Syed Uzair Ul Hassan
CS YEAR 3 (SEMESTER 6)
REG # 2018-UET-NML-CS-20
UB # 1802020
Email: suzair2018@namal.edu.pk

Designing a Protocol for Chat App

June 11, 2021

Abstract

We need to design a protocol for a chat application. Our chat application will use this protocol to provide the mechanism of communication between clients and server. Using this protocol, our application will transfer the message of one client to another with the help of a server. Total there will be tree main entities in our protocol, Client, co-server and main Server. Our protocol will follow Client-Server based pattern for the communication. There will be two operations which a client can perform using our chat application. One is text messaging to other user and second is voice messaging to other user.

Table of Contents

Expected Outcomes of Protocol:

Some capabilities which we are expecting from the protocol are:

- Protocol should be cost efficient.
- Protocol should not be too much complex.
- Protocol should be easy in handling.
- Protocol should be able to manage work load.
- Protocol should instantly establish connection between client and server when needed.

Application Requirements:

Requirements of our chat application are mainly categories into two types. These two types are as follows:

Functional Requirements:

- App should allow the user to create personal account.
- App should have proper mechanism for the privacy of clients.
- App should be able to save and manage client's contact numbers.
- App should allow user to communicate with other users via text messages or audio messages.
- App should allow the user to save the chat history.
- If the receiver side is offline, app should make it sure that the message is delivered when receiver side comes online.
- App should allow the user to block a particular sender, if necessary.
- If the receiver side is online, the message should be delivered instantly.
- App should prevent the loss of data.
- If a client sends a message to multiple receivers, message should be delivered to all receiver instantly and at same time.

Non-Functional Requirements:

- App should be able to change its UI according to the device in which it is running.
- App should make it sure that the message should be delivered in the same manner as it was sent.
- User can schedule messages.
- There should be a chat back-up mechanism in the app.
- App should be android base as well as web application.

Application Structure:

Chat application will use a client-server based mechanism. There will be a total of 3 entities in our application structure. One is the client, second is the co-server and third is the main server. There will be a co-server assigned to every user. Means that every user will have its own co-server. These co-servers will always remain connected to our main server.

Each co-server assigned to the user will have the port number, phone number and IP address of the user. When the client wants to send a message, the message will first send to its assigned co-server, the co-server will encode the message and will transfer the encoded message and receiver's information to the main server.

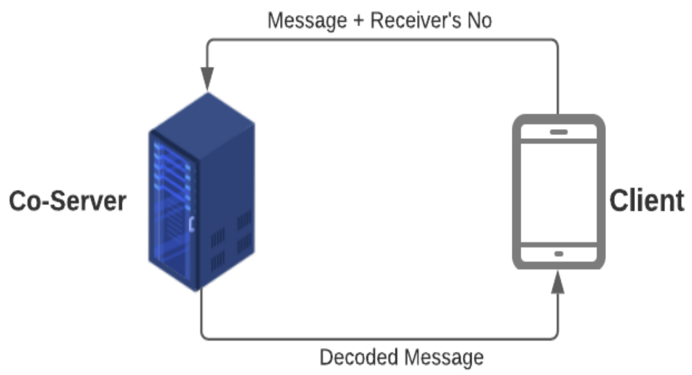

Figure 1: Client and Co-Server

The main server will have the information of all co-servers, stored in data base already, assigned to every client. When the main server will get an encoded message and receivers information from any of the co-server, it will directly transfer the message to the receiver's co-server as the main server has all the information of the co-servers and the co-servers are already in a connected state with the main server. When the main server will transfer the encoded message to the receiver's co-server, the receiver's co-server will decode the message and will transfer to the receiver's device, when the receiver will come on-line.

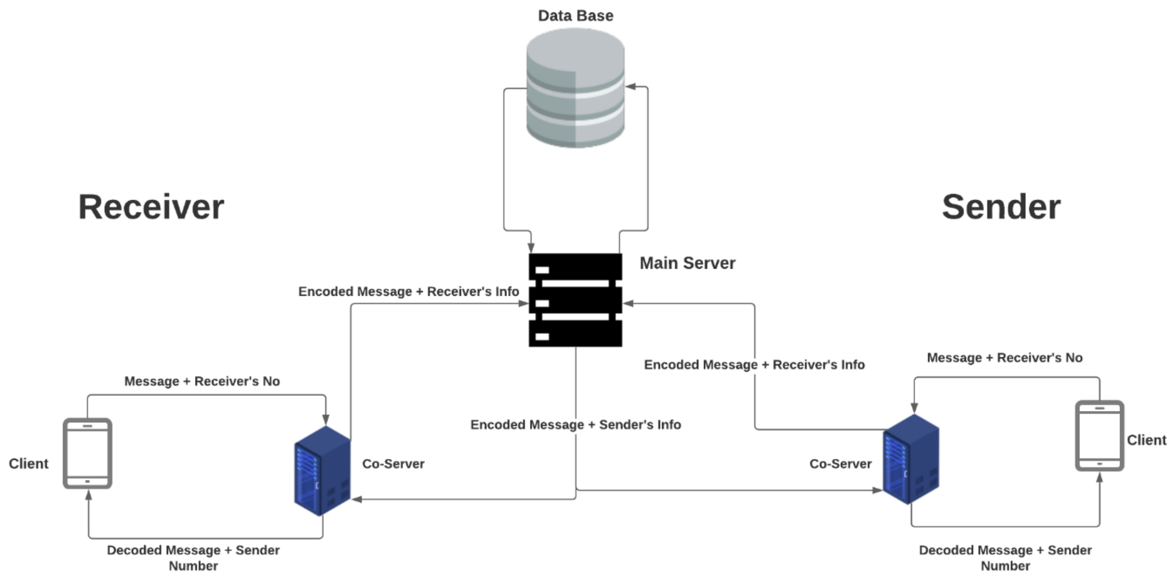
Data Base
Receiver
Sender
Main Server
Encoded Message + Receiver's Info
Encoded Message + Receiver's Info
Message + Receiver's No
Message + Receiver's No
Encoded Message + Sender's Info
Client
Co-Server
Co-Server
Client
Decoded Message + Sender Number
Decoded Message + Sender Number

