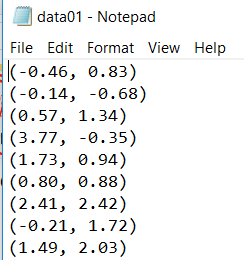
**ENTS 640 Networks and Protocols I**

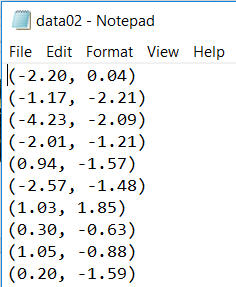
**Fall 2018**

**Project**

Problem Description and Solution:

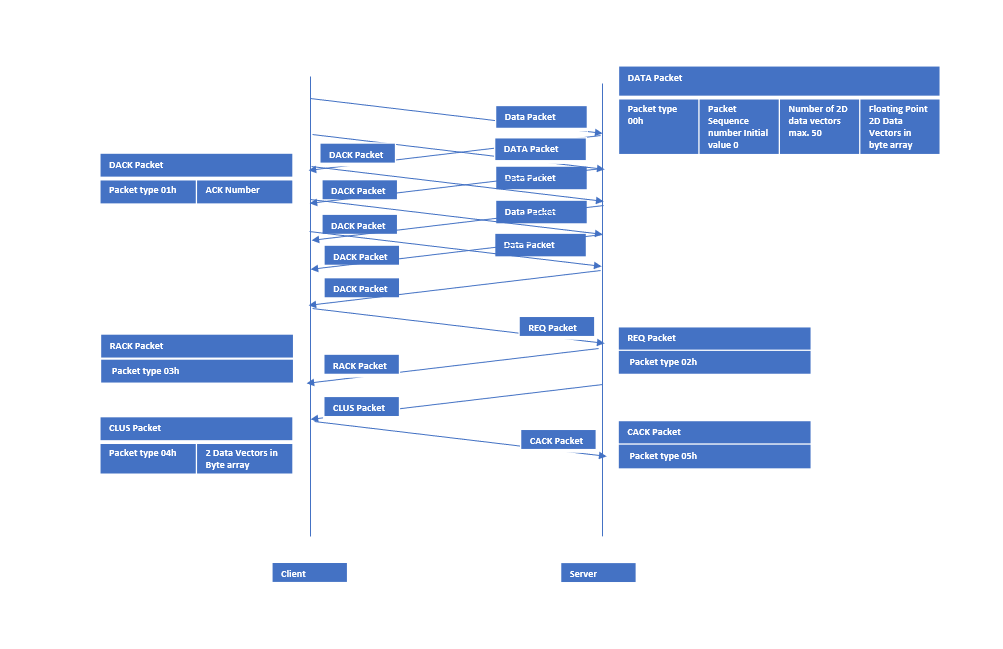
There are two Data files containing floating-point two dimensional data vectors. Data files contain one 2D data vector per line enclosed in parentheses, and the dimensions of the vector are separated by commas. The first value on each line is the 1st dimension, and the second value is the 2nd dimension of the 2D data vector. The data vectors are 2D real vectors, and their dimensions are floating-point values. The file data01.txt contains 243 2D vectors, while the data02.txt file contains 114 vectors.





Write a distributed networking application in Java consisting of a client and a server implementing reliable Data transfer over UDP packets. The client should read in the floating point 2D data vectors from the text files and store them in a data structure. Then, convert the floating point 2D data vectors into byte sequence. Prepare multiple UDP data packet with the header fields of packet type value, packet sequence number, total number of data vectors and the packet payload having the value of the data vectors in byte-encoded format. Send the packets to the server using UDP Socket and Datagram classes. The client should send these data packets and wait for server acknowledgements. Once, all data vectors are sent, clients should send a request to server for data computation. Server should receive the data in each packet, check for correctness of data by ensuring correct sequence number and packet type and hence send acknowledgement respectively. Server should acknowledge the computation request from the client. Server should convert the byte data into the floating point 2D data vectors and store it in an appropriate data structure. Server should perform K-means clustering in which initially server generates a random floating point 2D data vector called as centroid and later using it, server should partition the complete data received from the client into two clusters of floating point 2D data vectors and compute two 2D vectors which will be representing the calculated cluster centroids. Then the server should prepare a cluster information CLUS packet with packet type and the value of two centroid data vectors in bytes, send the CLUS packet to the client and wait for its acknowledgement. Client should reliably receive the CLUS packet, send its acknowledgement and convert the centroids 2D data vectors into floating point 2D data vectors and print on console.

