## DHT P2P System Design Document

## System Architecture

Our DHT P2P System is designed as a decentralized network of 8 nodes arranged in a 3-dimensional hypercube topology. Each node is identified by a 3-bit binary ID (000 to 111) and runs on a separate port (5000 to 5007).

## Key Components:

1. **Peer Nodes**: Implemented in main.py, each node can create topics, publish messages, subscribe to topics, and query the DHT.
2. **DHT Implementation**: The core DHT functionality, including topic storage and message routing, is likely implemented in a separate file (e.g., dht.py).
3. **API Layer**: The system provides RESTful APIs for creating topics, publishing messages, subscribing to topics, and querying the DHT.
4. **Testing and Benchmarking**: Two separate scripts (test\_script\_benchmark.py and test\_script\_peer.py) are used for performance testing and functional testing respectively.

## Design Decisions and Tradeoffs

1. **Hypercube Topology**:
   * Decision: We chose a 3-dimensional hypercube topology with 8 nodes.
   * Tradeoff: This provides a balance between network complexity and routing efficiency. However, it limits scalability to powers of 2.
2. **Asynchronous Operations**:
   * Decision: We used Python's asyncio for concurrent operations.
   * Tradeoff: This improves performance and allows handling multiple requests simultaneously, but increases code complexity.
3. **RESTful API**:
   * Decision: We implemented a RESTful API for inter-node communication.
   * Tradeoff: This provides a standardized interface but may have higher overhead compared to custom protocols.
4. **In-Memory Storage**:
   * Decision: Topics and messages are likely stored in-memory (based on the simplicity of the test scripts).
   * Tradeoff: This provides fast access but limits persistence and scalability.
5. **Separate Benchmark and Peer Testing**:
   * Decision: We created separate scripts for benchmarking and peer testing.
   * Tradeoff: This allows for focused testing of different aspects but may lead to some code duplication.

## Possible Improvements and Extensions

1. **Scalability**:
   * Improvement: Extend the system to support a variable number of nodes, not limited to powers of 2.
   * Implementation: This would require modifying the routing algorithm and potentially the node addressing scheme.
2. **Persistence**:
   * Improvement: Add persistent storage for topics and messages.
   * Implementation: Integrate a database (e.g., SQLite for each node) to store data persistently.
3. **Fault Tolerance**:
   * Improvement: Implement data replication and node failure recovery.
   * Implementation: Add mechanisms for data replication across multiple nodes and procedures for redistributing data when nodes fail or new nodes join.
4. **Security**:
   * Improvement: Add authentication and encryption for inter-node communication.
   * Implementation: Implement SSL/TLS for API calls and add an authentication mechanism for nodes.
5. **Load Balancing**:
   * Improvement: Implement dynamic load balancing to distribute topics more evenly.
   * Implementation: Develop a load monitoring system and algorithms to redistribute topics based on node load.
6. **Caching**:
   * Improvement: Implement a caching layer to improve performance for frequently accessed topics.
   * Implementation: Add a caching mechanism at each node, with cache invalidation protocols.
7. **Advanced Routing**:
   * Improvement: Implement more sophisticated routing algorithms to optimize message delivery.
   * Implementation: Research and implement algorithms like Kademlia or Chord for more efficient routing.
8. **Comprehensive Monitoring**:
   * Improvement: Add a monitoring system for real-time performance and health metrics.
   * Implementation: Integrate a monitoring solution (e.g., Prometheus) and create a dashboard (e.g., Grafana) for visualizing system metrics.

## Conclusion

Our current DHT P2P System provides a solid foundation for a distributed topic-based messaging system. The design choices made allow for efficient routing in a small-scale network while providing a clear API for topic management and message exchange. The separate benchmarking and peer testing scripts enable thorough evaluation of both performance and functionality.However, there is significant room for improvement, particularly in areas of scalability, persistence, fault tolerance, and security. These improvements would be crucial for deploying the system in a production environment or scaling it to a larger number of nodes.