



San José State
UNIVERSITY

CMPE 207 - Project Report

VIDEO STREAMING
GROUP 12

Professor: Younghee Park
San Jose State University

Team Members:

Rohit Kotalwar	(010805080) Class ID - 27
Ramya Chowdary Nadella	(010100649) Class ID - 42
Pooja Prakashchand	(010814258) Class ID - 53
Rakesh Datta	(010808447) Class ID - 15

Table of Contents

1. Abstract.....	3
2. Introduction	4
3. Protocol Design.....	4
4. RFC 1889 - The Real Time Transport Protocol (RTP).....	5
i. RTP header fields:.....	6
5. Application Design	6
6. Terms and API's used in our Project	7
7. Concepts involved in Java for Multithreading in our project.....	7
8. Concepts involved in Java for Socket Programming in our project	8
9. Server side implementation	9
10. Project setup.....	11
11. Handling concurrency in the Server	12
i. Code snippets to explain the concurrency achieved.....	13
12. Multicasting.....	14
i. Code snippets to explain the Multicasting	15
13. Deployment of the Server and the Client.....	15
14. Execution Snapshot	17
15. Test Cases.....	19
i. Test case 1: Testing the streaming sequence of video for multiple clients:	20
ii. Test case 2: Statistics for Response time:	20
iii. Test case 3: Statistics for Packet Drop:	21
iv. Test case 4: Currency Testing using JMeter:	21
16. Challenges faced and challenges solved	21
17. References	22

Abstract

Computer networking is designed to connect multiple hosts on various locations and also to enable the communication between them such that the hosts can share data like text files, audio and video files. Video streaming has been extensively used for information broadcasting. There are multiple ways we can transfer a video file over a network. One of the fastest and efficient way is streaming. Video streaming refers to continuous transmission of video file between users in real time. Real time video transmission is widely used in surveillance, conferencing, media broadcasting and applications that include remote assistance. In this project, we leverage the strength of media streaming and design a video streaming server that communicates to a client using RTP (Real Time Transport Protocol) for media exchange. We deployed the Server and Client code on a Home Network. For testing the code, we have created test cases for measuring the Server Response time, analysing the packet loss, testing the streaming sequence of video for multiple clients and verified the maximum number of clients that the server can handle.

Introduction

A video Streaming server provides continuous streaming of a video to the client that is requesting for it. Availability of improved and enhanced transmission facilities (i.e. high speed LAN, wireless Ethernet) make it further possible to use video streaming in fast real time applications.

Some of the characteristics of the video streaming server are:

- It allows the user play the video on the go, without waiting for the entire file to be downloaded.
- Capable of sending payload using RTP.
- The video streaming server can unicast video to a single user requesting for it, broadcast the data to all, or it can multicast it to a group of users at the same time.

Protocol Design

The RTP protocol has been used for the transmission of real-time streams. RTP defines a standardized packet format. It is also a network control protocol which delivers audio and video over IP networks. It solves the media frame packing, the timestamp and the synchronization issues by adding a special RTP header to the packed media data. The format is suitable for both unicast and multicast transport.

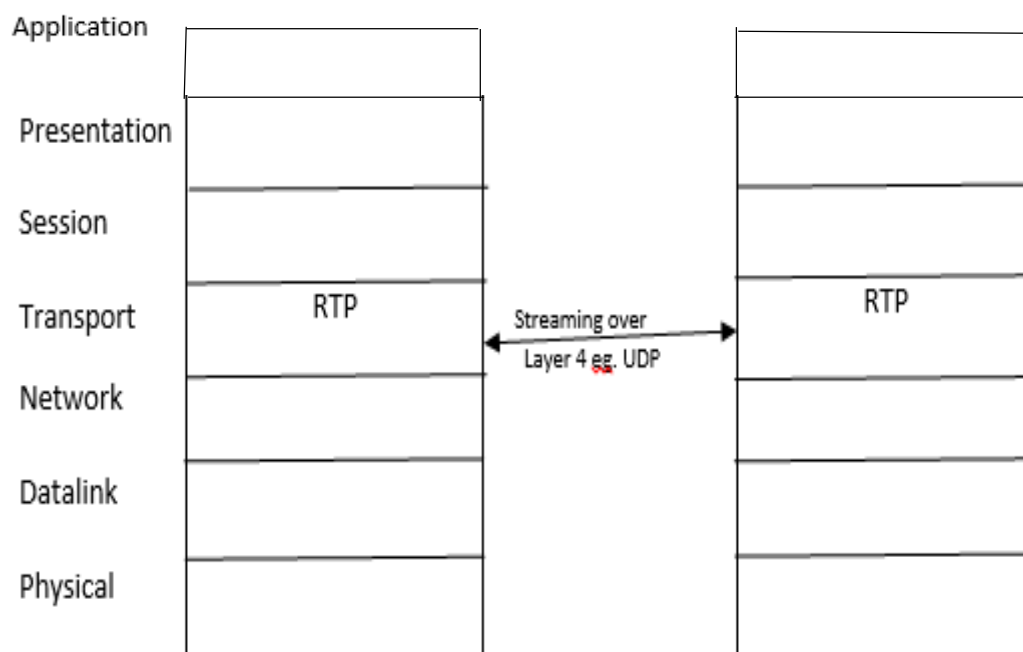


Figure 1: Protocol design with respect to OSI layers

RFC 1889 - The Real Time Transport Protocol (RTP)

The Real Time Transport Protocol (RTP) defines a model for transmission of real-time media. The model describes packaging methods of data with real-time characteristics, such as interactive audio and video. Moreover, the RTP suite defines a control protocol. The RTP Control Protocol (RTCP) allows monitoring of the media data delivery. It adds minimal control and identification functionality to the RTP stream.

RTP neither guarantees timely delivery nor a continuous stream. On this matter it entirely depends on lower layer services. However, a Sequence Number Field is included in the RTP packet header in order to reconstruct the sender's packet sequence. But this does not prevent the lower network layers from sending out-of-sequence packets. It does not interfere into the transmitting logic of upper layers. The stream processing queues at the server often deliver media object parts out-of-sequence in order to force the clients to pre-buffer media presentation. Another important part of the RTP packet header is the Payload Type. It identifies the data format of the RTP payload. Furthermore, a Timestamp field in the RTP header field reflects the sampling instant of the first octet in the RTP data packet.

For video streaming, RTCP was defined to add minimal control to the RTP. This control is obtained by transmitting periodic control packets to all participants of a streaming session. The control packets are distributed using the same mechanism as the RTP data packets. RTCP adds four functions to RTP transmissions:

1. It provides feedback about the quality of the data distribution (diagnoses network problems).
2. It carries a persistent transport-level identifier called CNAME.
3. It helps to adjust the RTP packet sending rate.
4. It conveys minimal session information: for example session's participant identification.

The functions 1-3 are mandatory for an RTP transmission in an IP multicast environment. For UDP and similar protocols, the RTP stream uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number.