Solution Design:



UML Diagram for Client and Server

|  |
| --- |
| Client |
| final int INITIAL\_SEQ\_NUM |
| -private static int numOfDataVectors(lnr: LineNumberReader): int  -private static float[][] rawDataArray(input: Scanner, numOfDataVectors: int) : float [][]  -private static IntToByteArray(data: int ): byte[]  -private static packetType(hexNum: int): byte[]  -private static twoBytePacket(num: int): byte[]  -private static printBytes(myByte; byte[]): void  -private static packetSender(vectorLen: int, dataVecStream: ByteArrayOutputStream, seqNum: int, ClientAddress: InetAddress, myClient: DatagramSocket): void  -private static extractPacketType(rcvData : byte[]):int |

|  |
| --- |
| Server |
| final int DATA\_PACKET  final int INITIAL\_SEQ\_NUM |
| +public static clusterCreateNewCentroid(m11: float,m12: float, fl: float[][],m21: float, m22: float, length: int): float[]  -private static extractPacketType(rcvData: byte[]): int  -private static byte[] packetType(hexNum: int): byte[]  +public static void print2darray(arr: float[][]): void  +public static printBytes(newbuf: byte[]) : void |

Solution Implementation:

The application is implemented in three phases using the below java libraries on both client and server.

Libraries for Client Code :-

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.io.LineNumberReader;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketTimeoutException;

import java.util.Arrays;

import java.util.Scanner;

import java.util.Timer;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

Libraries used for Server Code :-

import java.io.ByteArrayOutputStream;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketTimeoutException;

import java.net.UnknownHostException;

import java.util.Arrays;

import java.util.Random;

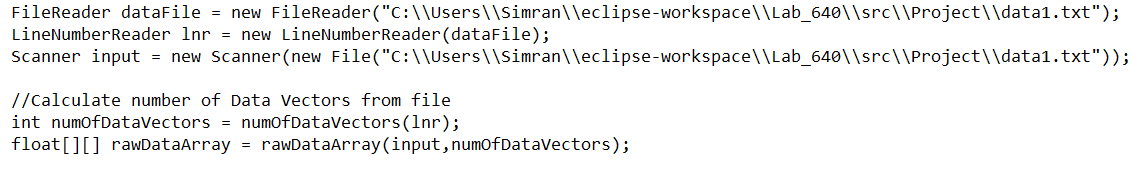
The three phases of application execution are below :-

1. The Data Vector Upload Phase
2. The Computation Request Phase
3. The Result download phase

The Data Vector Upload Phase :

The Client Side :

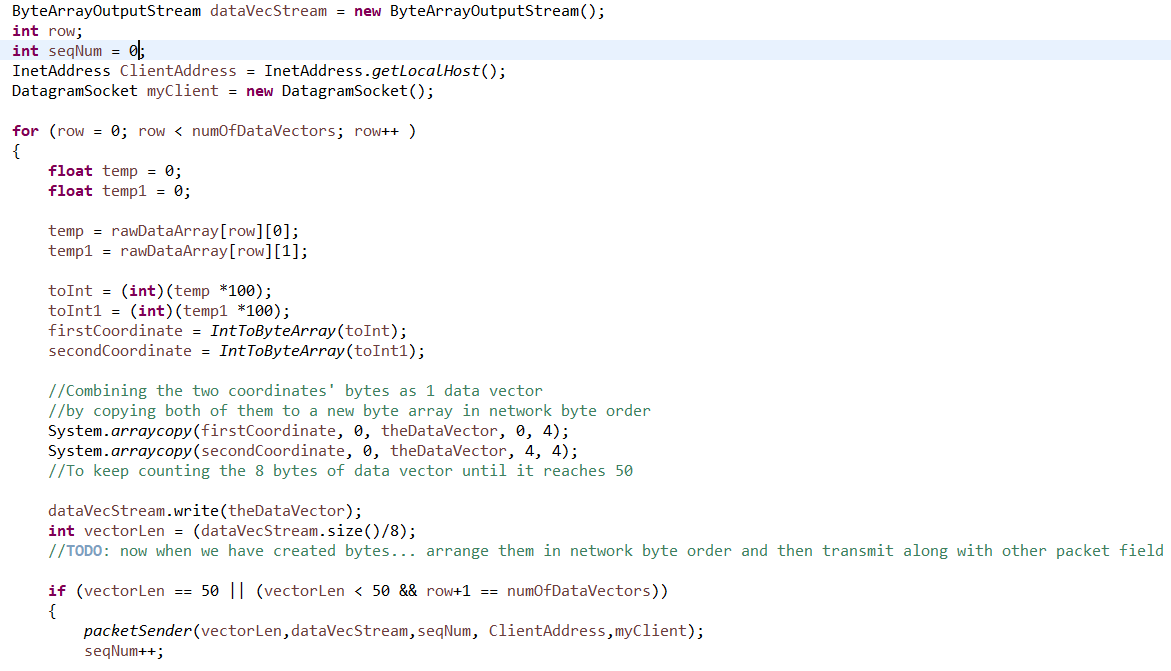
* The two text files containing two-dimensional (2D) data vectors are saved on the client disk path.
  + File Reader class is used to read in the contents of the data file and then total number of data vectors in the file are identified using LineNumberReader class on the datafile created.
  + Scanner input is used to upload the contents of the data file in string format in a file called input.

