CO324: VOICE CONFERENCE

E/15/180, E/15/243, E/15/271

GROUP 20

SEMESTER 5

21/10/2019

TABLE OF CONTENT

1) Abstract
2) Introduction
3) Description of Classes
4). Designing
- Data Serialization
- Handling losses and reordering
- Concurrency

5) Test
6) Performance metrics using netem
7). References

ABSTRACT

Today, communicating with people who are living all around the world, has been a very popular and needy thing. Therefore, an audio conference system that gives chances people to chat with each other even they are in different places in that world, has a more demanded among other implementations.

Then, in the project, Voice conference, java application has to be designed and coded as a basic peer to peer voice conferencing application that is similar to Skype in two iterations.

1. In the first iteration, voice communication which is between two parties was implemented.
2. In second iteration, first iteration was extended to support multi-party call conferencing

In this project, it is based on a client-server type of application. The server handles all the traffic. The person who is from one network and desires to call with another person who is belonging to the other network, has to send a request to the server. Thereafter, the server accepts the request and a successful voice conversation can be held.

So, this application was implemented in-order to transfer voice message to a single person or group who are connected through a Wi-Fi network. This is a real time application. When a server records the message, it will be broadcast to the connected Wi-Fi clients.

INTRODUCTION

Peer to peer communication is a strategy that is used in applications such as BitTorrent, and Skype. In this project, a basic peer to peer voice conferencing application which is similar to Skype was implemented in two iterations.

First, voice communication between two parties was implemented. This allows both parties to talk in both sides in the same time. Not only that, but in this iteration, a system also was implemented to run multiple voice communication between these two machines.

Then the application was extended to support multi-party call conferencing using UDP multicast as the second iteration in this project. However, while implementing the program, it was assumed that the only one party is speaking at a time. This created application takes a multicast group address as a command line argument. Furthermore, it was assumed that all participants were directly reachable by their IP addresses and multicast functionality is available.

Finally, the performance of our application was measured under real-world conditions using a network emulator.

DESCRIPTION OF CLASSES

Following are the classes which have been created in application *for peer tp peer communication* and the reasons for creating each of them.

- Client.java

  This is the class which has to be run in order to play the voice conference program. By typing the ip address of peer as an input, it can run the program with this java class.

- Record.java

  Record sound into a byte array format it and pass to send.java

- Send.java

  Sending the byte array

- receiveAndPlay.java

  Take incoming packets and play them

- PacketFormat.java

  Format data to packet, including a sequence no and identify the sequence no of incoming packets.

Following are the classes which have been created in application *for multicasting communication* and the reasons for creating each of them.

- DemoRun.java

  This is the class which has to be run in order to play the voice conference program. By typing the multicasting ip address of peer as an input, it can run the program with this java class.

- PacketFormat.java

  This is the class that defines a new packet format to send packet sequence number to other party.

- CaptureAudio.java

  To capture the sending Audio Stream process, this class was implemented.

- PlayAudio.java

  The captured audio streams can be played using this class.

- AudioOut.java

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

- Record.java

  The program was designed to find the packet loss also. For that criteria, packet sequence number in an was stored in an array-list using this java class.

DESIGNING

- Data Serialization

  In the project, data serialization was used in order to convert data objects into a byte stream. Then, that data serialization would be done at the method called "serialize" in the class PacketFormat.java.

```
//serialize the byte stream to the 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();
        }
}
```

In the same PacketFormat.java class, a method called "deserialize" also was implemented to convert 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

For a perfect performance of this application, there were some important things to be managed. Among of those important things, time was considered as the most important one, because time delay would affect to have bad and disrupt connection in a conversation. To achieve a good conversation with other parties through this program, time delay should be less than around 200 milli seconds.

When capturing and playing the multicasting data, there is a little bit of a delay in this program. Not only that, but there will be packet losses also when multicasting the data. This is because sometimes packets might not come in order as well as they will be lost. So, it was a part of error handling. With minimizing those packet losses, a smooth conversation was expected through our project. For that, the amount of packet lose would be calculated in 10s time durations. According to those packet losses, the program was coded to minimize it. This is done in the class PlayAudio.java.

Anyway, in the code, there would not be anything programed to retransmit data in the conversation. This is because it will repeat the past conversation. So we need to minimize the packet loss as discussed above.

- Concurrency

Threads was used to handle concurrency. In the CaptureAudio.java and PlayAudio.java, threads was used to control capturing and playing the data stream in the multicasting system. For the multicasting, it is needed to check whether there is an established connection or not. To control this also, threads were used.