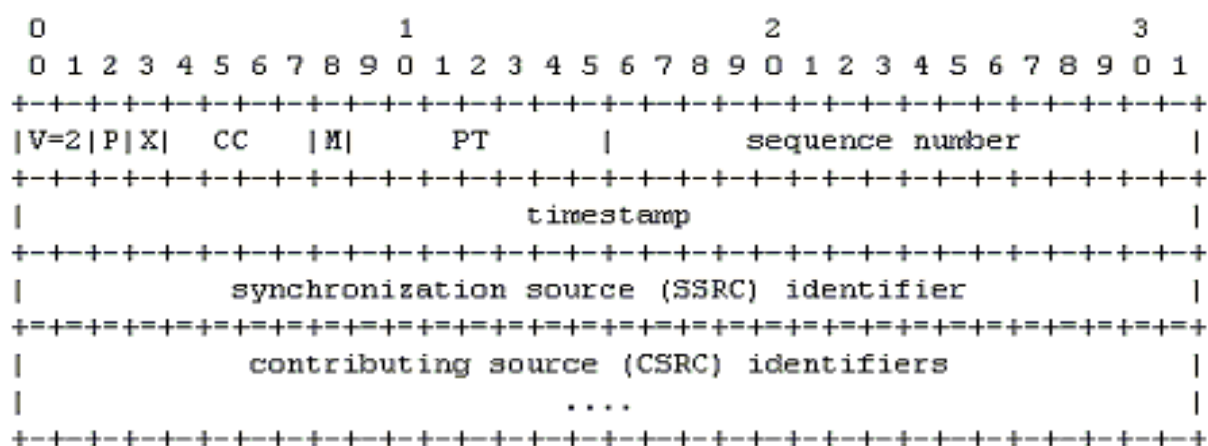


Figure 2: RTP Header format

RTP header fields:

The first twelve octets are the basic header fields in every RTP packet.

- **Version (V):** We have used version 2 for our specification. 1 is used for first draft version of RTP.
- **Padding (P):** We have set the padding to zero. If the padding is set to value other than zero, the packet contains additional padding octets at the end which are not a part of payload. This field is used by encryption algorithms or for carrying several RTP packets.
- **Extension(X):** We have set the extension to zero. If in a particular header, extension is set to a value other than zero, then the header is supposed to be followed by exactly one header extension.
- **CRSC count (CC):** We have set the CC to zero. The mixer inserts a CRSC list which is the SSRC identifiers which contribute to the generation of a particular packet into the RTP header of the packet.
- **Marker (M):** Few packets in the stream are marked such the boundaries for the frame to allow particular events.
- **Synchronization Source (SSRC):** SSRC is used for RTP implementations to prevent the collisions. It has to be unique within the same RTP session. So the SSRC is mostly chosen randomly.
- **Sequence Number:** We have initially used the frame number as the sequence number which increments on sending a RTP packet. Sequence Number is used by the receiver to detect the loss of packets and to restore the packet sequence.
- **TimeStamp:** We have used Time as the Timestamp which is derived from a clock that monotonically and linearly increases in time. The sampling instant for the RTP packets is established by referring the Time used while sampling the video. If any two videos or a video has to be synchronized with an audio, then their RTP timestamps can be related using timestamp pairs in RTP SR packets.
- **PayloadType:** This field identifies the type of the RTP payload which determines the understanding of the application.

Application Design

There are seven important classes in our application:

StaticVideoStreamingServer – It responds to each incoming request from the client. The server makes calls to RTPPacketizer class to packetize the video data. The server streams back the video to the client.

VideoStreamingClient – Implements the client and also the GUI. This GUI displays the video received from the server for Unicast Communication.

RTPPactetizer – It is used to packetize RTP packets. It receives the video data, constructs RTP packets and de-packetizes them at client side.

StreamVideo – This class is used to read the video from a file.

threadOfClient – This class is used to handle the number of clients by creating threads.

MulticastClient – This class implements the client and the GUI to display the video received from the server. Multiple clients can join the network to establish the one to many connection for video streaming.

RTPDepacketizer – This class implements the de-packetization of the packets received to render the video.

Terms and API's used in our Project

Concepts involved in Java for Multithreading in our project

Concepts	Description
Runnable interface	Used to implement the run() method when a thread is created
Thread	Thread is a pre defined class that implements Runnable interface and any class that extends Thread, overrides the run method
Thread.Start()	The start() method, runs the thread by calling the run() method.
Thread.Sleep()	The sleep() method allows the thread to sleep for the specified number of time.
Thread.run()	It is used to perform the actions when a thread is created.
Thread.join()	It is used to wait for a thread to complete.
Thread.getName()	It is used to get the name of the thread currently executing
Thread.isAlive()	It is used to check if the thread is alive.
Synchronization methods	It blocks the execution of the method by other thread when one thread is executing the method.

Table 1: Concepts involved for Multithreading in JAVA

Concepts involved in Java for Socket Programming in our project

Concepts	Description
Socket	Used for communication between two end points
Socket(InetAddress address, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address.
Socket(InetAddress address, int port, InetAddress localAddr, int localPort)	Creates a socket and connects it to the specified remote address on the specified remote port.
connect(SocketAddress endpoint)	Connects this socket to the server.
close()	Closes this socket.
getInetAddress()	Returns the address to which the socket is connected.
getLocalAddress()	Gets the local address to which the socket is bound.
getLocalPort()	Returns the local port number to which this socket is bound.
getPort()	Returns the remote port number to which this socket is connected.
setReuseAddress(boolean on)	Enable/disable the SO_REUSEADDR socket option.
DatagramSocket	Constructs a datagram socket and binds it to any available port on the local host machine.
DatagramSocket(int port, InetAddress laddr)	Creates a datagram socket, bound to the specified local address.
getReuseAddress()	Tests if SO_REUSEADDR is enabled.

<code>DatagramPacket(byte[] buf, int length)</code>	Constructs a <code>DatagramPacket</code> for receiving packets of length <code>length</code> .
<code>DatagramPacket(byte[] buf, int length, InetAddress address, int port)</code>	Constructs a datagram packet for sending packets of length <code>length</code> to the specified port number on the specified host.
<code>getInputStream()</code>	Returns an input stream for this socket.
<code>getOutputStream()</code>	Returns an output stream for this socket.
<code>notify()</code>	Wakes up the thread that is waiting.
<code>wait(long timeout)</code>	current thread waits until either another thread invokes the <code>notify()</code>
<code>getByName(String host)</code>	Determines the IP address of a host, given the host's name.
<code>MulticastSocket</code>	Create a multicast socket.
<code>MulticastSocket(SocketAddress bind addr)</code>	Create a <code>MulticastSocket</code> bound to the specified socket address.
<code>MulticastSocket(SocketAddress bind addr)</code>	Joins a multicast group.
<code>receive(DatagramPacket p)</code>	Receives a datagram packet from this socket.
<code>send(DatagramPacket p)</code>	Sends a datagram packet from this socket.

Table 2: Concepts for Socket Programming in JAVA

Server side implementation

Video data should be constructed into RTP packets. A packet is created, the fields of the header are set in the packet header and the payload (i.e one frame) is copied into the packet. When the server receives a request from the client, a timer is started. This timer triggers for every 100ms. At these times, the server reads one video frame from the file and send it to the client. The server creates an RTP packet object which is encapsulated video frame. The server calls the first

constructor of the class RTP packet to perform encapsulation. The RTP packet is constructed in the following format.

