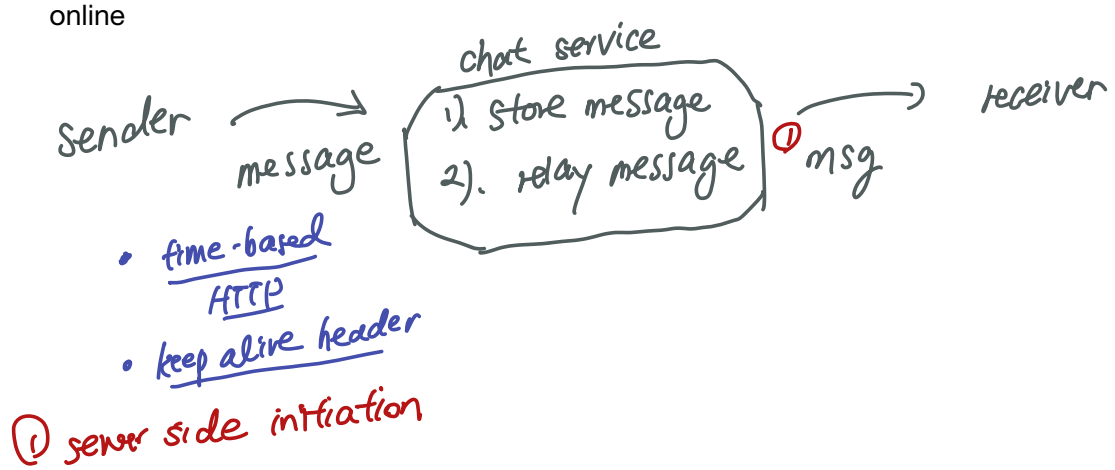
Focus on designing a chat app like Facebook messenger, it should be able to do

- 1-1 chat
- Group chat
- Online presence
- Multiple device support (can be logged into multiple accounts at the same time)
- Push notification

Backgrounds:

1. Client does not communicate directly with each other. One thing to keep in mind is that each client connects to a chat service, which supports all the features mentioned above.
   A. Receive messages from other clients
   B. Find the right recipients, relay the message to the recipients
   C. If a recipients is not online, hole the message for that recipient on the server until the person is online

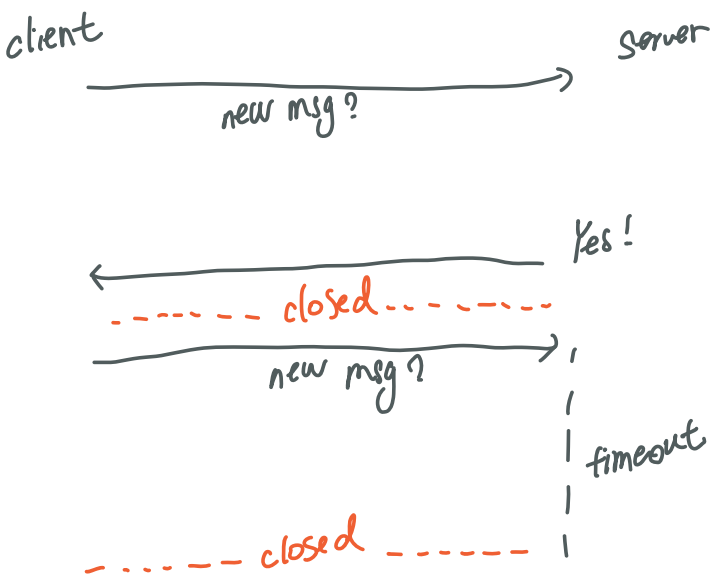


Polling

- The client periodically asks the server if there a message available. Depending on polling frequency, this can be very, very costly.
- Think about it, you don't wanna ask a question too many times. It would be very costly to the server answer time

Long Polling

- Clien asks once, server returns only if there's a message; and then client asks again

