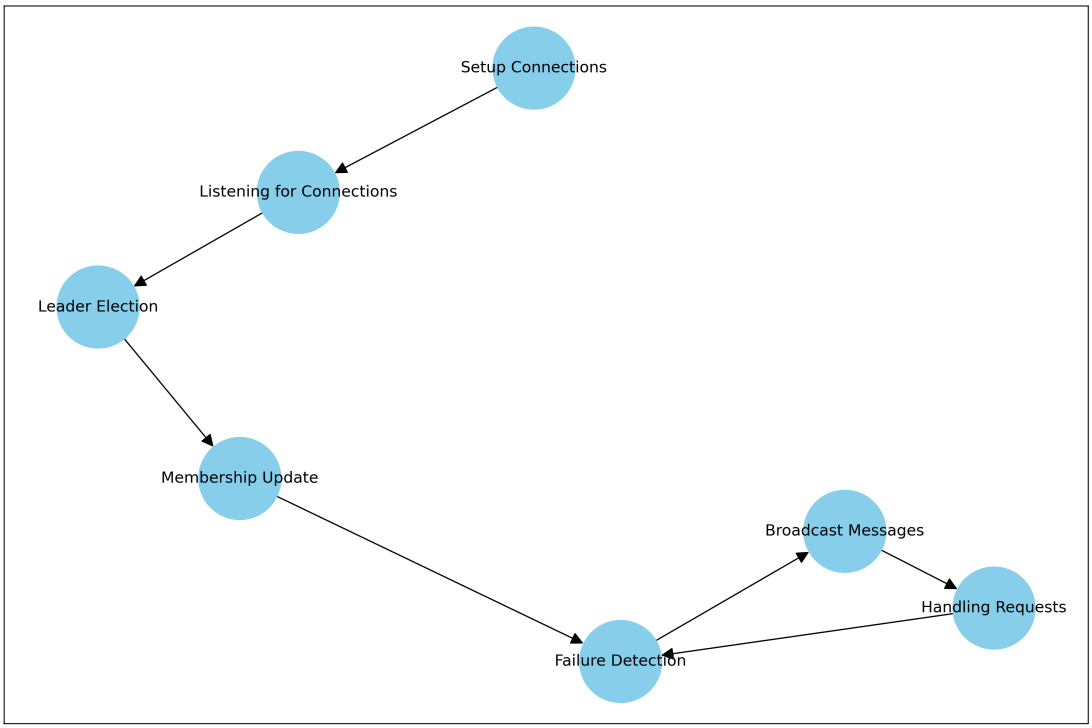
Report: Distributed Membership System

System Architecture

The system implements a distributed membership service where a group of peers communicates over a network to maintain a shared list of active processes. Each peer runs an instance of the membership protocol, participating in leader election, failure detection, and membership updates. The architecture leverages TCP and UDP communication, multi-threading for concurrent tasks, and a fault-tolerant design using periodic heartbeats and message exchanges.

Key Components:
- **Peers**: Each peer runs an instance of the membership protocol and communicates with others through TCP for control messages and UDP for heartbeat monitoring.
- **Leader Election**: A leader is chosen among the peers to manage membership changes. The leader initiates updates and handles pending operations.
- **Membership List**: Maintains the active set of processes in the system, dynamically updated based on failure detection and peer joins/leaves.
- **Failure Detection**: Peers send periodic heartbeats to each other to monitor liveness. If a heartbeat from a peer is missed for a specified duration, the peer is considered failed.
- **Pending Operations**: Add or delete operations are considered pending until confirmed by the majority, ensuring consistent updates across the membership list.

State Diagram



Design Decisions

1. **Concurrency Model**: Multiple threads are used to handle different tasks concurrently—TCP message handling, heartbeat sending, and failure detection.
   - **Message Listener**: Each peer has a thread dedicated to listening for TCP messages, allowing real-time handling of membership updates and requests.
   - **Heartbeat Mechanism**: A separate thread for sending heartbeats ensures that failure detection does not block other operations.

2. **Leader-Based Membership Management**: The leader is responsible for updating the membership list when peers join or leave the network. This ensures a single source of truth for the membership list, preventing conflicting updates.
   - **Leader Selection**: When a leader fails, a new leader is chosen based on a predefined algorithm, ensuring the system remains operational.

3. **Failure Handling**: The system detects peer failures by monitoring heartbeats and removes unresponsive peers from the membership list.
   - **Crash Handling**: Peers may intentionally crash after performing an operation (e.g., for test cases), and the system handles these cases through heartbeat-based failure detection.

4. **Message Serialization/Deserialization**: Messages are serialized into strings for TCP transmission and deserialized upon receipt, ensuring compatibility and consistency across peers.

Implementation Issues

1. **Connection Management**: Handling incoming and outgoing TCP connections between peers is challenging, especially when peers can disconnect or crash. The system ensures that connections are continuously monitored, and broken connections are re-established if necessary.
   - **Solution**: By using `recv` with non-blocking and timeout options, the system checks for broken connections and removes them from the active connection list.

2. **Heartbeat and Failure Detection**: The reliability of the failure detection mechanism is critical, especially when network delays or high loads cause missed heartbeats.
   - **Solution**: The system introduces configurable heartbeat intervals and failure detection thresholds to handle varying network conditions.

3. **Concurrency**: Multiple threads accessing shared data (e.g., membership list) introduce race conditions.
   - **Solution**: Mutexes are used to protect shared data structures like the membership list and connection maps, ensuring thread safety.

4. **Leader Election Timing**: A delay in detecting the leader's failure could cause the system to stall until a new leader is chosen.
   - **Solution**: A combination of heartbeat monitoring and quorum-based consensus ensures that leader failure is quickly detected and resolved.

Test Case 4 Implementation

For the specific test case 4 (`-t` option), the leader initiates a sequence of operations:
1. Waits for all peers to join the system.
2. Sends a `REQ` message of type `DEL` to all peers except the next leader.
3. Crashes the current leader, forcing a new leader election.

This test case verifies the robustness of the system in handling leader crashes during membership updates and ensuring that the new leader takes over seamlessly.

Conclusion

This distributed membership system ensures robust fault tolerance, dynamic membership updates, and efficient failure detection. By leveraging a combination of TCP for control messages and UDP for heartbeats, along with multi-threaded concurrency, the system provides a reliable

solution for managing distributed peers. The design decisions around leader-based membership management and failure detection play a key role in ensuring system consistency and availability.