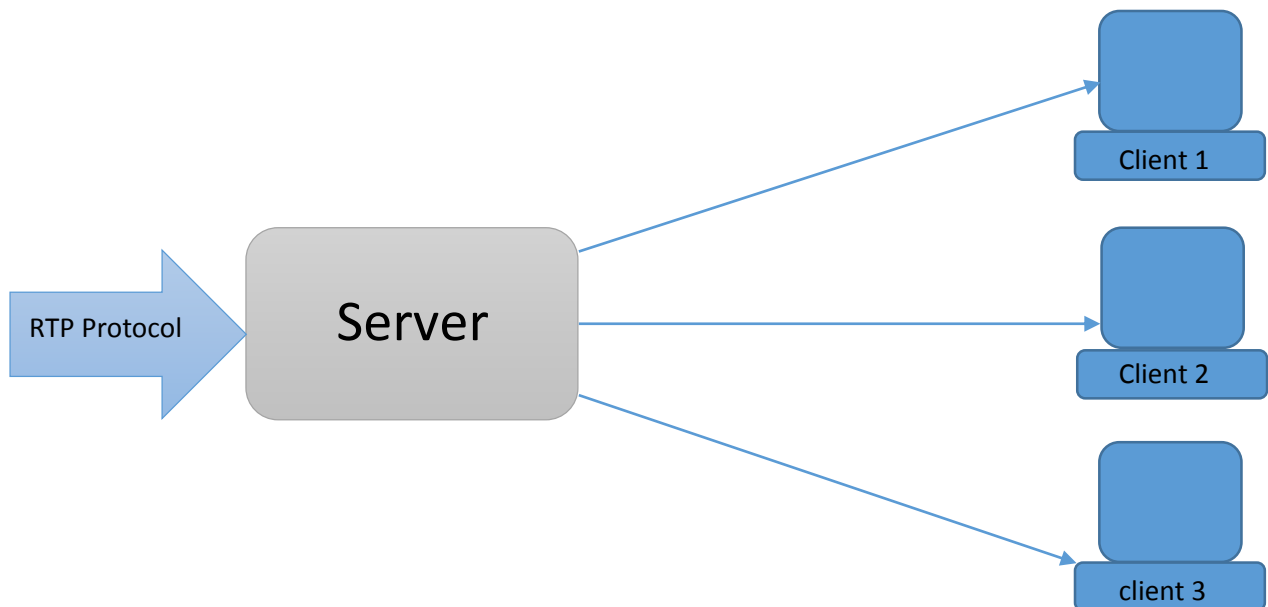


Figure 3: Server side Implementation

The below scenario diagram shows how the client and server interacts with the port number. TCP connection is established with a specified port. The client listens to the server on the specified port and receives packets by the server.

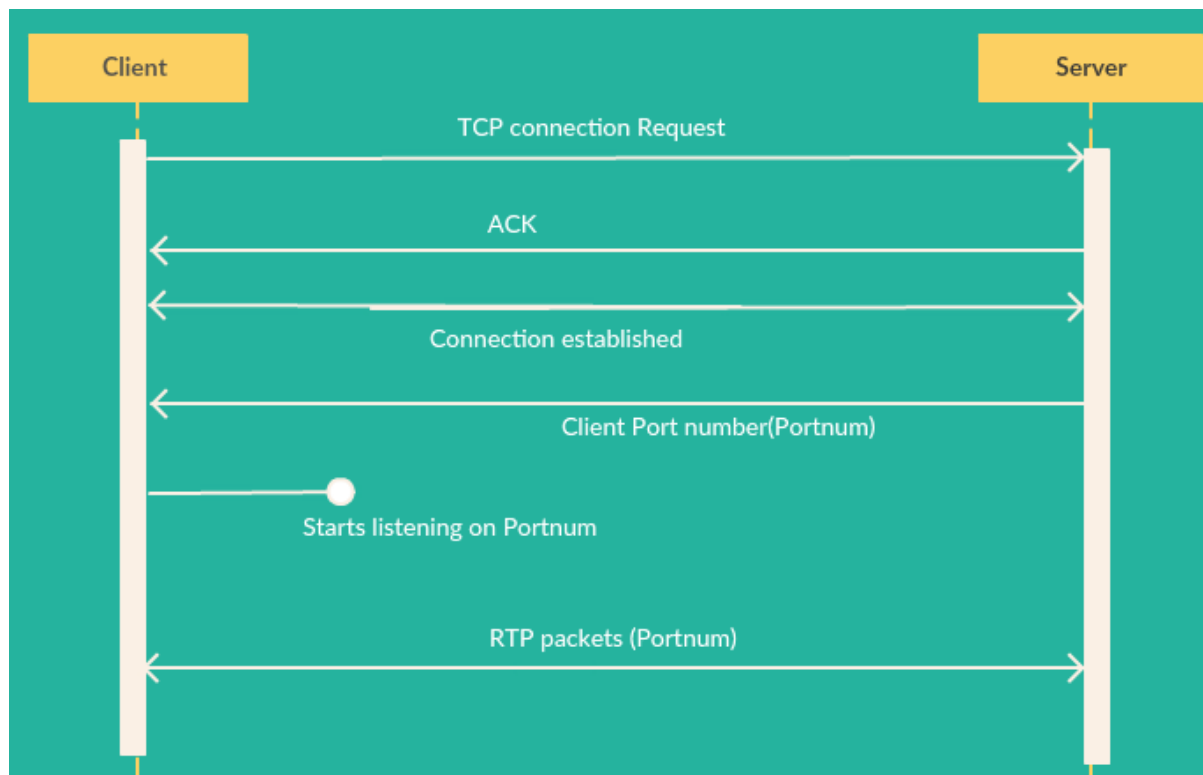


Figure 4: : Scenario diagram showing Client and Server interaction with the port number

Project setup

We used eclipse to set up the project. The below table shows the description of the various classes used in the project.

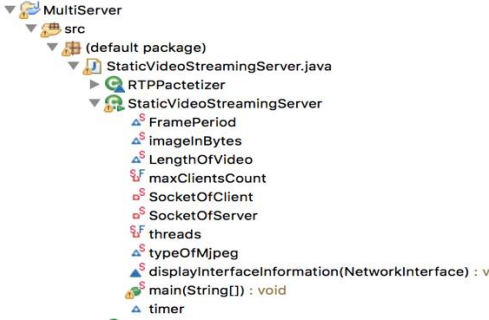
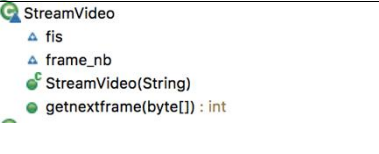
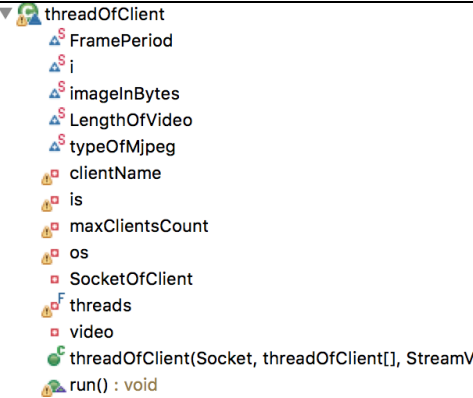
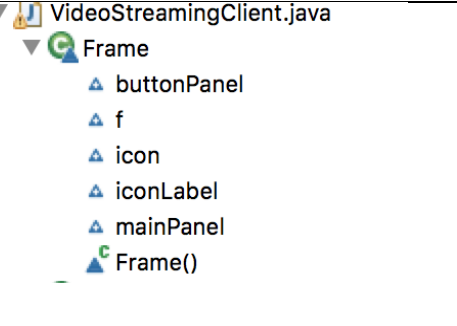
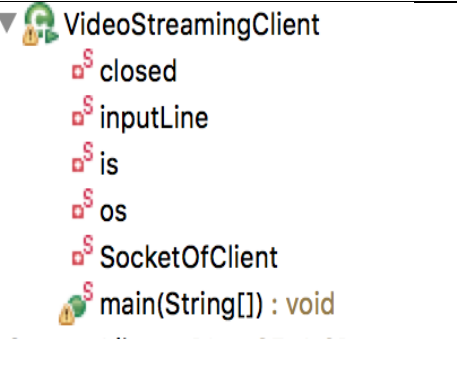
	<p>This is the main method where sockets are created.</p>
	<p>This is the Streaming video class where next frame is fetched</p>
	<p>This is the thread creation class where each thread handles a new client.</p>
	<p>This is the frame creation class where a GUI is created for each client.</p>
	<p>This is the client class that is used to connect to the server.</p>