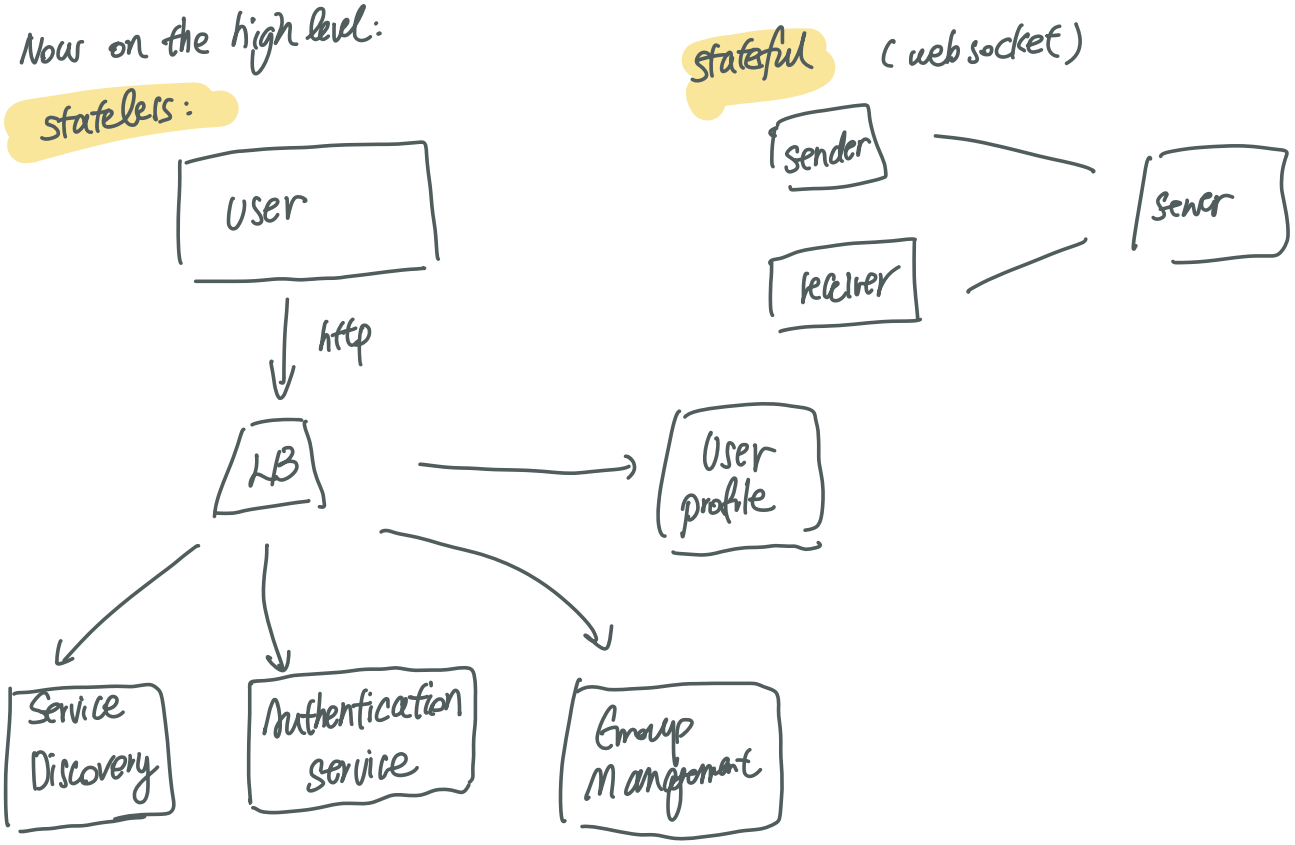


Lon Polling has some issues:

- Sender and receiver may not connect to the same chat server. HTTP based servers are usually stateless. If you use round robin for load balancing, the server that receives the messsage might not have a long-polling connection with the client who receives the message. (Basically you might just hit a different chat server at the end)
- **(New font!)**
- **It's still very inefficient, because the user is inactive, it still wastes a lot of resource.**

WebSocket

- The most common solution; The connection is initiated by the client, it's bi-directional and persistent. It starts its life as a HTTP connection and could be up graded via some well-defined handshake to a WebSocket connection. Though this persistent connection, a server could send updates to a client.
- The primary interface for connecting to a WebSocket server and then receiving data on the connection
- Allows web browser and web server with lower overhead than half-duplex alternatives such as HTTP polling, facilitating real-time data transfer from and to the server. This is made possible by providing a standardized way for the server to sending content to the client without being first requested by the client, and allowing messages to be passed back and forth while keeping the connection open



