Twitter System Design

Functional Requirements

- Profile Capabilities
    - Upload images/videos to profile
    - Set biography and privacy settings
        - Maintain list of blocked users
        - Can choose to keep profile private to public unless followed
    - Keep following/follower list
        - If private, user must accept follow before other user can follow
- News Feed
    - Include trending posts
    - Include popular posts that are of interest to the user
    - Include posts from people that the user follows
    - Include posts from people from their follow list that have interacted with other posts
    - Keep track of posts that the user interacts with to continue to show the user posts that they would like seeing
- Direct messaging
    - 1-1 communication OR group communication
        - Keep chat history
        - Keep timestamp
        - Private conversation so encrypt
        - Keep conversation id
        - Notify user(s) when receiving a message
        - Keep in inbox unread if not opened yet
        - Can add other users to conversation
- Create posts
    - Can set privacy settings
    - Can edit post once posted
    - Interactions with user post will notify user if the correct settings are on
    - Like, Share, Comment, Post, Reply
    - Video/images/Links support
    - Mentions/Tagging
        - Must be able to link to and alert this other user

Non-Functional Requirements

- Platform should scale to 1B daily users
- Platform should be consistent on multiple devices
- Users should have a personal feed of their interests based on interaction

Overall Assumptions

- 1B daily users
- 10 posts/day per user
- 40 messages per day
- Posts and messages about 500Mb (includes video/images)

Storage Estimation

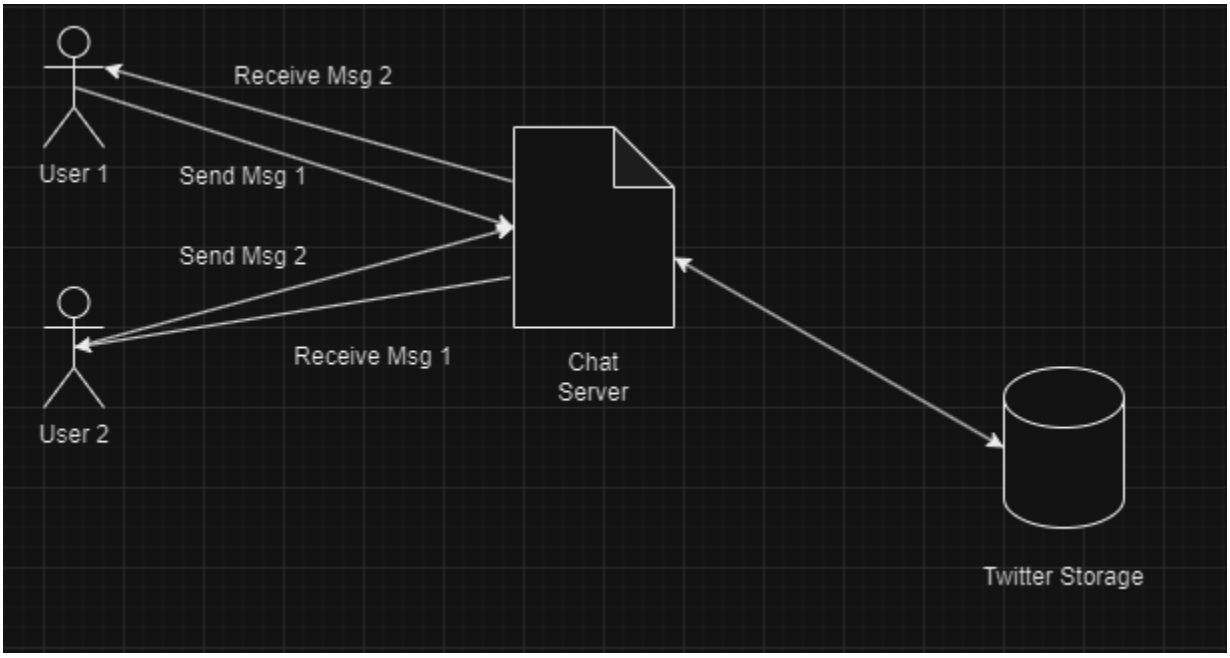
Storage requirement for 1 day = 1B*10*40 = 400B messages/posts * 500MB = 20,000TB per day