Figure 2: Overall Mechanism

The Protocol:

A network protocol is a way or set of rules to communicate between different devices in the same network. It allows connected devices to send and receive useful information. A Network Protocol is mainly composed of different components as explained below:

Category of Protocol:

Protocols are mainly divided into two categories:

- **Standard Protocol:**
  A standard protocol is a protocol which is built by a team of experts and it is not restricted to any particular kind of device. For example DNS, FTP etc.

- **Proprietary Protocol:**
  These protocols are built by some specific organization for their own use only. These type of protocols can only be used in some particular kind of devices. For example Apple design its own protocol for its applications.

We will build a standard protocol because our chat application should be feasible to run on any device, having an Android OS or Windows or iOS or Linux. We will not restrict our protocol for some particular kind of devices.

Level of Protocol:

There are three levels at which we can implement a protocol:

- Application Level
- Hardware Level
- Software Level

We will implement our protocol at application level as we need protocol for the communication purpose of our chat Application.

Functions of Protocol:

Some major functions which our protocol of chat application will perform are:

- Flow of data
- Connection establishment
- Error Detection
- Sequencing of data etc.

Key Elements of Protocol:

There are four key elements of a Protocol as:

- Message Type
- Message Format
- Message Syntax
- Message Semantics

The working structure of a protocol depends on these key elements of a protocol.

- ### Message Type:

  There are three main types of messages in a protocol. These are:

  1. Data Messages
  2. Command Messages
  3. Control Messages

  In our protocol we will use data messages and command messages both. Client will use command messages to request the co-server for the establishment of connection when the client comes online.

  Client will also use data messages to get received messages from its co-server and will also use data messages to push data towards the co-server, when he wants to send a message to another client. Similarly, co-server will use data messages to send the encoded messages towards the main server and to receive the encode messages from the main server, sent by another client. Main server will use data messages to receive encode messages from the sender's co-server and will use data messages to send the encoded messages towards the co-server of receiver.

- ### Message Format:

  There are two types of message format. These are:

  1. Text Oriented
  2. Binary Messages

  Our protocol will use both formats. In every data package, whether it is used for connection establishment, send by client to co-server or the package send by co-server to main server, header of the package will be text oriented but the body of the package will be binary messages due to the safety and security purposes except connection request message also known as command messages. Command messages will be completely text oriented. Because in command messages there is no privacy issue and we just need to establish connection.

Text oriented messages are close to human and easy to understand but difficult for the machine and to code. That's why we are just using this format in our header section. Binary format is difficult to understand but it is easier for the machine to code. So we will use this format for our body of data packages.

- ## Message Syntax:

The client will send its co-server a mini packet of data in which there will be two components. First component will be the header in which there will be information about the type of message, whether the message is command type for connection or data type for sending/requesting data and second component will be the body in which there will be information if the client wants to establish connection or the message which the user wants to send or the request to get the received messages, depending on the type of message. This mini data packet will be of size 8 bits. 2 bits for header and 6 bits for body

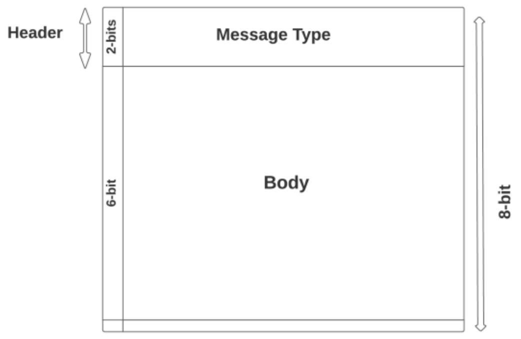

Figure 3: Syntax of mini Data Package

Our main server will receive the data from co-servers in the form of packets. Each packet will be of 32 bits. Packet will be distributed into two main components. First component will be the header and the second component will be the body. In the header, there will be 3 divisions. Each division will be of 4 bits. First 4 bits will include the sender's information, next 4-bits will include the receiver's information and remaining 4-bits will be the checksum bits.

Second component will be the body. This component will be of size 20-bits and it will includes the message of the sender.