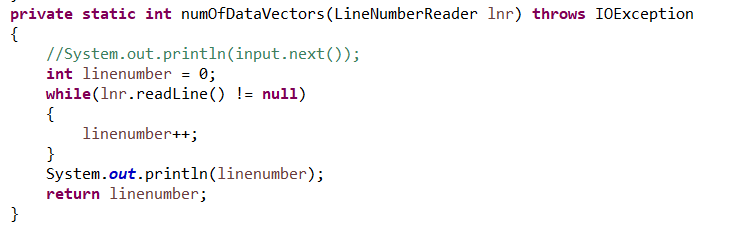


* A function rawDataArray is called to create a 2D Array of floating point 2D data vectors. Pattern class is used to define a regular expression and Matcher class is used find the pattern in the input file. Matcher class then used to save the group found as the element of 2D array.
* Here, 2D Arrays are used as the data file contains one 2D data vector per line enclosed in parentheses, and the dimensions of the vector are separated by commas. The first value on each line is the 1st dimension and the second value is the 2nd dimension of the 2D data vector.

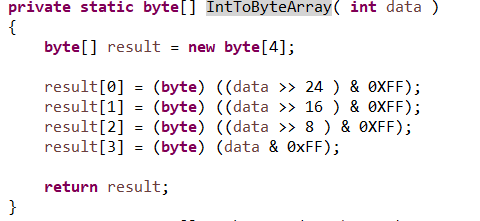


* ByteArrayOutputStream class is used to create a buffer “dataVecStream” of 2D data vectors in byte encoded format.
* 50 2D data vectors are byte-encoded and filled in the payload of the Client’s first data packet. Packetsender function is created to assemble a DATA packet to send to the server using UDP sockets Datagram packet and socket classes. Data packet contains Packet type, sequence number of the packet, number of data vectors sent in the packet and the byte-encoded data vectors array all transmitted in network byte order. That is, the most significant byte is transmitted first, the second most significant byte is transmitted next. The dimension is transmitted first and the second dimension is transmitted last.