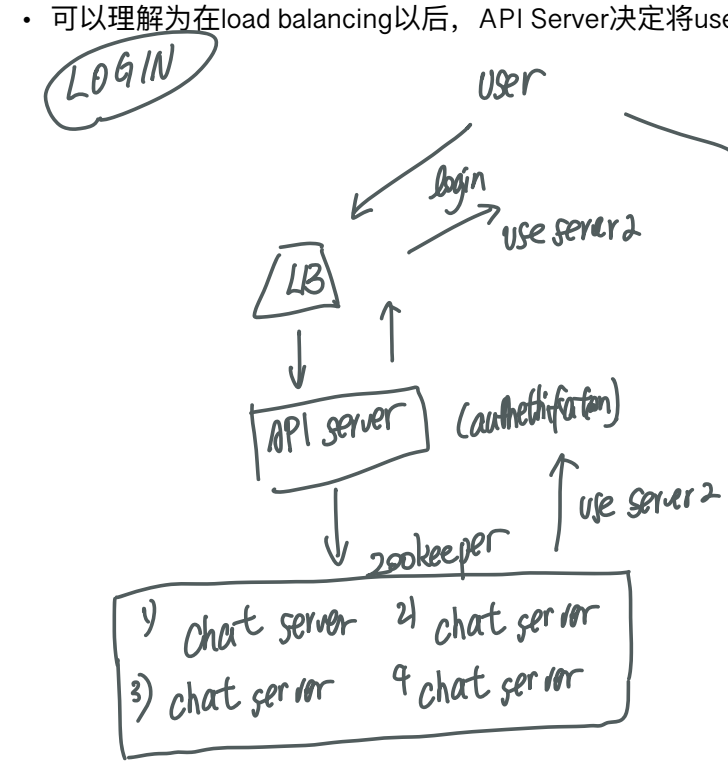
Storage/Data Layer

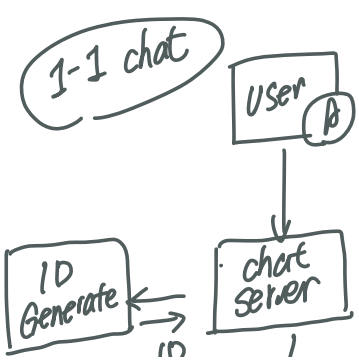
- Deep down the technical stack is the dat a layer. Data layer usually requires some effort to get it correct. An important decsision we must make is to decide on the right type of database to use. Relational or NoSQL?
- Two types of data exist in a typical chat
  - Chat history data: the amount of data is enormous for chat systems. A previous study reveals that Facebook messenger and WhatsApp provide 60 billion messages a day. **Only recent data** is used extensively; however, user will **jump to specific messages,** like how I usually search for stuffs. (The read to write is about 1:1 For 1 on 1 chat apps. (1:1 .. ?)
  - User data info: profile, setting, general information. These can be saved in a relational database

