Scalable Twitter-Like System Design Documentation

1. System Overview and Requirements

A. Functional Requirements:

- Tweet Creation and Management: Users can post tweets, retweet, like, and reply. Tweets are immutable once published.
- Timeline/Feed Generation: Real-time and near-real-time generation of personalized timelines based on follower relationships.
- User and Relationship Management: Maintain user profiles and follower/following relationships.
- Search and Discovery: Enable keyword and hashtag searches across tweets.
- Notifications: Provide real-time notifications for mentions, replies, and other activities.
- Analytics: Collect and monitor performance metrics, user engagement, and overall traffic.

B. Nonfunctional Requirements:

- Scalability: Must support billions of users and handle tens of thousands of requests per second.
- Low Latency: Ensure rapid tweet display, feed generation, and notifications.
- High Availability and Fault Tolerance: Design for redundancy across geographic regions, avoiding single points of failure.
- Efficient Data Storage: Capable of storing billions of tweets over several years and supporting high write/read rates.
- Security: Use HTTPS for all client interactions and secure protocols for interservice communication.

C. Assumptions and Realistic Traffic Estimates:

- User Base: Assume 200 million registered users with approximately 1-2% active concurrently.
- Tweet Write Load: Plan for up to 10,000 tweet writes per second at peak.
- Read Traffic: Timeline fetches and profile views can exceed 100,000 queries per second.
- Follower Graph: Each active user may follow hundreds of others, potentially leading to millions of fanout messages per second.

2. Component Responsibilities and Data Flow

A. Client Interaction and Global Access:

- Client Devices (Mobile and Web): Use HTTPS for secure connections and WebSocket (over TLS) for real-time updates.
- Global Load Balancer and DNS: Direct incoming requests by geographic proximity and load; DNS-based routing sends users to the appropriate regional load balancer.

B. API Gateway and Authentication:

- API Gateway: Authenticates incoming requests, enforces rate limits, and routes traffic to backend services.

* Handles between 50,000 to 100,000 requests per second during peak periods.

* Uses HTTPS for client-to-API and gRPC/REST (over secured TCP) for interservice communication.

- Authentication Service: Verifies user credentials and issues secure tokens.

C. Tweet Ingestion and Processing:

- Tweet Ingestion Service:

* Validates input (length checks, prohibited content, etc.).

* Generates unique Tweet IDs using a Distributed ID Generator.

* Persists tweet data to the Tweet Data Store.

* Publishes messages to a Message Broker for timeline fanout.

* Designed to support up to 10,000 tweet writes per second.

- Message Broker (e.g., Apache Kafka): Decouples tweet ingestion from downstream processing and supports high throughput (millions of messages per second).

D. Data Storage and Persistence:

- Tweet Data Store (NoSQL Distributed Database):

* Data is partitioned by Tweet ID or User ID using consistent hashing.

* Data replication (e.g., replication factor of 3) ensures durability and fault tolerance.

* Each tweet record includes Tweet ID, User ID, content, timestamp, and engagement counters.

- User Data Store (Follower Graph):

* Stores user profiles and follow relationships.

* Optimized with partitioning by User ID (using in-memory indexing or a dedicated graph database).

- Timeline Cache (In-Memory Store, e.g., Redis Cluster):

* Holds precomputed timelines for fast retrieval with sub-100ms latency.

* Deployed as a cluster with replication and partitioning for heavy read loads.

E. Timeline Generation and Fanout Service:

- Timeline Generation Service:

* Retrieves follower lists from the User Data Store.

* Fans out new tweets to update follower timelines in the Timeline Cache.

* Uses asynchronous and batch processing to manage high fanout volumes.

- Feed Retrieval Service: Fetches precomputed timelines from the Timeline Cache for user requests.

F. Ancillary Services:

- Notification Service: Sends real-time push notifications and in-app alerts for new tweets, replies, and mentions.
- Search Service: Indexes tweets for keyword and hashtag search using engines like Elasticsearch.
- Analytics and Monitoring Service: Collects key metrics (volume, latency, error rates) and monitors overall system health.

3. Protocols and Traffic Handling

- Client-to-Server Communication:

* All client interactions use HTTPS for secure communication.
* Real-time updates employ secure WebSocket connections over TLS.

- Interservice Communication:

* Use gRPC or internal REST over secured TCP connections (with mutual TLS).

- Load Balancing:

* Global load balancing is achieved through DNS-based routing.
* Regional load balancers (e.g., HAProxy or AWS ELB) distribute the traffic among available API gateways.

- Traffic Management:

* Autoscaling policies dynamically add instances during traffic spikes (up to 100K requests per second).
* Caching layers (such as the Timeline Cache) reduce direct load on persistent stores and maintain sub-100ms response times.

4. Final Thoughts

This design outlines a robust architecture for a Twitter-like system that meets high scalability, low latency, and high availability requirements. It clearly describes how:

- The NoSQL data store partitions and replicates tweet data across multiple servers.
- Various services (API Gateway, Tweet Ingestion, Timeline Generation, and Ancillary Services) handle realistic traffic loads.
- Secure protocols and load balancing techniques are employed to manage traffic efficiently.

The resulting system is engineered to support billions of users and millions of concurrent operations, ensuring both performance and reliability.