice communication between two parties which allows both parties to talk in both sides in the same time. Also in this iteration we also implemented the system to run multiple voice communication between these two machines.

In the second iteration we modified the program to multi-party conferencing using UDP multicast. We implemented the program assuming that only one party is speaking at a time. Our application takes a multicast group address as a command line argument. We will assume all participants are directly reachable by their IP addresses and multicast functionality is available.

Classes

- DemoRun.java

  This is the class which we run the voice conference program. Simply by entering the multicasting ip address as an input we can run the program with this java class.

- PacketFormat.java

  With this class we define a new packet format to send packet sequence number to other party.

- CaptureAudio.java

  With this we can capture the sending Audio Stream process on it.

- PlayAudio.java

  With this class we can play the captured audio stream.

- AudioOut.java

  With this class we can send the audio stream to other multicasting stations through multi casting network.

- Record.java

  To find the packet loss we stored the packet sequence number in an array-list using this java class.

Designing

- Data Serialization

  In the PacketFormat.java class we used data serialization to convert a data object into a byte data stream. With the "serialize" method we can do that serialization to the data object.

```
public static byte[] serialize(Object obt) throws IOException {
      try(ByteArrayOutputStream baos = new ByteArrayOutputStream()){
            try(ObjectOutputStream oos = new ObjectOutputStream(new
       BufferedOutputStream(baos))){
            oos.writeObject(obt);
         }
         return baos.toByteArray();
      }
}
```

  And in the same PacketFormat.java we have a method called "deserialize" to convert back the converted byte data stream to a data object.

```
public static Object deserialize(byte[] bytes) throws Exception {
       try(ByteArrayInputStream baos = new ByteArrayInputStream(bytes)){
        try(ObjectInputStream oos = new ObjectInputStream(new
        BufferedInputStream(baos))){
                  return oos.readObject();
        }
        }
}
```

- Handling losses and reordering

 In this system timing is very important because in a conversation time delay is affecting the conversation greatly. In this conversation normally delay should be less than around 200 milli seconds. In this program there is a little bit of a delay when capturing and playing the multicasting data. Also there will be packet losses when multicasting the data because of sometimes packets might come in a different order and also packets cab be lost. We need to minimize these packet losses to have a smooth conversation. In this program when programming we calculated the amount of packet lost in 10s time durations. According to those packet losses we coded the program to minimize it.

 Normally in a conversation we don't need to retransmit data because it will repeat the past conversation. So we need to minimize the packet loss as discussed above.

- Concurrency

 In this program to handle concurrency we used threads. In the CaptureAudio.java and PlayAudio.java we used threads to control capturing and playing the data stream in the multi casted system. When we multicast we need to check whether there is an established connection or not. We used threads to control this.

Test

When running the program we implemented the system to show the packet loss in each 10s that we ran the program. With that we can test the program and check the errors. Then we can correct those errors. With this method we can test the program.

```
if((endTime - startTime)> 10000){
        System.out.println("No of lost packets in 10s - "+ rec.getLsCount());
        startTime = new Date().getTime();
        rec = new Record();
}
```

Performance metrics using netem

We used netem to calculate the percentage of the packet loss. There is approximately 200 to 300 packet losses between 10s durations. It is around 10% percentage of sending packets.