Table 3: Project Setup

Handling concurrency in the Server

The server supports concurrency. Each client is served by the thread created. Java supports concurrent programming by which multiple threads can be created within a single process. A single sequential flow of control within a process is called a thread. Each thread has its own local and stack variables within a process. The client sends a request to the server. The server accepts the request and the video gets streamed on the client side. Multiple clients can request the server for streaming the video.

The scenario diagram below shows how the server handles a single client.

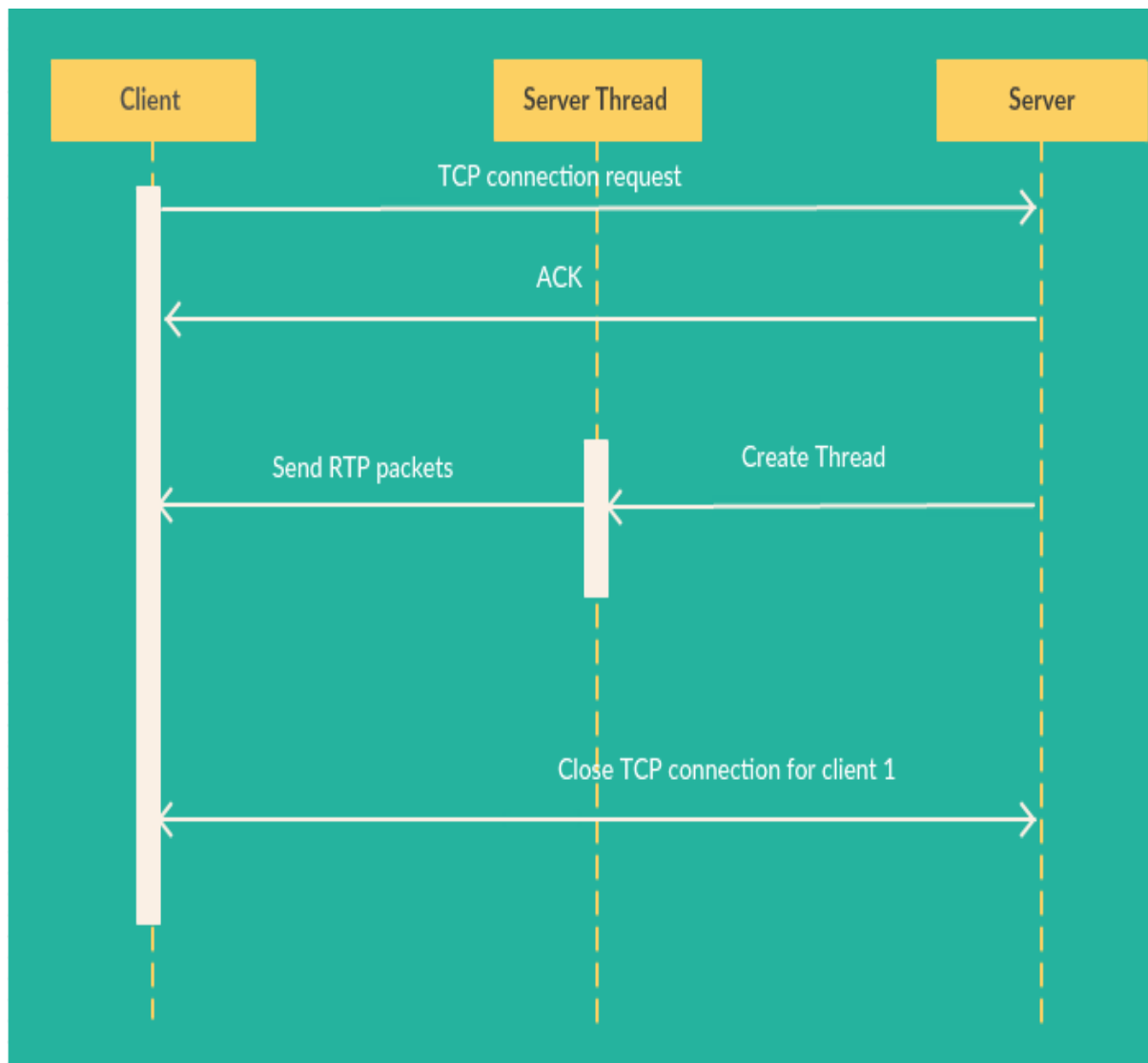


Figure 5: Scenario diagram of a server handling a single client

After successfully implementing a single client and server interaction, we implemented concurrency by created threads. Each thread handles a client. The client initiates a request and the server responds by creating a secure connection (TCP connection). Once a connection is established, RTP packets are sent so that the client streams the video.

The scenario diagram below shows the interaction of two clients with the server and the threads.

