**Project Abstract**

**Title**

Peer to Peer ChatSDK with distributed hash tables

**Student(s)**

Benedikt Kaiser BSc.

**Project Advisor**

Alexander Palmanshofer MSc.

**Date and Time**

Hagenberg, 07.07.2023

Version: 09/2022

# Motivation

My master project will be an open source ChatSDK for iOS that can be easily integrated into an existing or new iOS app via the Swift Package Manager.

To properly understand a decentralized ChatSDK, one must first understand how a centralized ChatSDK works.

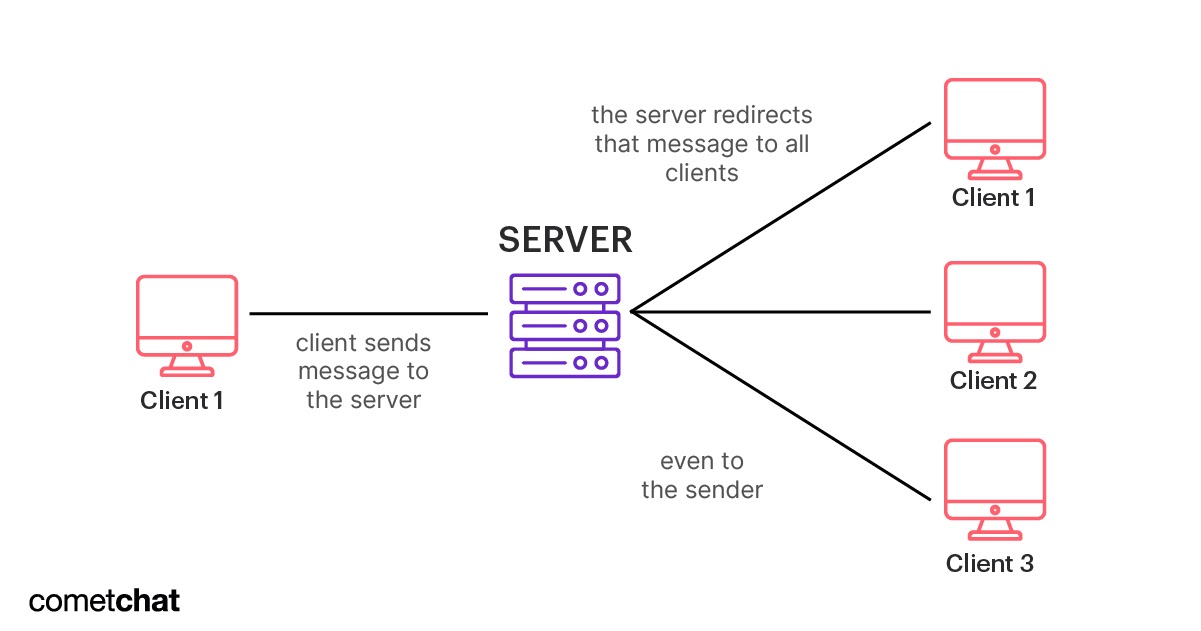


Figure 1 https://www.cometchat.com/blog/chat-application-architecture-and-system-design

Here in Figure 1 you can see that each communication runs via a server. The advantage here is that the server always has the same IP address and can be reached at any time. Client 1 only has to send the message addressed to client 3 to the server and the server informs client 3 with a push notification that it can retrieve a new message.

However, the drawback to this communication is that the server can read and store any message that it transmits. Furthermore, the entire network can be paralyzed in the event of a server failure.

In a distributed Peer to Peer network there is no such single point of failure or entity that could read data. This is possible due to storing the individual ip addresses in a distributed hashtable.

# Realization

This work is about setting up such a network of smartphones. The IP address of a user can change at any time. Thus, a system must be built that stores all IP addresses in a distributed manner, keeps them up to date, is easily expandable and efficient. For this, the already existing protocol Kademlia comes into play.

Kademlia is a protocol to map IP addresses to their corresponding clients using distributed hash tables (DHT).

Each client has stored a part of the hash table as a routing table itself and can therefore look into its routing table when a new message is received. If it does not have the necessary remote station in its own routing table, the client sends a request to the node in its routing table whose hash is "most similar" or the node with the smallest "distance" to the desired recipient. The latter repeats the procedure until the receiver is found and sent to the original sender.

However, the distance is not a physical proximity but is usually determined with an XOR link between the two nodes.

In my work, each user is assigned a public key, which is used to ensure uniqueness in the system.

The whole system will be implemented for iOS in Swift and will be available as a package for the Swift Package Manager.