# Question 4 Report

*Kshitij Nigam*
*2023202031*

## 1. Broad Overview of the System

The Byzantine Fault Tolerance (BFT) system is designed to simulate the behavior of generals in a Byzantine Fault scenario, ensuring that consensus can still be achieved even in the presence of faulty or malicious nodes. The system leverages gRPC, an open-source RPC framework, for communication between nodes (representing generals). It simulates the process of decision-making where each general (node) must agree on a strategy, such as "Attack" or "Retreat," despite the presence of traitors (malicious generals). The key features of the system include message propagation, round-based decision-making, majority voting, and fault tolerance. The system adheres to the Byzantine Fault Tolerance problem constraints and ensures that honest generals reach the same final decision, even with up to t traitors.

## 2. Design Choices for Each Requirement

- **gRPC Framework**: We chose gRPC as the communication mechanism because it offers high-performance, strongly-typed communication, and supports features like authentication, bidirectional streaming, and flow control, which are critical for a distributed system simulation. It also integrates well with Protocol Buffers for defining interfaces and ensures efficient data serialization.
- **Node Representation**: Each node in the system represents a general in the Byzantine simulation. A node has an ID, a flag indicating if it is the commander or a lieutenant, and a flag indicating whether it is a traitor. The decision-making process for each node is based on the messages it receives from other nodes, and the node updates its decision through majority voting after each round.
- **Message Propagation**: The system propagates messages from one node to another. Each node sends its current decision to all its peers in each round. This ensures that all generals are aware of the decisions made by others, and the information is propagated redundantly, ensuring consensus despite faulty nodes.
- **Majority Voting**: After receiving messages from all other generals, each node makes a decision based on the majority vote. If the majority votes "Attack," the decision is "Attack"; otherwise, "Retreat." This ensures that the system converges to a final decision based on the majority opinion, even if some nodes are traitors.
- **Fault Tolerance**: The system is designed to tolerate up to t traitors. The condition $n > 3t$ is enforced, ensuring that the number of honest generals is always greater than the number of traitors, allowing for the system to reach consensus despite malicious behavior.
- **Simulation Rounds**: The system operates in multiple rounds where the generals exchange orders and update their decisions. The rounds simulate the

communication delays and the process of refining consensus over time. The rounds include the initial commander broadcast, cross-verification, secondary verification, and continued propagation, followed by the final consensus.

## 3. Assumptions and Future Work

- **Assumptions**:
    - The number of generals n and traitors t are known at the start of the simulation.
    - The message delay is bounded and synchronous, meaning each node receives messages within a known time window.
    - The traitors will always attempt to deceive honest generals by providing false information, while honest generals will always follow the protocol and vote based on the majority decision.
    - The system assumes that the number of generals n is greater than 3t, which is necessary for reaching a consensus even with traitors present.
- **Future Work**:
    - **Fault Simulation**: We can further enhance the system by simulating network failures or delays and observing how the system adapts to these real-world scenarios.
    - **Scalability**: The current system is designed for a small number of generals. Future work could include optimizing the system for larger-scale simulations with hundreds or thousands of generals.
    - **Performance Optimizations**: We could explore performance improvements such as reducing the number of message exchanges required in each round or employing more advanced consensus protocols.
    - **Advanced Fault Models**: In addition to the Byzantine Fault Model, we could explore other types of fault models, such as crash faults or omission faults, and implement additional recovery mechanisms.

## 4. Other Aspects of the System

- **Concurrency**: The system makes use of goroutines to handle concurrent operations, allowing each node to send and receive messages asynchronously. This ensures that the system can handle multiple nodes efficiently.

- **Message Acknowledgment**: Each node acknowledges the receipt of a message, ensuring reliable communication in the system. This reduces the chance of lost messages and ensures that all nodes eventually receive the necessary information for decision-making.

- **Logging and Debugging**: Detailed logging is implemented for each node, which helps in tracking the system's behavior during the simulation. Logs capture the rounds, the orders being propagated, and the final decision made by each node.