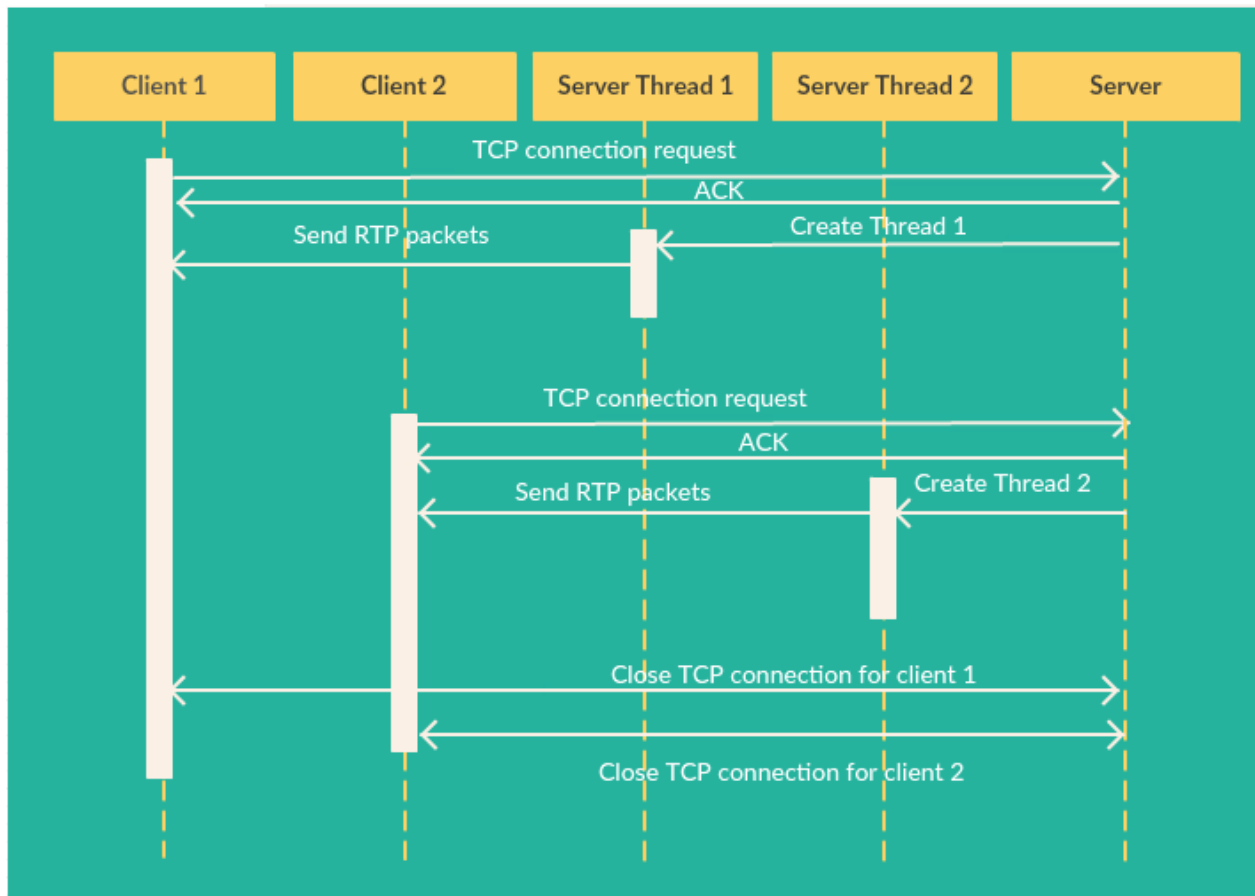


Figure 6: Scenario diagram to show multithreading for our server

Code snippets to explain the concurrency achieved

The code snippet below is used to create the thread which is done by extending the Thread subclass. A constructor of the class is created which accepts the number of clients(threads) to be created, the video name to be streamed and the socket of the client.

```

class threadOfClient extends Thread {
    static int i=0;
    static int imageInBytes = 0;
    static int typeOfMjpeg = 128;
    static int FramePeriod = 100;
    static int LengthOfVideo = 500;
    private Socket SocketOfClient = null;
    private StreamVideo video;

    private String clientName = null;
    private DataInputStream is = null;
    private PrintStream os = null;
    private final threadOfClient[] threads;
    private int maxClientsCount;

    public threadOfClient(Socket SocketOfClient, threadOfClient[] threads, StreamVideo a) {
        this.SocketOfClient = SocketOfClient;
        this.threads = threads;
        maxClientsCount = threads.length;
        video = a;
    }
}
  
```

After the threads are created, the start() method is called which calls the run() method. The run() method Packetizes the data and sends the RTP packets encapsulated in a UDP packet.

```
while (true) {
    int image_length = video.getNextFrame(buf);
    System.out.print("Image Length\n" + image_length);
    InetAddress address = SocketOfClient.getInetAddress();
    RTPPacketizer rtp_packet = new RTPPacketizer(typeOfMjpeg, imageInBytes, imageInBytes*FramePeriod, buf, buf.length );
    int packet_length = rtp_packet.getLength();
    byte[] packet_bits = new byte[packet_length];
    rtp_packet.getpacket(packet_bits);
    outPacket = new DatagramPacket(packet_bits, packet_length, address, PORT);
    DataSocket.send(outPacket);
    Thread.sleep(1000);
    System.out.println("Server sends packet of size : " + outPacket.getLength());
    System.out.println("Server sends packet of containing data : " + outPacket.getData()+"Port no"+outPacket.getPort());
    try {
        Thread.sleep(500);
    } catch (InterruptedException ie) {
```

Multicasting

Network communication through IP to a specific destination or multiple destinations happen by using different forms of addressing like unicast, Anycast, Multicast and broadcast. Unicast is transmitting a packet to a single destination while broadcast is sending a datagram to a subnetwork. Multicast is transmitting datagrams to a set of hosts which have been configured to a specific group which is not connection oriented. Multicasting requires datagram support such as UDP (best-effort reliability) or raw IP. Multicasting support is optional in IPv4 but mandatory in IPv6. In the destination field address, a group address is given unlike a single address in Unicast. The hosts in the group can join or leave the group any time using Internet Group Multicast Protocol (IGMP).

A special address of class D identify the multicast groups where class D addresses are in the range of 224.0.0.0 to 239.255.255.255. IGMP enables multicast routers to detect the multicast traffic and forward it using multicast IP forwarding tables. The IP Time To Live (TTL) field limits the extent of multicast. In Unicast, the video server has to send a separate video stream to the network for each client that has requested for the video which consumes more link bandwidth as the number of clients increase. They also consume a lot of bandwidth within the network. In a multicast environment, the video server transmits only a single video stream for each multicast group. The copies of the streams are created by the multicast routers and switches and sent to the clients or hosts configured to the multicast group.

Issues in multicasting:

Time to Live: Set a default value of 1 for outgoing multicast datagrams.

Loopback mode: A copy of each datagram will be looped back and processed. By default loopback is enabled.