Bandwidth Estimation

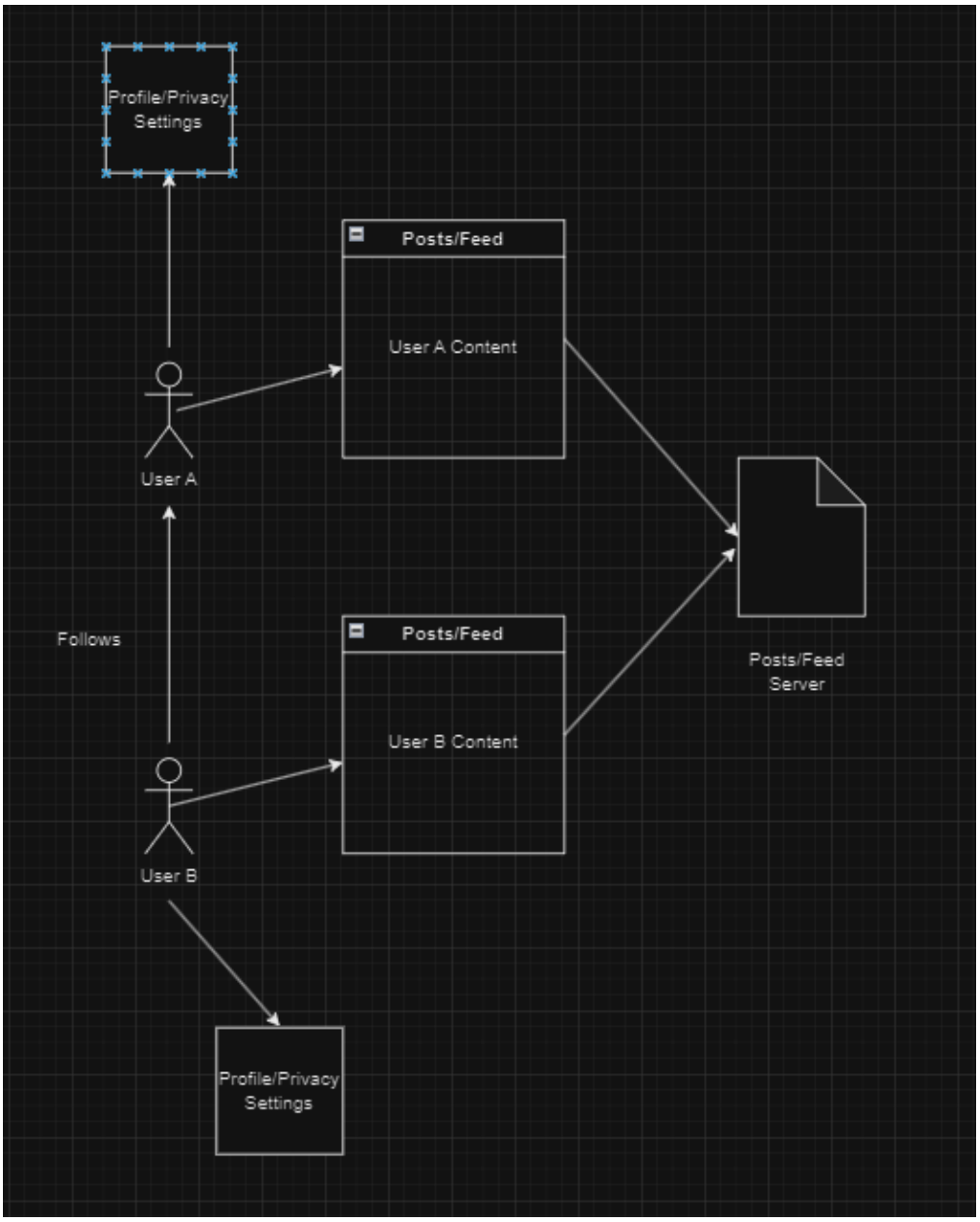
20,000TB/86400 = 185MB/s

High-Level Design

For direct messaging



For posts/feed/profile



Chat server – handles communication between users

-- each server has a hashtable with a user ID and connection object

WebSocket – handles bi-directional communication between chatServer and user

DB for storing all users chat histories

Load Balancer to handle all of users connection requests

CreatePost(content,media,uid)

GetFeed(uid)

Follow(uid,user2id)

Message(author,msgID,content,time)

Send(author,destinationUser,msgID,encryption)

Receive(author,destinationUser,msgID,encryption)

StoreMsg(author,destination,msgID,encryption)

Groupmsg(list of userIDs, groupID, encryption)

Database – relational database like a graph to maintain following/followers

End to End Flow

Goal: UserA intends to post to their feed

1. User creates a blank post and fills it with content to post to their feed
2. Post goes to feed server and stores it
3. Feed server sees who follows User1 to send to their feed
    a. Additionally, if it is a trending tweet, it will show users who have similar interests
4. Algorithm decides if it is relevant to their feed

Goal: UserA intends to send a message to UserB

1. UserA sends message to UserB
2. Message goes to chatServer
3. Chatserver receives the msg and stores it
4. Server looks at destination and sends it to closest server if not current server
5. Server sends UserB message