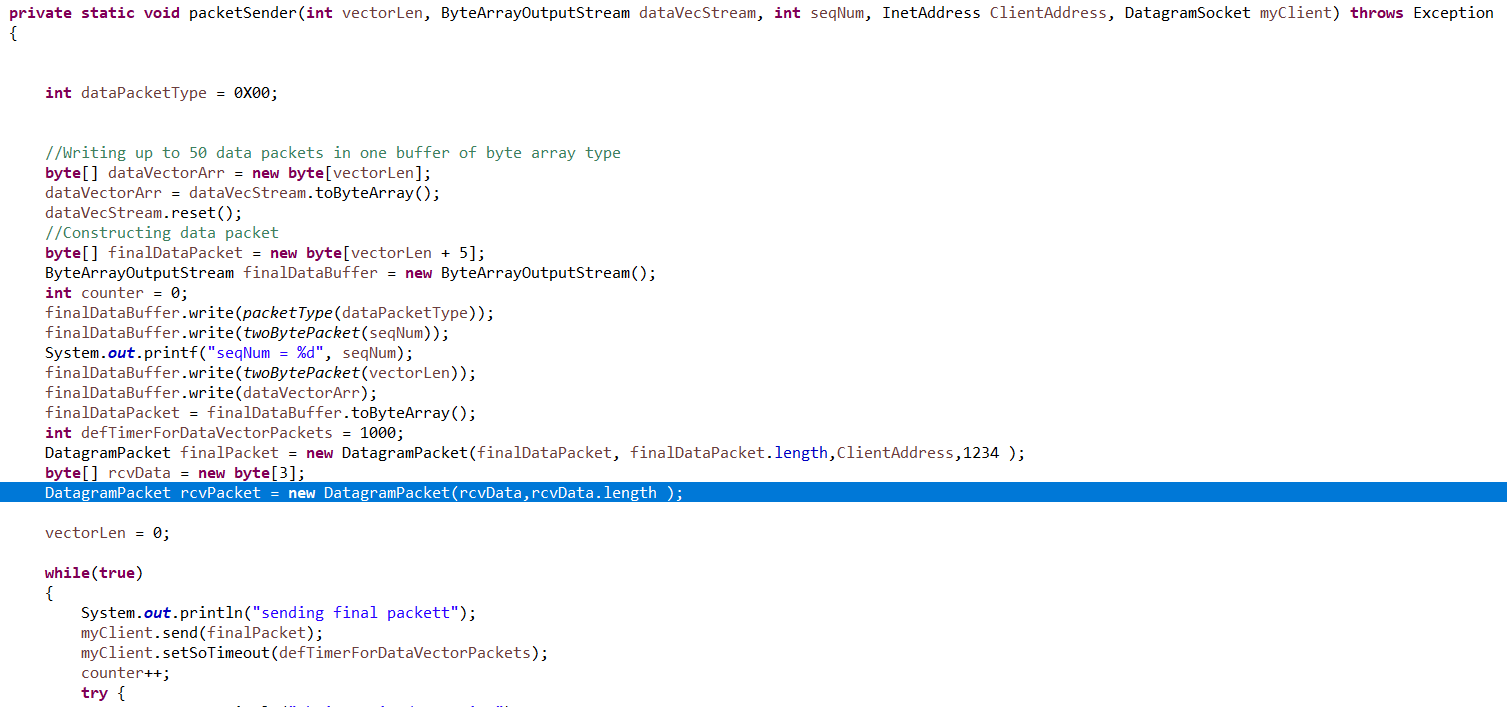


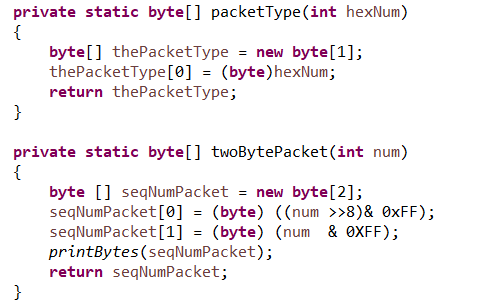
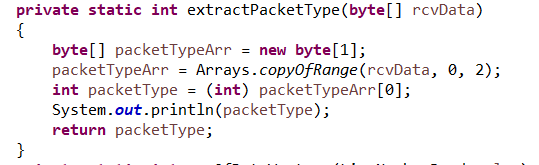


* A function IntToByteArray is used for byte conversion of the int value of each floating-point dimension of the 2D data vector.



* Client is initiating a timer for each data packet and expecting an DACK packet from server. If timer expires, Client writes a timeout error, doubles the timer and sends the Data packet again.
* Client checks each DACK packet if the sequence number received by the server is same as sent by the client and the packet data type is correct. If sequence number was different and server discard the sent Data packet. Client sends the expected sequence number Data packet again.



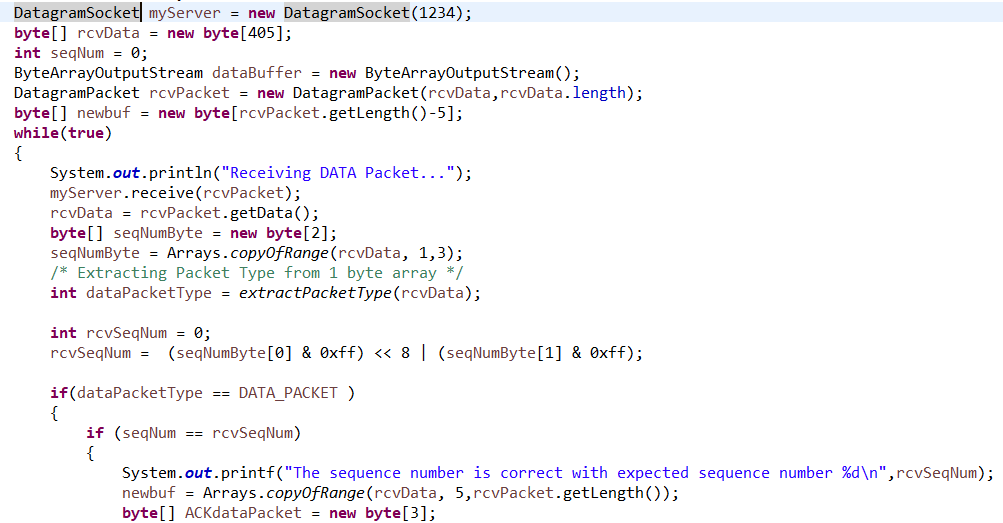
 

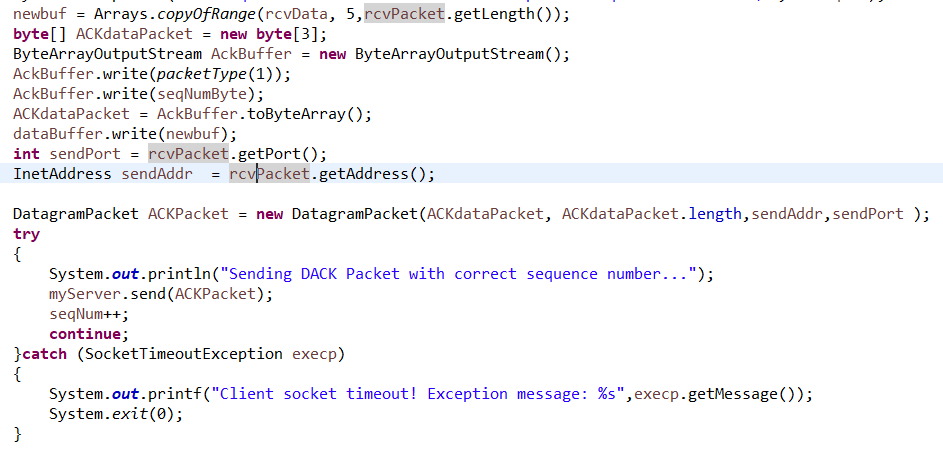
The Server Side :