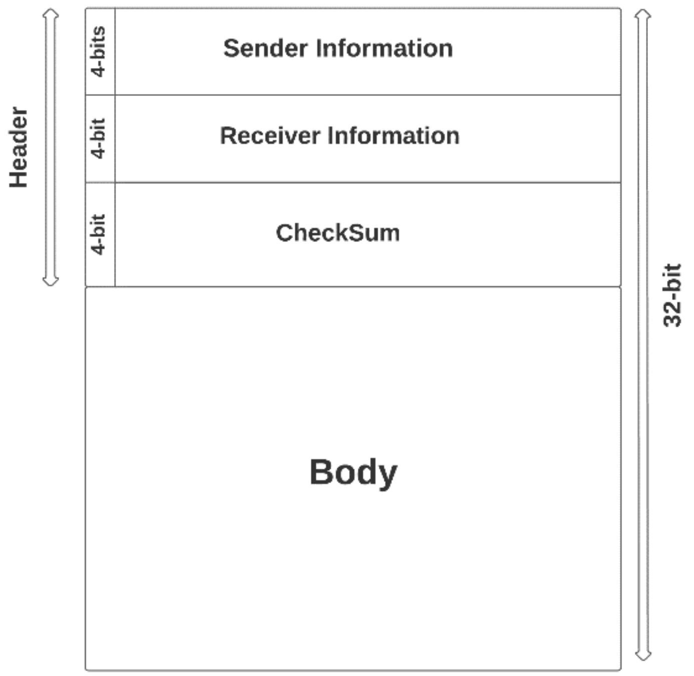

Figure 4: Structure of Data Packet

- ## Message Semantics:

Our co-server will be the part of main server. Actually a sub part of main server. These two entities of our protocol will always remain connected with each other and will be always in a state of listening. When a user comes online, its assigned co-server will automatically establish connection with him. When the user will come online, first he will request pending messages from its assigned co-server automatically. And if a user wants to send a message to another user, he will sent request, including the message and receivers information, to its assigned co-server. Then its assigned co-server will encode the message, will create a package and will send the package to the main server.

The main server will then read the package and will transfer the encoded message and sender's information to the receiver's co-server. The receiver's co-server will then decode the message and will wait for the receiver to come online.

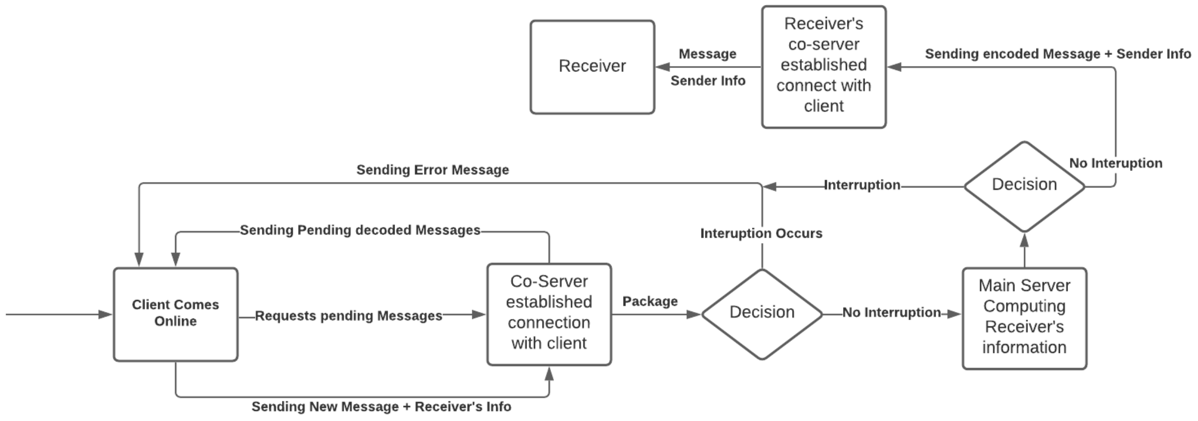
Receiver
Message
Sender Info
Receiver's co-server established connect with client
Sending encoded Message + Sender Info
Sending Error Message
Interruption
Decision
No Interuption
Sending Pending decoded Messages
Interuption Occurs
Client Comes Online
Requests pending Messages
Co-Server established connection with client
Package
Decision
No Interruption
Main Server Computing Receiver's information
Sending New Message + Receiver's Info

Figure 5: How the protocol will work

If the user exceeds the limit of message content or there is something missing in the data which the co-server receives from the user, the co-server will automatically generate an error and will send it back to the user. Similarly if there is any issue at the co-server end or at the main server, the co-server will automatically send an error message back to the user.

Additional Information for Developer:

- Try to avoid any kind of distraction, work in peaceful environment.
- Try to use python programing language for the implementation of the protocol.
- Try to use Visual Studio code to do code Implementation.
- Try to do implementation in sub-parts.
- Try to use break points while debugging the code.
- Try to do small check tests on every part of the code.
- Try to add comments in the code for better readability.
- Try to split code in different files and then import that files in our main file.
- If you feel any confusion, please feel free to discuss with your supervisor.