TEST

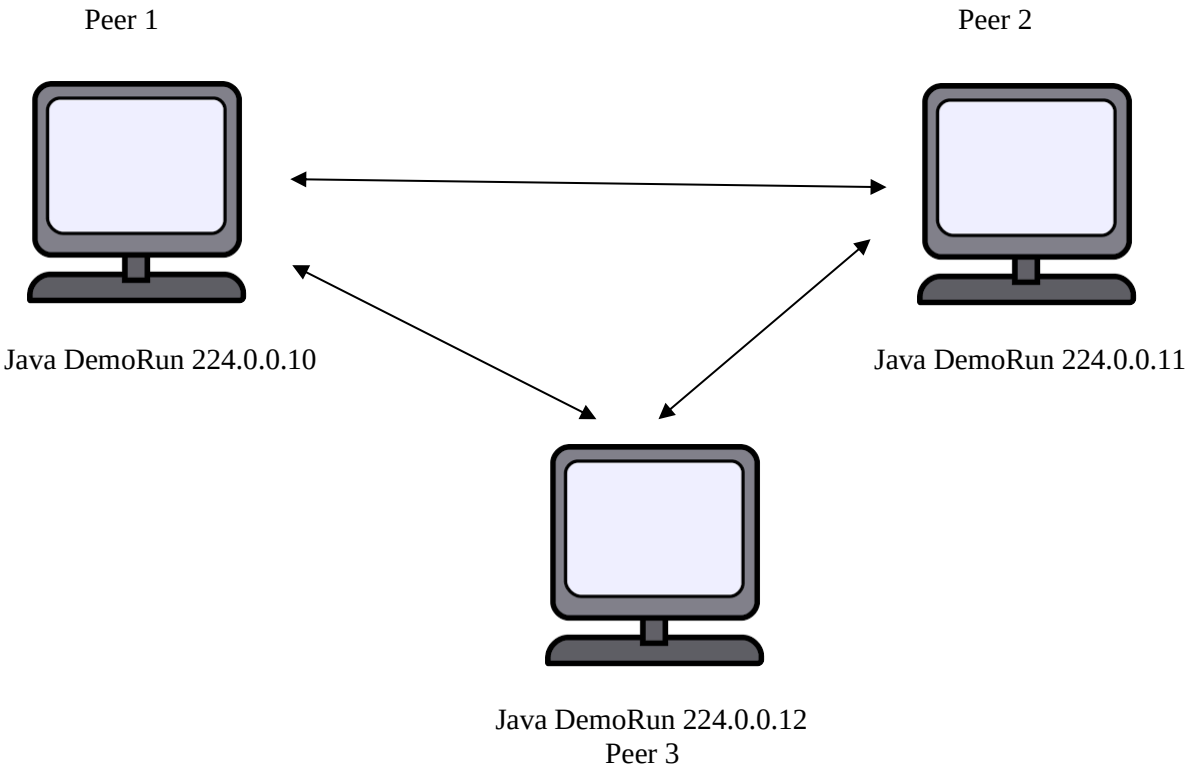
As mentioned above, implemented the system was designed to calculate the packet loss in each 10s when running the program. From that, the program can be tested to check the errors. Follwing method was implemented to test the program.

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

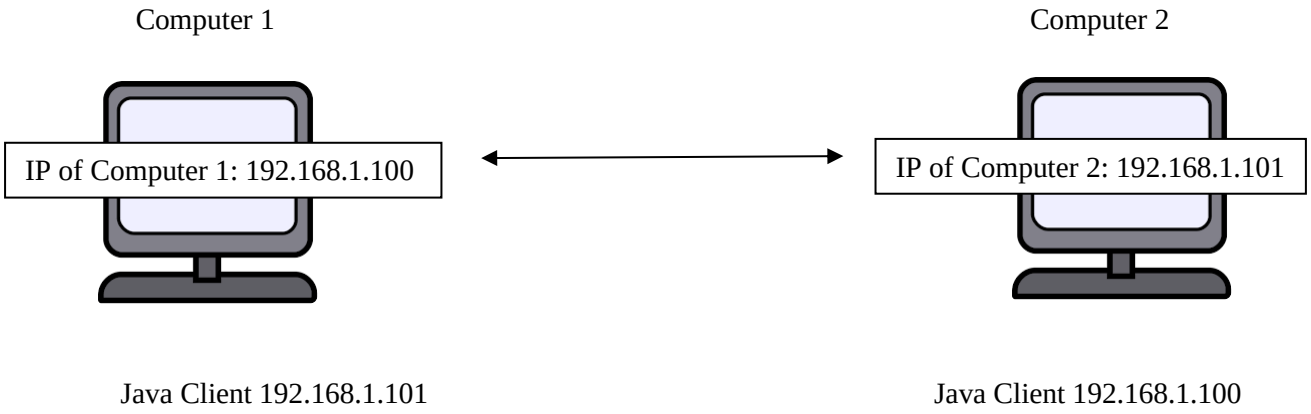
Then, if there is an error, it can be corrected.

- To test on more than two machines
    - Java DemoRun <IP address>

      In here, IP address should be a one which is in range of multicast address. All IP multicast group addresses fall in the range from 224.0.0.0 through 239.255.255.255.



- To test on two machines



PERFORMANCE METRICS USING NETEM

Netem was used to calculate the percentage of the packet loss.

The following will show results at voice conference

```
sewwandi@sewwandi-Inspiron-3543:~/Documents/aca/CO324 Network & Web application/Group 20 Assignment$ java DemoRun 224.0.0.10
Available mixers:
0 Port HDMI [hw:0]
1 Port PCH [hw:1]
2 default [default]
2 Mic is supported!
Available mixers:
0 Port HDMI [hw:0]
1 Port PCH [hw:1]
2 default [default]
2 Mic is supported!

No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
No of lost packets in 10s - 0
```

REFERENCES

1. https://calomel.org/network_loss_emulation.html
2. https://docs.oracle.com/javase/tutorial/sound/sampled-overview.html
3. https://github.com/junit-team/junit4/wiki/Getting-started
4. https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/ip_multicast/White_papers/mcst_ovr.html#wp1008683
5. https://www.cs.unm.edu/~crandall/netsfall13/TCtutorial.pdf
6. http://www.tldp.org/HOWTO/Multicast-HOWTO-2.html