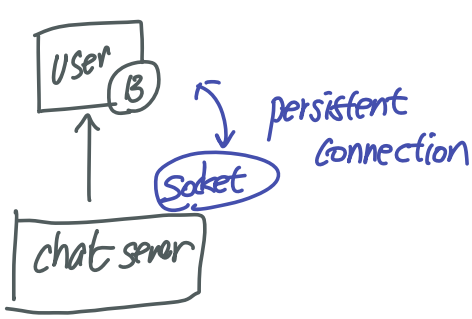
Service Discovery

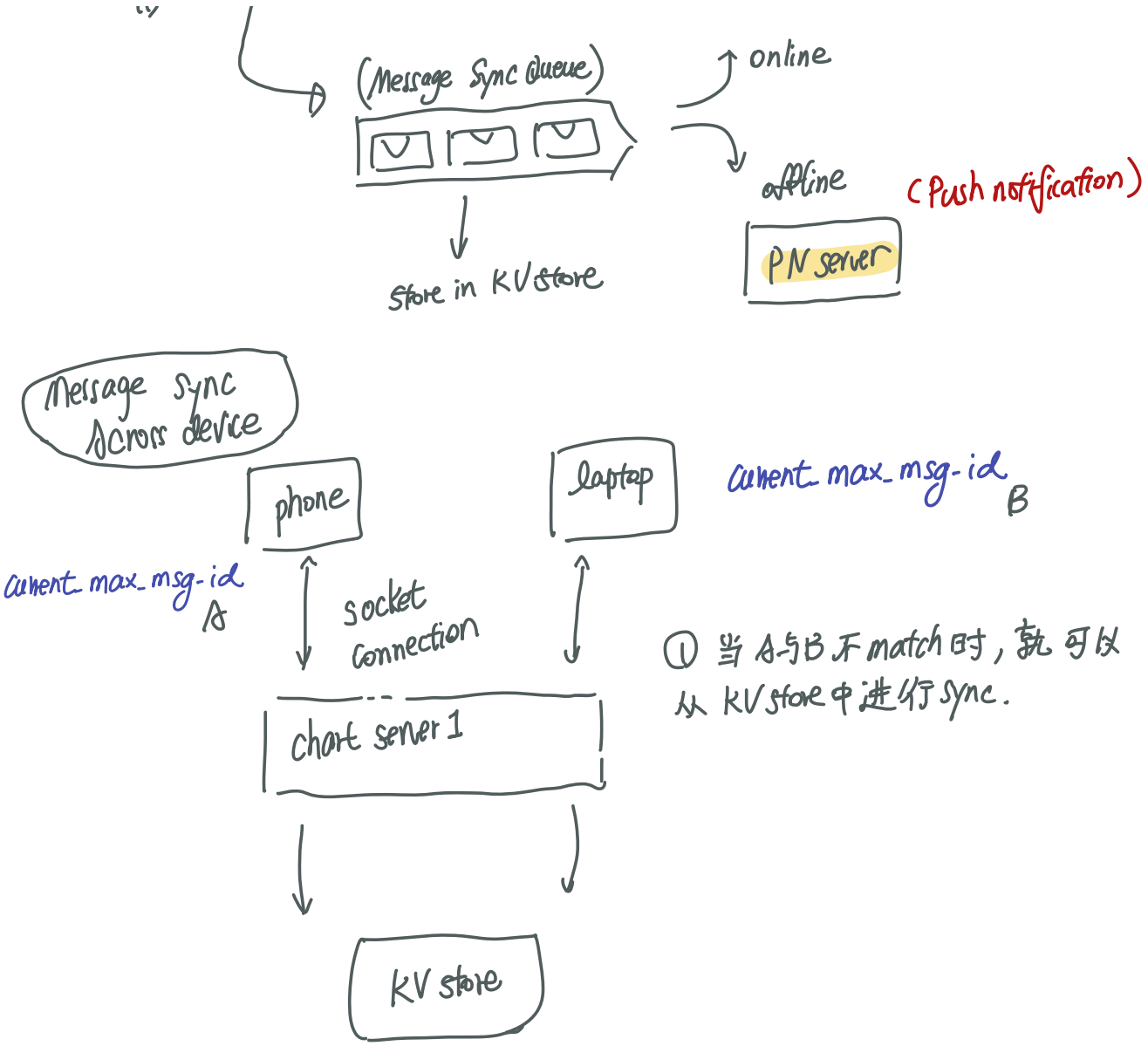
The primary role of service discovery is to recommend the best chat service for a client based on the criteria like geographical location, server capacity, etc. **Apache Zookeeper** is a popular open-source solution for service discovery. It registers all the available chat servers and picks the best chat server for a client based a predefined criteria.

- 可以理解为在load balancing以后， API Server决定将user的chat request发给具体的chat server









Online Presence:

- The online presence is mostly managed by a WebSocket. A few flows that need to be considered
- **User login:** during the service discovery stage. After a web socket connection is built between the client and the real-time service, a online status and “last_active_at” time stamp are saved. 比如，user A 进入，last-active-at 是4点，online status是active就记录好
- **User logout:** online status记录为offline
- **User disconnection:** a the persistent connection between client and the server is lost. A naive way to handle user dis connection is to mark the user as offline and change the status to online when it re-establishes. However, sometimes this happens too **often** and it will make the presence indicator change too often.
    - **Heart beat:** periodically the online client sends a heartbeat event to presence server. If a presence servers receive a heartbeat event within a certain time, it’s considered online. Otherwise, it’s offline.

    - 可以设置为，5个heartbeat （每个heartbeat为5秒），30秒内如果server没有收到任何heartbeat的时候，就把Client的status定为offline
- **Online Status Fanout:**
    - Presence Server uses a **pub-sub** model, in which each friend pair maintains a channel. When User A's online status changes, it publishes the event to three channels, channel A-B, A-C, and A-D (they are subscribed to user A). The communication between clients and servers is through real-time WebSocket. 这样的fanout对小的group是可行的，但是一旦人数上来就会有bottleneck。