Port Reuse

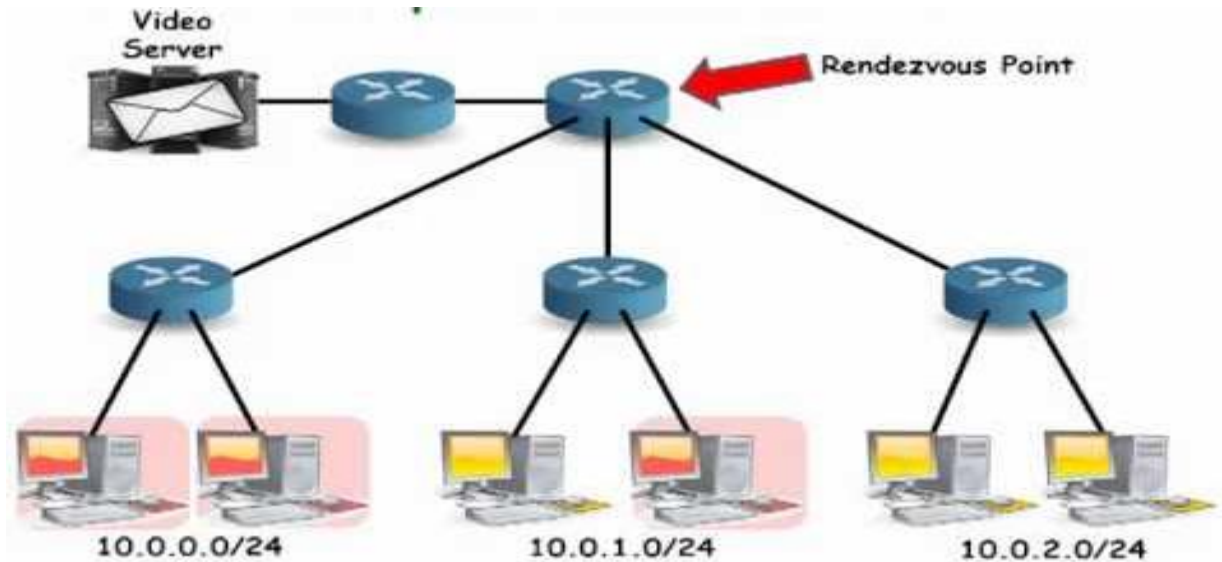


Figure 7: Working of Multicast Routers in Multicasting

Code snippets to explain the Multicasting

A multicast socket is created which is bound to a port 8888. All processes must bind to the same port in order to receive the multicast messages. We join a multicast address 224.2.2.3 by using the method `joinGroup()`. We use `receive()` or `receivefrom()` methods to read the messages. We can also use the same socket to send messages.

```
multicastSocket = new MulticastSocket(8888);
InetAddress address = InetAddress.getByName("224.2.2.3");
multicastSocket.joinGroup(address);
JFrame frame = new JFrame("Display Image");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = (JPanel) frame.getContentPane();
JLabel label = new JLabel();
while (true)
{
    inPacket = new DatagramPacket(inBuf, inBuf.length);
    multicastSocket.receive(inPacket);
    RTPDepacketizer rtp_packet = new RTPDepacketizer(inPacket.getData(), inPacket
    String msg = new String(inBuf, 0, inPacket.getLength());
```

Deployment of the Server and the Client

We chose to deploy the server and client in two ways:

- i. Creating a local network where the Server is running on one laptop and Client is running on multiple laptops.
- ii. Deploying the Server and Client on AWS EC2 instance in the Cloud. To connect the server instance from local computer, we assigned elastic IP address.

The below figure shows the running instance that was created on AWS to host the server script.

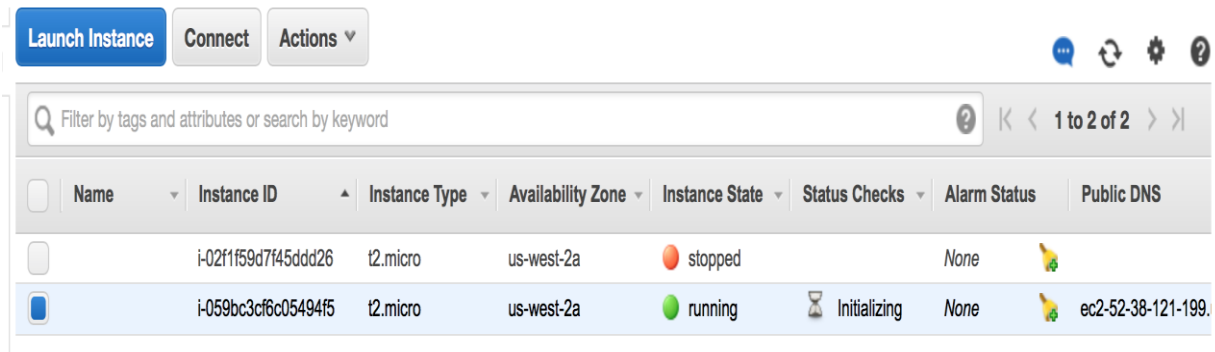


Figure 8: Showing the AWS instance running the server code

Connection was established to the AWS server using the below command on the terminal.

```
ssh -i "207.pem" ec2-user@ec2-52-38-121-199.us-west-2.compute.amazonaws.com
```

The connection establishment to the local machine is as shown below.

```
Last login: Sat May 7 14:26:27 on ttys000
Rohits-MacBook-Pro:~ rohitkotalwar$ cd Downloads
Rohits-MacBook-Pro:Downloads rohitkotalwar$ ssh -i "207.pem" ec2-user@ec2-52-38-121-199.us-west-2.compute.amazonaws.com
Last login: Fri May 6 23:54:31 2016 from 130.65.254.16

 _ _ _ _ _
 _ | ( _ /   Amazon Linux AMI
 _ \| _ _ _ _

https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/
[ec2-user@ip-172-31-21-42 ~]$
```

Figure 9: Showing the connection establishment to the local machine

The below figure represents the running instance of the server code on AWS and connecting the server from the localhost.

```
[ec2-user@ip-172-31-21-42 ~]$ ls
clientThread.class  HttpDownloadUtility.class  MultiThreadChatServerSync.class  StaticVideoStreamingServer.class
Frame1.class        HttpDownloadUtility.java    MultiThreadChatServerSync.java    StaticVideoStreamingServer.java
HttpDownloader.class movie.jpeg                  MultiThreadClient.java           VideoStream.class
HttpDownloader.java  MultiThreadChatClient1.class RTPpacket.class

[ec2-user@ip-172-31-21-42 ~]$ javac StaticVideoStreamingServer.java
[ec2-user@ip-172-31-21-42 ~]$ java StaticVideoStreamingServer
Enter the port Number:
8787
StaticVideoStreamingServer
Port Number Used=8787
```

Figure 10: Running instance of Server

The below figure shows the packet transfer from the server code running on AWS to the client.

```

[ec2-user@ip-172-31-21-42 ~]$ ls
clientThread.class  HttpDownloadUtility.class  MultiThreadChatServerSync.class  StaticVideoStreamingServer.class
Frame1.class        HttpDownloadUtility.java    MultiThreadChatServerSync.java    StaticVideoStreamingServer.java
HttpDownloader.class  movie.jpeg                  MultiThreadClient.java            VideoStream.class
HttpDownloader.java  MultiThreadChatClient1.class  RTPpacket.class
[ec2-user@ip-172-31-21-42 ~]$ javac StaticVideoStreamingServer.java
[ec2-user@ip-172-31-21-42 ~]$ java StaticVideoStreamingServer
Enter the port Number:
8787
StaticVideoStreamingServer
Port Number Used=8787
Client socket 33808
Client socket /50.141.33.9
New Connection accepted
In Thread: 1
Sending the data to the client:/50.141.33.9
Sending the data to the client:33808
Server sends packet of size : 65012
Server sends packet of containing data : [B@5d400f33Port no33808
Sending the data to the client:/50.141.33.9
Sending the data to the client:33808
Server sends packet of size : 65012
Server sends packet of containing data : [B@18a60d19Port no33808
it portNumber = Integer.parseInt(br.readLine());
System.out.println("Enter the server Ip address: ");
String host = br.readLine();

```

Figure 11: packet transfer from the server code running on AWS to the client

Execution Snapshots

On running the server, the user is prompted to choose Multicasting or Unicasting. We can choose any of the options. For example, Unicasting was chosen, as shown in the figure below.

```

import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.System.out;

public class StaticVideoStreamingServer{

    static int imageInBytes = 0;
    static int typeOfMjpeg = 26;
    static int FramePeriod = 100;
    static int LengthOfVideo = 1000;

    Timer timer;
    private static ServerSocket SocketOfServer = null;
    private static Socket SocketOfClient[] = new Socket[100];
    private static final int maxClientsCount = 10;
    private static final threadOfClient[] threads = new threadOfClient[maxClientsCount];

    public static void main(String args[]) throws Exception{

