UniFi Software: Science and Technology 2025/26 – Distributed Programming – 03.01.2026

Project proposal Francesco Biribò (Enrollment number 7199598)

General idea:

> A decentralized, persistent, chat system where a set of nodes act as a single chat server for users, who can send messages and files to each other in private chats as well as group chats.

In particular, my idea was to develop a chat system, where users can send messages with each other over a distributed system, that acts as a single server with message persistence.

Users would be able connect to the application, create an account, send friend requests and messages to other users, both in 1-on-1, direct message style, chats and chat groups.

The chat server, on the other hand, is a distributed system of nodes that autonomously manages itself. In order to distribute the workload on multiple nodes, some of the nodes are chosen to store the messages that are sent between users, some others will be *input* nodes, while the remaining ones will just forward messages inside the network. This can be achieved by using an election algorithm to choose a leader, that is in charge of assigning the roles to the nodes.

Multiple nodes are chosen to store the messages to allow a sort of fail-over for message persistence. The input nodes are the ones that are in charge of gathering the user input and process it, starting the forwarding process inside the network. Having multiple input nodes prevents having a bottleneck on the user input side of the application, but will also require proper synchronization to deliver messages in order and avoid inconsistencies.

Finally, since the system is expected to appear as a single server, users should be able to connect to it independently from the currently active input nodes, in order to do so a simple name server could be deployed. Once the election process is over, and the leader has chosen the input nodes, it will register them under a certain URL/static IP that users know about. The clients connect to said URL/IP and will be directed to the correct node (for example http://<static-chat-url>/<username> will bring us to <username>'s profile page independently from the actual node that handles our request).

On the user, or client, side, I though about using a RESTful API in order to manage users, friend requests and group chats (just the group itself, not the messages, for which I though about using gRPC streams). This would be available both as an invokable API from the terminal (e.g. using cURL http requests) and as a web page.

On the server side, a single node will receive the request from the user and will cooperate with the other nodes in order to satisfy the request; that could be, for example, forwarding the request to the appropriate nodes (the storage ones). My idea was that the leader, once elected, would choose some nodes to be storage nodes and some to be input nodes; the input nodes will forward information to update the db with (users, messages etc) to the storage nodes, this could be achieved by making each input node know at least 1 storage node, and each storage node knows each other, so they can synchronize themselves, while input nodes send information to one node only. As for now I am not entirely sure about the mechanism to use for inter-node communication, probably gRPC, for the buffer and its streams.

Anyways, here proper synchronization is really needed, since we support multiple storage nodes, each with it's copy of the database (users, friend requests, messages, etc); the goal is persistence, even in case of a storage node failure, hence the database duplication.
Finally, on the server side, we could also manage user sessions and authentication, to avoid fraudolent activity on behalf of other users' accounts.