1 The Byzantine Fault Tolerance (BFT) Problem

In a distributed network, not every participant can be trusted. Some may fail, lie, or act maliciously. This leads to the *Byzantine Fault Tolerance Problem*:

> **Problem:** *If all the generals coordinate their attack and strike at the same time, the battle is won. However, if two generals falsely claim they will attack but instead retreat, the battle is lost. This failure in coordination undermines the entire strategy.*

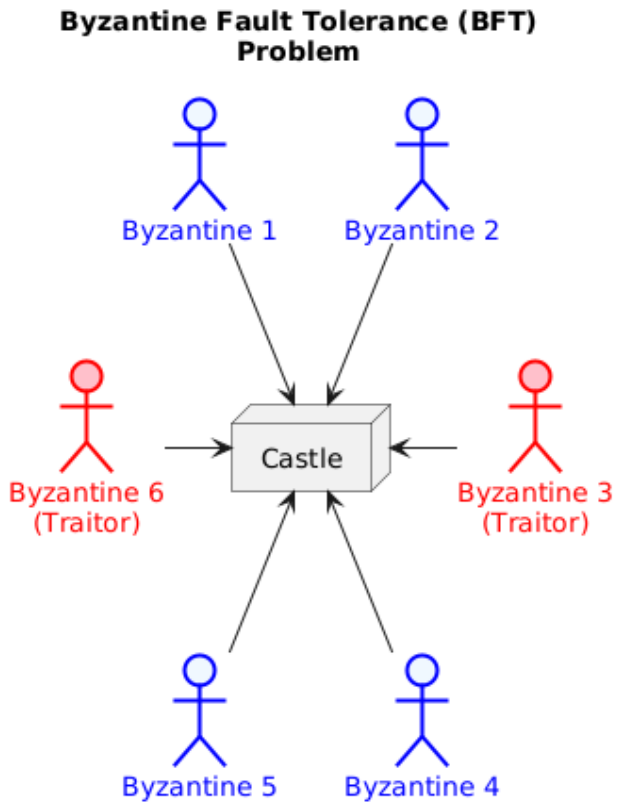

Figure 1: Image

1.1 Fault Tolerance

Fault tolerance is the ability of a system to continue operating correctly even when some of its components fail:

- **Crash Faults:** Nodes may crash or become unresponsive.
- **Byzantine Faults:** Nodes may behave arbitrarily, including sending conflicting information to different nodes.

> **Theorem:** A distributed system can tolerate up to faulty nodes (f) if it has at least $3f + 1$ nodes (n) in total, so $n \geq 3f + 1$.

$\boxed{PROOF}$

1. **Majority Requirement:** For consensus to be achieved, the honest nodes must form a majority. This ensures that the faulty nodes cannot sway the decision.
2. **Node Distribution:**
   - **Total nodes:** $n = 3f + 1$
   - **Faulty nodes:** f
   - **Honest nodes:** $n - f$
3. **Majority Condition:**
   - For honest nodes to form a majority: $2f + 1 > f$
   - This inequality holds true because $2f + 1 - f = f + 1 > 0$.
4. **Fault Tolerance:** Even if f nodes are faulty, the remaining $2f + 1$ honest nodes can outvote the faulty nodes, ensuring consensus.
5. **Conclusion:** A distributed system with $n = 3f + 1$ nodes can tolerate up to f faulty nodes while maintaining both safety and liveness.

2 BFT Protocols

They are designed to ensure that a distributed system can reach consensus even in the presence of faulty or malicious nodes. The goal is to allow a network of nodes to agree on a single value or state, even if some nodes are compromised.

$\boxed{GOALS}$

- **Safety:** The system must ensure that all honest nodes agree on the same value.
- **Liveness:** The system must ensure that the protocol eventually terminates and all honest nodes reach a decision.
- **Fault Tolerance:** The system must tolerate a certain number of faulty nodes without compromising safety or liveness.

2.1 Byzantine Broadcast (BB) Protocol

This protocol is designed for synchronous networks and allows a source node to broadcast a message to all other nodes in the network, ensuring that even if some nodes are faulty, the remaining honest nodes can still reach consensus on the message.

$\boxed{ALGORITHM}$

1. **Initialization:** The source node sends a message to all nodes.
2. **Propagation:** Each node forwards the message it receives to all other nodes.
3. **Acknowledgment:** Each node applies a decision rule (e.g., majority voting) based on the messages received to agree on the final value.
4. **Termination:** The protocol terminates when all honest nodes agree on the same value.

Image

Figure 2: Image

2.1.1 Single Point of Failure

The protocol relies on a single source node to initiate the broadcast. If this node is compromised, it can send conflicting messages to different nodes, potentially disrupting consensus. This creates a *Single Point of Failure*, where the entire system never reaches consensus.

2.2 Rotating Leaders Protocol

This protocol is designed for partially synchronous networks and is a variant where the leadership role rotates among nodes in successive rounds or epochs.

> **Single Point of Failure Solution:** In this protocol, nodes take turns being the leader, ensuring that no single node has control over the entire system.

$\boxed{ALGORITHM}$

1. **Initialization:** A leader is selected (e.g., using a round-robin or random selection).
2. **Proposal:** The leader proposes a value to the other nodes.
3. **Voting:** Each node votes on the proposed value, and if a majority agrees, the value is committed.
4. **Rotation:** After a round, the leadership role rotates to the next node.

Image

Figure 3: Image

2.2.1 Round-Robin Leader Election

The leader is chosen in a round-robin fashion, ensuring that each node has an equal opportunity to propose values. This prevents any single node from dominating the protocol and reduces the risk of a single point of failure.

2.2.2 Random Leader Election

In this approach, a leader is randomly selected for each round. This randomness helps to distribute power among the nodes and can mitigate the impact of a single point of failure.

2.3 Dolev-Strong Protocol

This protocol is designed for synchronous networks to tolerate up to $t < n/3$ malicious nodes. It ensures that all honest nodes agree on the same value, even in the presence of adversarial behavior. The protocol leverages digital signatures to authenticate messages and prevent tampering, making it robust against Byzantine faults.

$\boxed{ALGORITHM}$

1. **Initialization:** The sender signs a message m and sends it to all other nodes.
2. **Propagation:** Each node that receives m with i signatures propagates it to all others, adding its own signature (now there are $i + 1$ signatures). Propagation continues for $t + 1$ rounds.
3. **Acknowledgment:** Each node accepts the message only if it has received a copy with at least $t + 1$ distinct signatures.
4. **Termination:** The protocol terminates when all honest nodes agree on the same value.

Image

Figure 4: Image

2.4 Tendermint Protocol

This is a protocol is designed for partially synchronous networks and is a state-of-the-art BFT consensus algorithm designed for blockchain systems. It combines the benefits of a Byzantine Fault Tolerant consensus mechanism with a practical approach to achieving consensus in a distributed network.

$ALGORITHM$

1. **Initialization:** Nodes are organized into a validator set, and a leader (proposer) is selected for each round.
2. **Proposal:** The leader proposes a block of transactions to the network.
3. **Pre-vote:** Validators broadcast their pre-vote for the proposed block if it is valid.
4. **Pre-commit:** Validators broadcast their pre-commit if they receive a majority of pre-votes for the block.
5. **Commit:** If a validator receives a majority of pre-commits, it commits the block and adds it to the blockchain.
6. **Rotation:** The leadership role rotates to the next validator for the next round.

Image

Figure 5: Image