```

Figure 12: Client is prompted to choose Multicasting or Unicasting

Once the server is running, the client is prompted to connect to a port number.

```

Problems Javadoc Declaration Console
StaticVideoStreamingServer (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_91.jdk/Contents/Hon
1. Unicasting Using Multithreading
2. Multicasting
Enter the choice
1
Display name: awd10
Name: awd10
InetAddress: /fe80:0:0:0:cc6a:67ff:fe12:1de9%awd10

Display name: en0
Name: en0
InetAddress: /fe80:0:0:0:f65c:89ff:fea3:4bc1%en0
InetAddress: /10.250.170.10

Display name: lo0
Name: lo0
InetAddress: /fe80:0:0:0:0:0:0:1%lo0
InetAddress: /0:0:0:0:0:0:0:1
InetAddress: /127.0.0.1

Enter the Port Number
8888
Usage: java StaticVideoStreamingServer <portNumber>
Now using port number=8888
  
```

Figure 13: The client is prompted to connect to a port number

Once the port number is specified, the client gets connected to the port number and the server sends the video packets. The below figure shows how the server handles multiple clients.

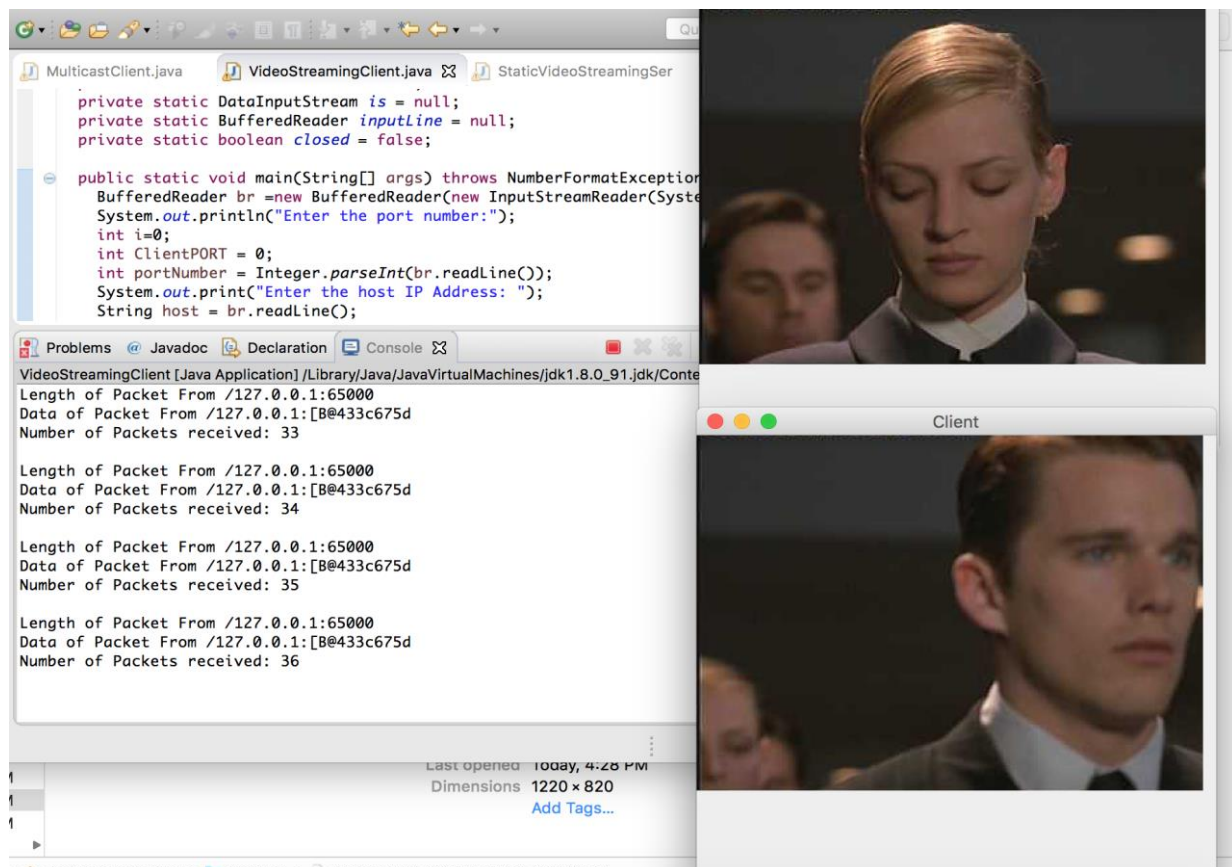


Figure 14: Server handling multiple clients on localhost

The figure below shows the Video streaming on a remote host. The Host is a Windows machine and the Client is a Mac machine.

