PA3: Design Doc

I. Introduction

This document describes the design of a decentralized P2P system that supports a publish/subscribe mechanism for topic-based message exchange. The system leverages a Distributed Hash Table (DHT) for efficient topic routing and uses a hypercube topology for optimized peer-to-peer communication. This design document provides an overview of the architecture, key components, design trade-offs, and possible extensions for future improvements.

II. System Architecture

The system consists of multiple peer nodes that communicate in a decentralized manner. Each peer can:

- Create, delete, publish to, and subscribe to topics.
- Use DHT to ensure efficient topic-to-peer mapping.
- Use a hypercube topology for routing requests and messages between peers.

The key components are:

- **Peer Node**: Each peer in the network, identified by a unique binary ID.
- **DHT-based Topic Routing**: Hash function used to assign topics to specific peers.
- **Hypercube Topology**: Ensures efficient routing by creating neighbors through bit-flipping.
- **Publish/Subscribe Mechanism**: Allows for the creation of topics and the publishing and subscribing of messages to topics.

III. Design Decisions

3.1 Peer Node Design

Each peer is initialized with:

- A unique **binary ID** (3 bits for 8 peers).
- An **IP address** and **port number** to facilitate network communication.
- A list of **neighbors**, calculated by flipping bits of the peer's binary ID (forming a hypercube).

Key Features of Peer Node:

- **Topic Storage**: Each peer stores topics assigned to it via the DHT.
- **Subscribers**: Keeps track of subscribers to topics.
- **Message Propagation**: Forwards messages to subscribers using DHT routing.
- **Request Handling**: Can create, delete, publish, and subscribe to topics based on requests received.

3.2 Distributed Hash Table (DHT)

- The **hash function** maps a topic name to a peer ID using SHA-256 and modular arithmetic. The peer responsible for the topic is determined by hashing the topic name and using the modulo of the hash value with the total number of peers.
- This ensures that topics are evenly distributed among peers and are assigned to the appropriate peer for storage and management.

3.3 Hypercube Topology

- Each peer has a set of **neighbors** derived by flipping one bit of its binary ID at a time. This creates an optimal routing network where each peer can efficiently find the next hop to forward a request.
- The **hypercube** allows for minimal hops (equal to the number of bits in the ID) between peers and ensures that each peer can access any other peer by following the routing path.

3.4 Publish/Subscribe APIs

The core publish/subscribe APIs include:

1. **Create Topic**: Allows a peer to create a topic if it is responsible for it.
2. **Delete Topic**: Allows a peer to delete a topic if it is responsible for it.
3. **Publish Message**: Allows a peer to publish a message to a topic.
4. **Subscribe to Topic**: Allows a peer to subscribe to a topic and receive messages from it.

These APIs are implemented asynchronously to handle multiple requests concurrently, ensuring non-blocking operations across the system.

IV. Trade-offs in Design

4.1 Hash Function

Trade-off:

- The hash function uses **SHA-256**, which ensures a uniform and secure distribution of topics. However, it can be computationally expensive, especially with a larger number of peers and topics.

Possible Improvement:

- A more lightweight hash function could be considered if performance under high load becomes a concern.

4.2 Hypercube Topology

Trade-off:

- The hypercube topology is highly efficient in terms of routing, ensuring that the number of hops between any two peers is minimized. However, the routing complexity grows with the number of bits in the peer ID, which could be limiting as the system scales.

Possible Improvement:

- For a large-scale system, consider transitioning to a different topology, such as **Chord** or **Kademlia**, which scales better with a higher number of peers.

4.3 Publish/Subscribe System

Trade-off:

- The system ensures that topics are created, deleted, and subscribed to efficiently through the DHT and hypercube mechanisms. However, when there are many subscribers or topics, **message propagation** can become slower due to the increased number of hops.

Possible Improvement:

- Implementing **message batching** or **caching** strategies could help mitigate the performance degradation caused by a large number of subscribers or topics.

V. Testing and Benchmarking

5.1 Request Forwarding

The request forwarding mechanism is tested by verifying that each node can access topics on all nodes in the network. This is essential to ensure that the routing mechanism is working as expected. During the testing phase, we also measure **response time** and **throughput** to assess the system's performance.

5.2 Latency and Throughput

Benchmarking is conducted using random workloads. The following metrics are collected:

- **Latency**: Time taken for a request to travel from one peer to another and get a response.
- **Throughput**: The number of requests processed within a fixed time window.

This data is used to identify bottlenecks in the system and optimize various components, such as the message forwarding mechanism and the hash function.

VI. Possible Improvements

6.1 Optimizing Routing for Larger Networks

Currently, the system supports up to 8 peers with a 3-bit ID. However, for scalability, we need to extend the peer ID length and modify the routing algorithm to support more peers efficiently.

How to Implement:

- Increase the bit-length of the peer ID to handle a larger number of nodes (e.g., 16-bit for 256 peers).
- Adapt the hypercube-based routing mechanism to handle more complex topologies and larger networks.

6.2 Fault Tolerance and Redundancy

Currently, the system assumes that all peers are online and operational. However, in real-world applications, peers may fail or become unavailable.

How to Implement:

- Implement **replication** to store copies of topics across multiple peers, ensuring that the system is fault-tolerant.
- Use a **heartbeat mechanism** to monitor peer availability and trigger re-routing if a peer fails.

6.3 Security and Privacy

Currently, the system does not implement any form of encryption or authentication for requests. As a future improvement, it would be beneficial to secure communication between peers.

How to Implement:

- Use **TLS** or **SSL** to encrypt messages between peers.
- Implement **authentication** and **authorization** mechanisms for topic access control.

6.4 Caching and Optimization

As the number of topics and subscribers grows, message propagation can become a bottleneck. Implementing caching strategies and optimizing data storage can help reduce overhead.

How to Implement:

- Implement a **local cache** for each peer to store recently accessed topics and messages, reducing the need for repeated lookups.
- Optimize **message forwarding** by batching messages for multiple subscribers.

VII. Conclusion

This P2P system provides a foundation for decentralized communication using DHT and hypercube topology. While the current implementation is optimized for a small network of peers, there are opportunities for scaling the system, improving performance, and adding features such as fault tolerance, security, and caching. These improvements would make the system more robust and suitable for larger, real-world applications.