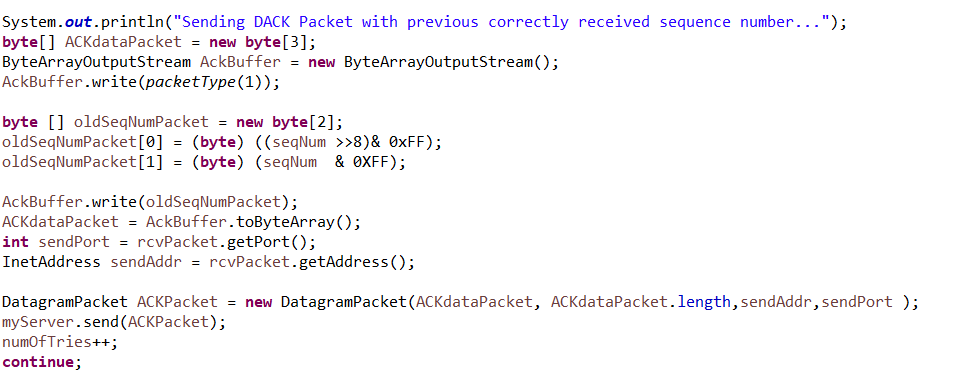
* The server receives the Data packet over the UDP socket, checks the correctness of the packet type and sequence number.
  + If correct, server extract the byte encoded data vectors from the received packet into a byte array “newbuf” using Arrays.copyOfRange java util method and then loads the data vectors into a byte buffer “dataBuffer” using ByteArrayOutputStream.
  + Once all the data packets are received the Server converts the data in the buffer to a byte array.

**byte**[] rcvDataVectorsArr = dataBuffer.toByteArray();

* + Server prepares DACK packet with the correct packet type and sends to the client.
  + If Sequence number is incorrect, Server ignores the packet received and sends an ACK packet with last correctly received DATA packet.
  + The code handles the socket timeout exception and all other exceptions during the application execution.



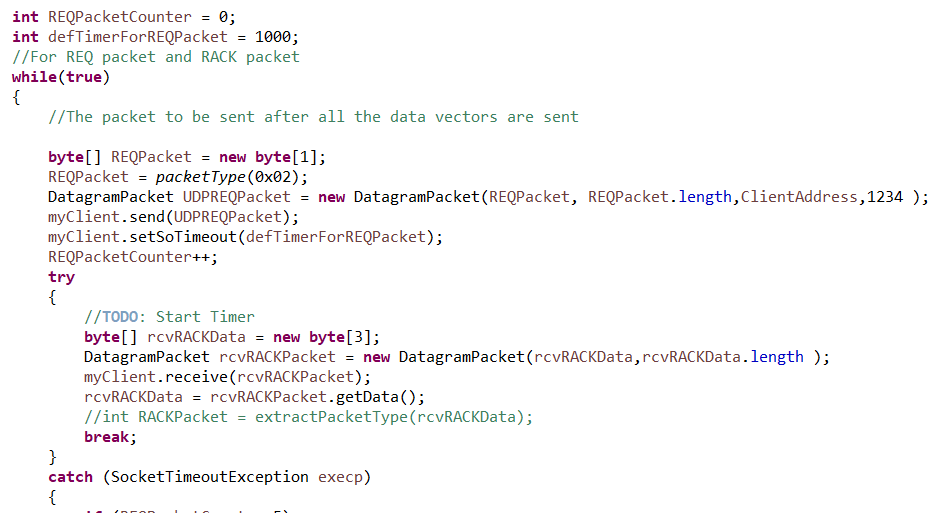




The Computation Phase :

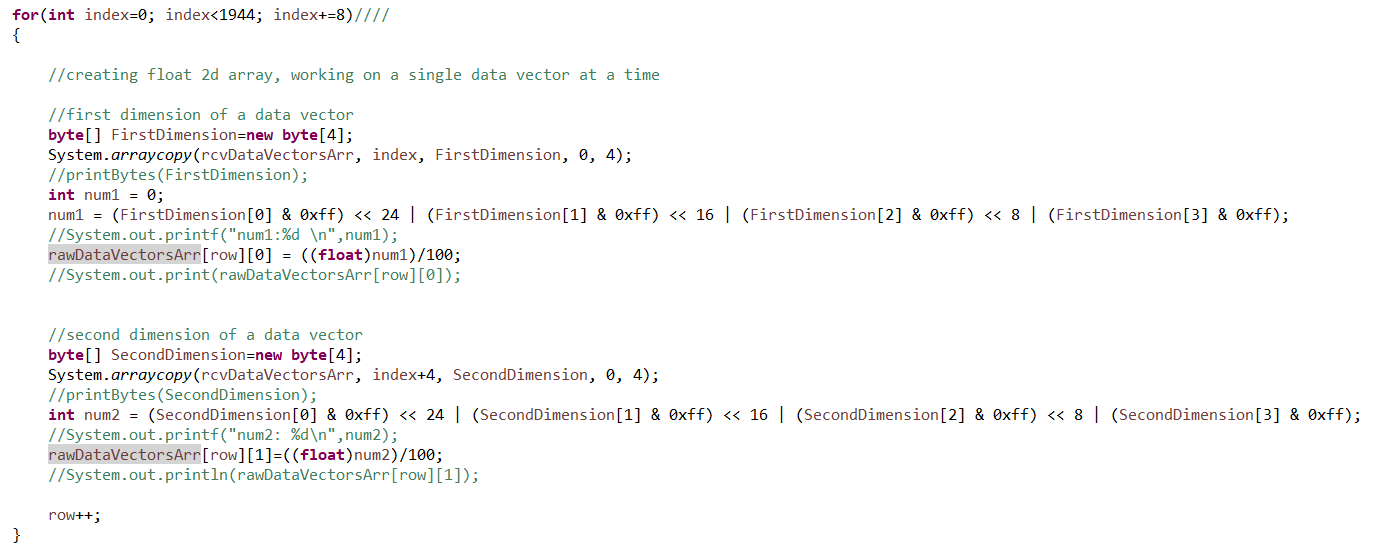
The Client Side:

* The client sends all the data vectors in groups of 50 as the max. number of data vectors to be sent in a data packet is given 50.
* The client sends a REQ packet to the server initiating a timer, REQ indicating all the data vectors are sent and server should begin the processing of the data. Client waits for the RACK acknowledgement from the server. Client checks RACK for correct packet type. If timer expires, timeout error is received, and packet retransmitted.
* For every timer in the application, the timer values double after each timeout and system throws an error of communication failure on receiving more than 4 timeouts.

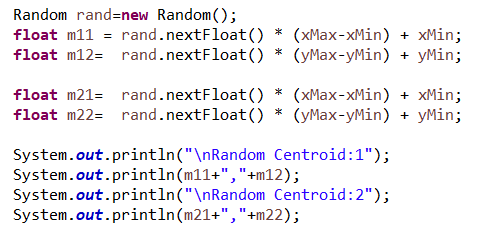


The Server Side:

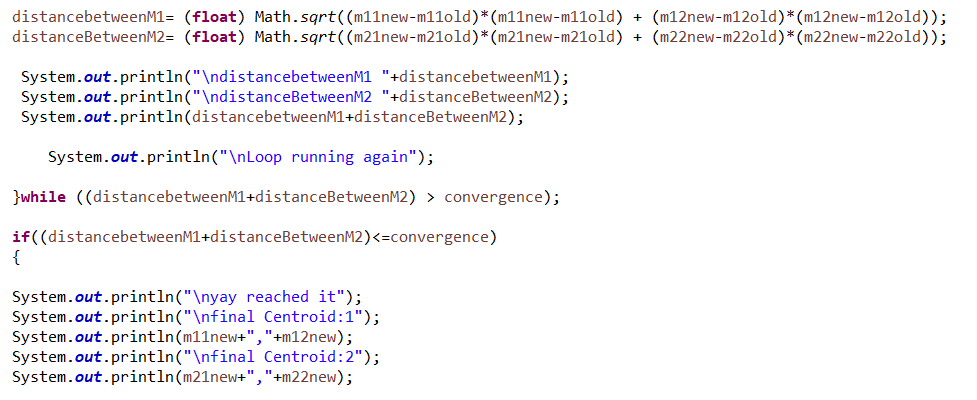
* The server receives the REQ packet, checks for packet type and prepares RACK packet.
* If RACK gets lost, server can receive multiple REQ packets. To handle such an event, server wait until a timer of 3 seconds expires, and if it does not receive any REQ packets during this time, it assumes that the RACK packet has been received by the client. Server begins the computation with the “rcvDataVectorsArr” the byte array of all the data vectors received from the client.
* The server extracts each data vector from the byte array, convert each dimension of each data vector into int and then floating-point co-ordinate. And creates a 2D Array “rawDataVectorsArr” of all the 2D data vectors received.

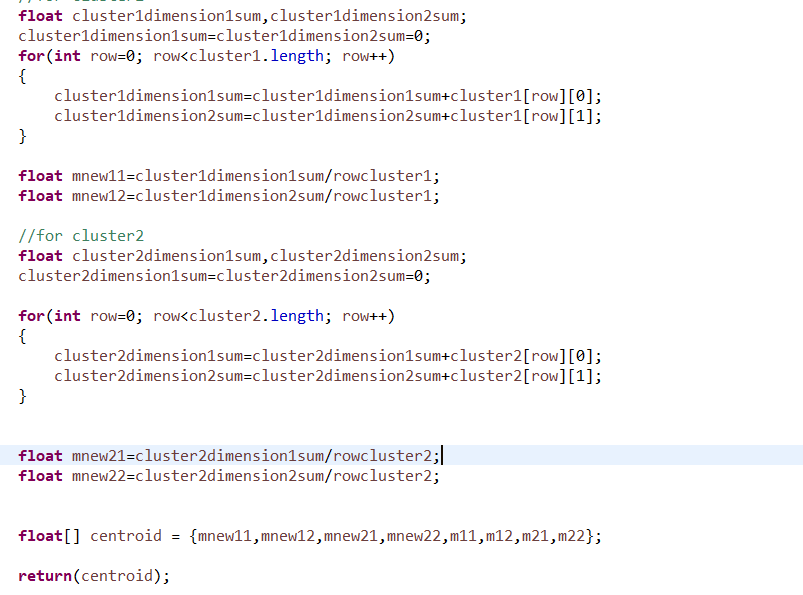
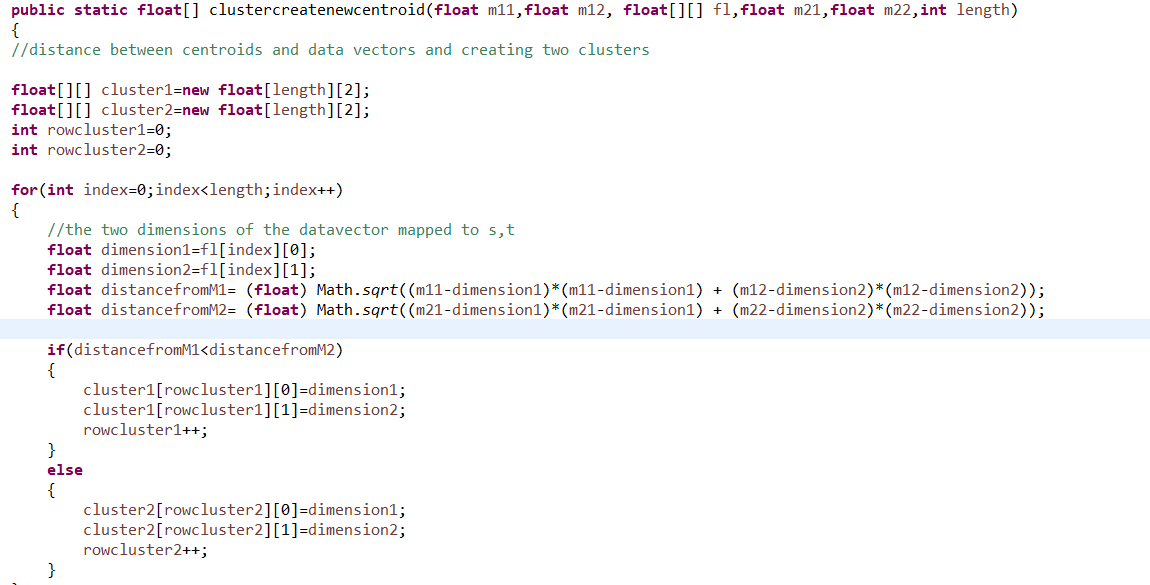