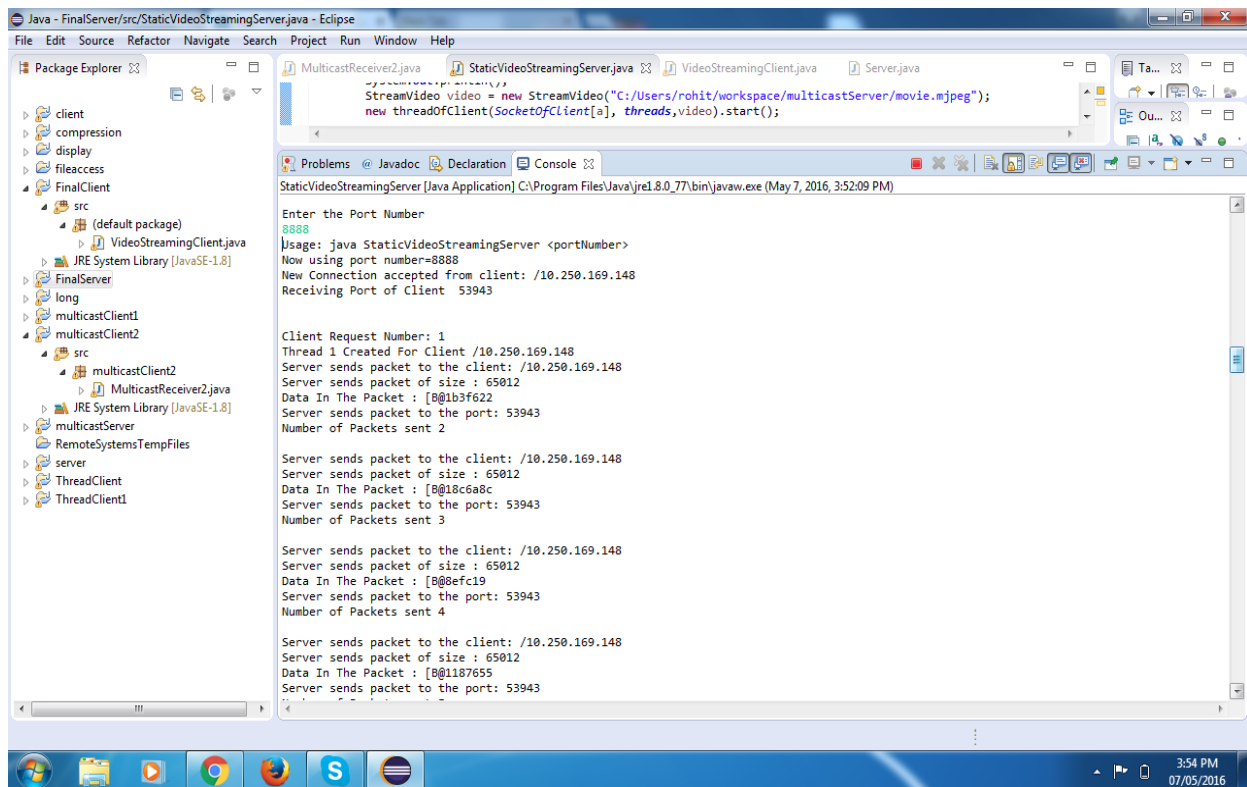


Figure 15: Server Running on Windows

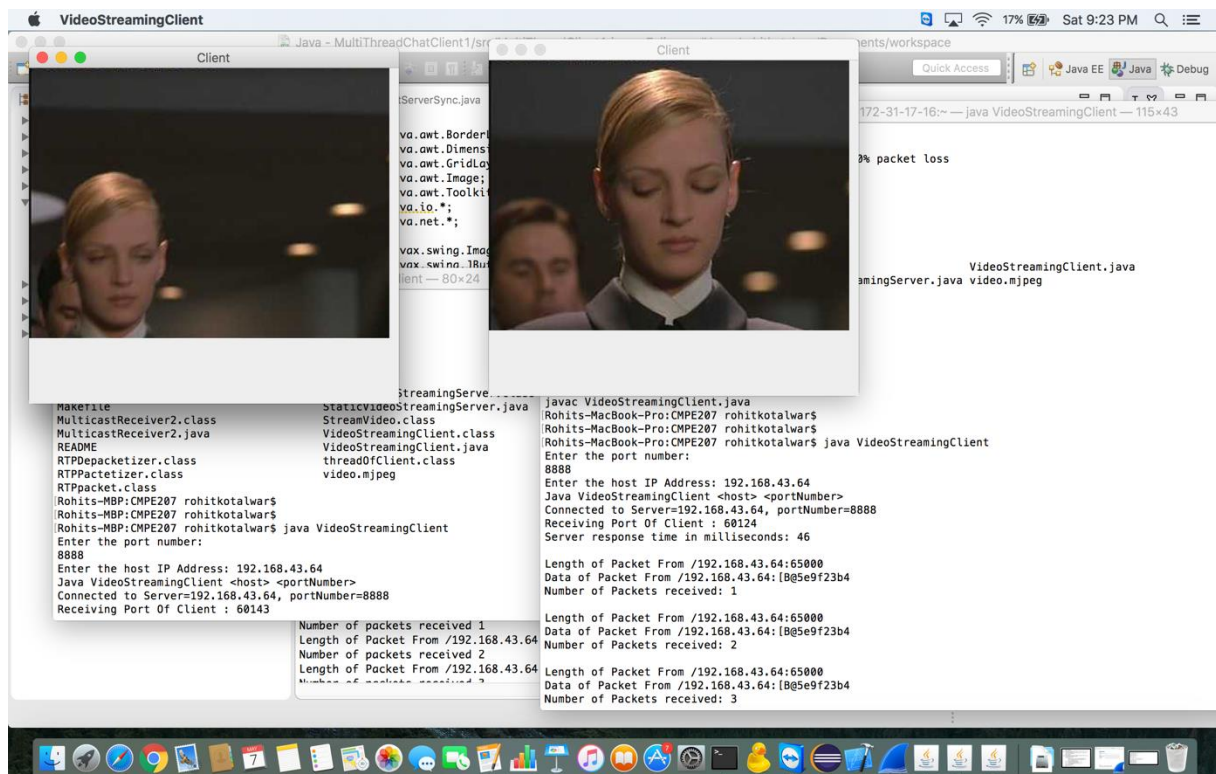


Figure 16: Client running on Mac

Test Cases

Test case 1: Testing the unicast streaming sequence of video for multiple clients:

Initially, we tested the functionality by streaming the video for a single client. Then we tested the functionality of the project by spawning multiple clients and each client would stream the video individually without effecting the stream sequence of other clients. The test case passed.

Test case 2: Statistics on Response time:

Response time is defined as the total amount of time taken for the server to respond to the client to establish the connection and start transmitting the frames.

We calculated the response time by spawning a number of clients. We spawned 20 clients and we observed the response time to be 2919.8 ms.

Client Number	Server Response Time in Milliseconds
1	170
2	4769
3	162
4	105
5	3420
6	55
7	3667
8	7502
9	11925
10	9134
11	141
12	216
13	1147
14	546
15	1306
16	328
17	783
18	12135
19	575
20	310
Average response time in milliseconds	2919.8 ms

Table 4: Response Time in Milliseconds

Test case 3: Statistics for Packet Drop:

- The number of packets sent from the sever and received by the client is compared.
No Packet Drops.
- Reliable communication.

Test case 4: Currency testing using JMeter:

No of Clients	No of Requests Accepted	No of Requests dropped
10	10	0
20	20	0
50	50	0
100	100	0
150	126	24

Table 5: Maximum number of clients handled

Challenges faced and challenges solved

Challenges faced:

- When we deployed the server in AWS, client could connect successfully to the remote sever. The server was able to send the packets. However, AWS EC2 instance has NAT issues due to which the client was not able to receive the RTP packets.
- It is observed that many Wi-Fi networks has firewall settings which blocks the packet transfers between server and clients.

Challenges Solved:

- To verify the connectivity between clients and server, including media packet transfer, we have used Wi-Fi Networks that has these firewall settings disabled. For example, SJSU_mydevices.

Network Wifi's	Firewall	Connection	Packet transfer
SJSU_premier	enabled	yes	no
SJSU_Guest	enabled	yes	no
SJSU_mydevices	disabled	yes	yes
HomeNetwork	disabled	yes	yes

Table 6: Testing for Challenges

References

- "International Journal of Machine Learning and Computing, Vol. 2, No. 6", 2012. [Online]. Available: <http://www.ijmlc.org/papers/252-L30067.pdf>. [Accessed: 06- May- 2016].
- "Introduction to IP Multicast Routing", 2016. [Online]. Available: <http://www4.ncsu.edu/~rhee/clas/csc495j/ip-multicast-part1.pdf>. [Accessed: 07- May- 2016].
- "Using Amazon EC2 Instances - AWS Command Line Interface", Docs.aws.amazon.com, 2016. [Online]. Available: <http://docs.aws.amazon.com/cli/latest/userguide/cli-ec2-launch.html>. [Accessed: 07- May- 2016].
- "Java Socket Programming Examples", 2016. [Online]. Available: <http://cs.lmu.edu/~ray/notes/javanetexamples/>. [Accessed: 20- Apr- 2016].
- "Java Thread Example – Extending Thread Class and Implementing Runnable Interface | JournalDev", Journaldev.com, 2016. [Online]. Available: <http://www.journaldev.com/1016/java-thread-example-extending-thread-class-and-implementing-runnable-interface>. [Accessed: 03- May- 2016].

Project Contribution

Class ID	Name	Project Deliverables
42	Ramya Nadella	Code, Test, Report, Demo
27	Rohit Kotalwar	Code, Test, Report, Demo
53	Pooja Prakashchand	Code, Test, Report, Demo
15	Rakesh Datta	Code, Test, Demo