## Ethereum Client-Server Service

Since our work so far has been done on geth and go-ethereum (both Go implementations of the Ethereum protocol) we’re going to name this element as “GoServer”.

GoServer is a application-layer which allows users to connect to the Ethereum blockchain (or even a private blockchain) thus interacting with storages and accounts there. In our implementation we’ve chosen to use a Client-Server approach, allowing multiple users to access the network from multiple sources. An example of this would be User A accessing the network from both its PC or from its Smartphone to check for pending requests.

Such architectures not only favors a more complex yet strangely elegant approach to access the network but it also reduces the weight of each user’s application on its device. Another way to see this is like a proxy service where users send requests to a single instance which in turns runs as a proxy connecting both users and the blockchain yet still remaining in the middle of the action.

Below we illustrated the whole Client-Server implementation:

On-Chain Elements

GoServer

User C’s Workspace

User B’s Workspace

Projects Database

User A’s workspace (Same for B and C)

As you can see from the diagram multiple users are able to send requests to the network via the GoServer at once. Each request is processed and queued for sending.

In order to enable a safer experience for users, the GoServer registers each user’s device and allows only known devices to operate. For instance, if a user wants to use his/her PC to edit and submit studies said user must “register” on the GoServer in order to do so.

Registrations are either manual or automatic, it can be configured once on the server startup. An example of this would be the following scenario:

User A wants to register its smartphone as a device in order to access the network. Whether User A has any other devices registered (like a PC or a WebApp) is of no concern.

Assuming that the server has been configured in order to accessible by everyone (aka automatic registration) and User A has been already registered on the server, User A has to follow these steps in order to add its smartphone:

1. Send a NEW\_DEVICE request to the server from the unknown device.
2. Server checks whether this user is registered and the device is not listed on its database.
3. Server adds the newly registered device to its database and links it to the personal virtual database of that user.
4. Server sends a unique key to the user in order to for the user to access the server.

Since the GoServer is responsible for connecting the user to the network, it also passes on request from other sources to the user, such as Material reviews, requests and so on. Then the server stores that information in a personal database for each user registered on it, allowing users to access their history but also avoiding data loss if a sudden server breakdown occurs.

However if the user is not registered on the server there are two more scenarios to cover:

1. The server has been configured in order to accept automatic registrations.
2. The server has been configured not to accept automatic registrations.

In the second case the server owner must register the user manually from the machine running the server, then the user can proceed to register its devices.

Otherwise, automatic registrations follow these steps:

1. User sends a registration request to the server, including any info regarding itself (username, password and so on).
2. Server checks whether a user with the same name has been registered.
3. Server registers the user and sends back its ID.

Additional policies regarding servers which don’t allow automatic registrations can be implemented, like the need for specific documents in order for the server owner to register a user and so on.

### GoServer Functionalities

The server itself doesn’t do much except from calling functions located inside the on-chain elements, this is to make sure that the server and normal workflow of the network is not compromised by reverse-engineering the code of the server.