* Server performs the K-means Clustering Algorithm in the received data. Server generates two random centroids post analyzing the range of both the dimensions of the 2D data vectors.



* Server perform partition of the complete data into two clusters, decision based on the distance of each data vector from a randomly generated centroid data vector. A function “clustercreatenewcentroid” is defined for the purpose.
* Once clusters are available, the server calculates the new centroid of each cluster by taking average of all the data vectors in the specific cluster. The function returns the value of the new centroid calculated for both the clusters and also send the old centroids used in the calculation.
* Server checks the distance between the old and the new centroids. If the distance is more than the convergence constant 0.00001. Then again the server partition the entire data vectors data from the client into two new clusters based on the distance of each data vector from the new centroids. The Server repeats this process partitioning and new centroid calculation until the distance between the last two centroids calculated is minimal, actually less than the convergence constant.
* When the distance goes below 0.00001, then this centroid is considered the final centroid data vector. This way Server divides the complete data into two clusters and computes two centroid data vectors representing each cluster.

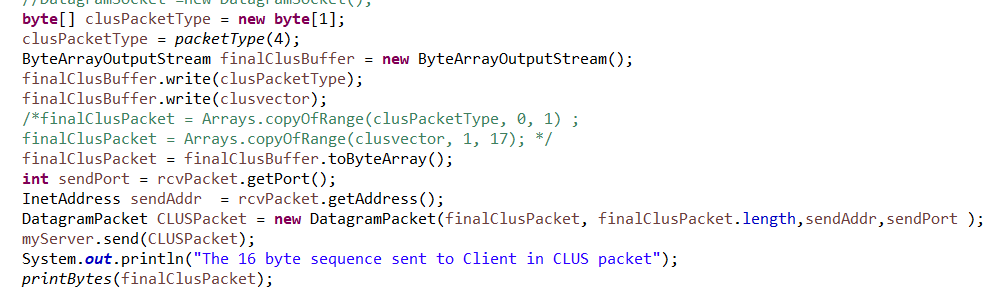


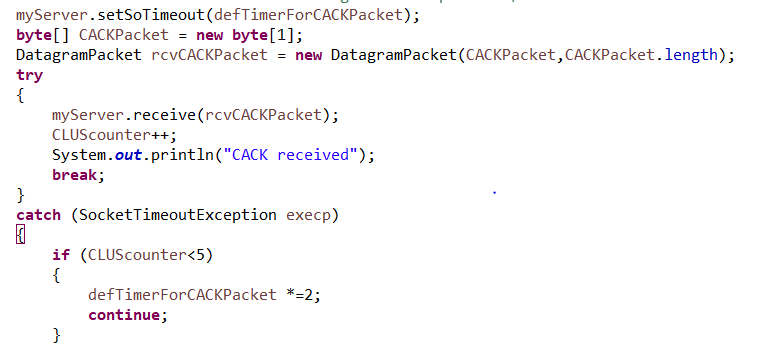
  


The Result Download Phase:

The Server side:

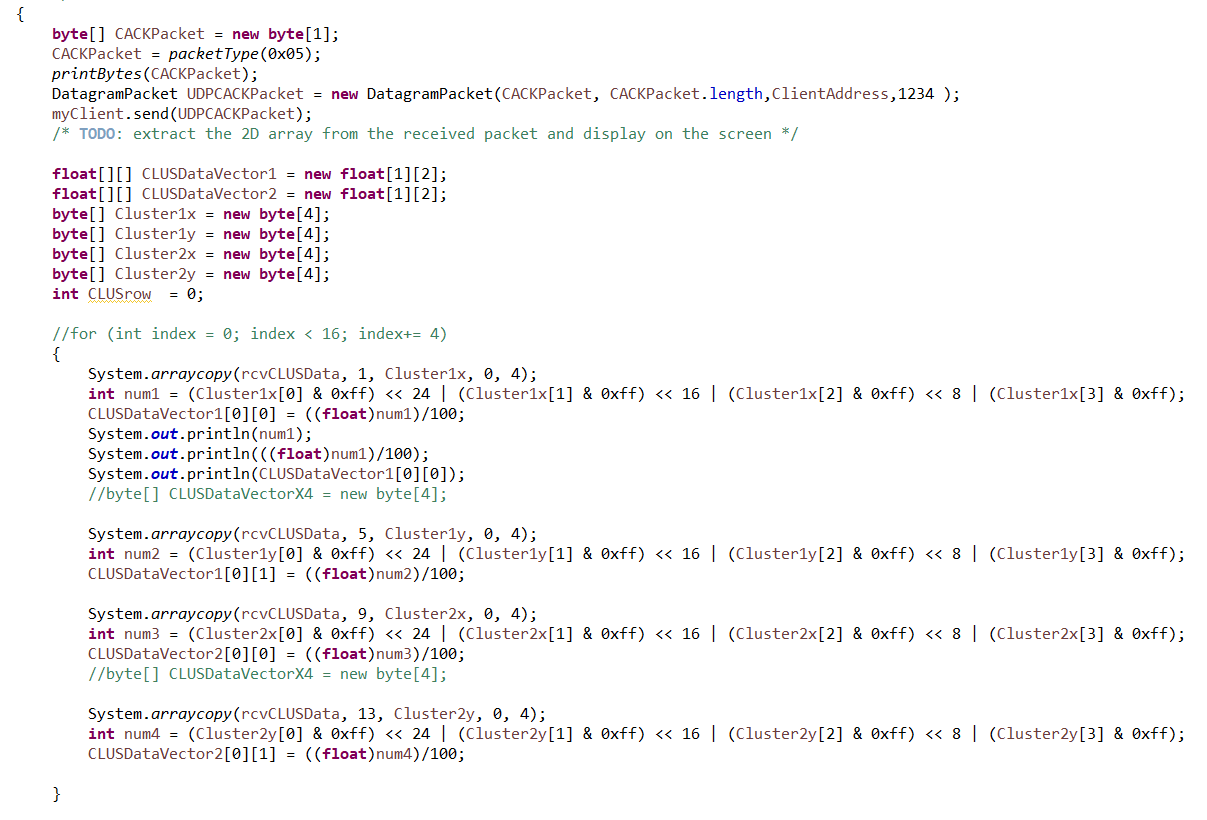
* Server stores the new centroids into a float array and then convert them into byte format for sending them to server.
* Server prepares a CLUS packet with the centroids in the byte-encoded format and the correct packet type.
* Server sends the CLUS packet, initiates a timer and waits for the acknowledgement from the client. Timeout is achieved if timer expires.





The Client Side:

* Client waits for 30 seconds after sending the REQ request if does not receive the CLUS packet, a server failure error is reported.
* Client receives the CLUS packet. Client sends an acknowledgement on receiving the final centroid data.
* Client also waits for any duplicate CLUS packets and sends CACK packet if it receives any.
* Client converts the received byte-encoded centroid vectors into two two-dimensional floating-point data vectors and prints them.



The Application Output for datafile1:

Client:

Client is sending Data values for K-Means Clustering with 243 data vectors

Receiving CLUS Packet

Sending CACK...

The first centroid is : (-1.18,-1.02)

The second centroid is : (1.1,1.08)

Existing System

Server:

Server is receiving data vectors...

Calculating Centroid by K-means Clustering Algorithm...

Sending CLUS Packet with calculated Centroids...

Starting timer to receive CACK Packet

CACK received

... Exiting System

The Application Output for datafile2

Client:

Client is sending Data values for K-Means Clustering with 114 data vectors

Receiving CLUS Packet

Sending CACK...

The first centroid is : (0.85,1.21)

The second centroid is : (-1.21,-1.2)

Existing System

Server:

Server is receving data vectors...

Calculating Centroid by K-means Clustering Algorithm...

Sending CLUS Packet with calculated Centroids...

Starting timer to receive CACK Packet

CACK received

... Exiting System

The Application Output for invalid packet type:

Client:

Client is sending Data values for K-Means Clustering with 114 data vectors

Invalid Packet Type

Invalid Packet Type

Client socket timeout! Exception message while trying to receive RACK packet: Receive timed out

The Application Output when timeout is reached:

Client is sending Data values for K-Means Clustering with 114 data vectors

Client socket timeout! Exception message while trying to receive RACK packet: Receive timed out

Server is receving data vectors...

Thanks !