Ahram Canadian University

BYZANTINE SYSTEM

2024

TA/ MAYAR ATEF

Team Members

Bakinam Mohamed	*42110396*
Aya Mohamed	*42110153*
Noha Elsayed	*42110440*
Salma abdrabo	*42110089*
Donia Salah	*42110426*

I. Introduction

Byzantine Fault Tolerance (BFT) is a concept in computer science that ensures a system continues to operate correctly even if some of its components fail in arbitrary or malicious ways. The term comes from the Byzantine Generals' Problem, a theoretical scenario that highlights the difficulties of achieving consensus in a distributed system with potentially treacherous participants.

The Byzantine Generals' Problem

Scenario: Multiple Byzantine generals camp around an enemy city. They need to agree on a common battle plan, but some generals might be traitors.

Challenge: How can loyal generals reach a consensus despite the presence of traitors trying to prevent agreement?

This problem is a metaphor for the challenges faced in distributed computing systems where some components might act incorrectly or maliciously.

How Does BFT Work?

Consensus Mechanism: BFT systems use specific algorithms to ensure all non-faulty components agree on the same state or value.

Fault Detection and Masking: These systems identify and mitigate the impact of faulty components.

Redundancy: Multiple components perform the same task to cross-verify results, ensuring system correctness even if some components fail.

Importance of BFT in Distributed Systems and Blockchain Networks

In Distributed Systems

Reliability: BFT ensures that distributed systems continue to function correctly even when some nodes fail or act maliciously.

Consistency: Helps maintain data consistency across different nodes, which is critical for applications like distributed databases.

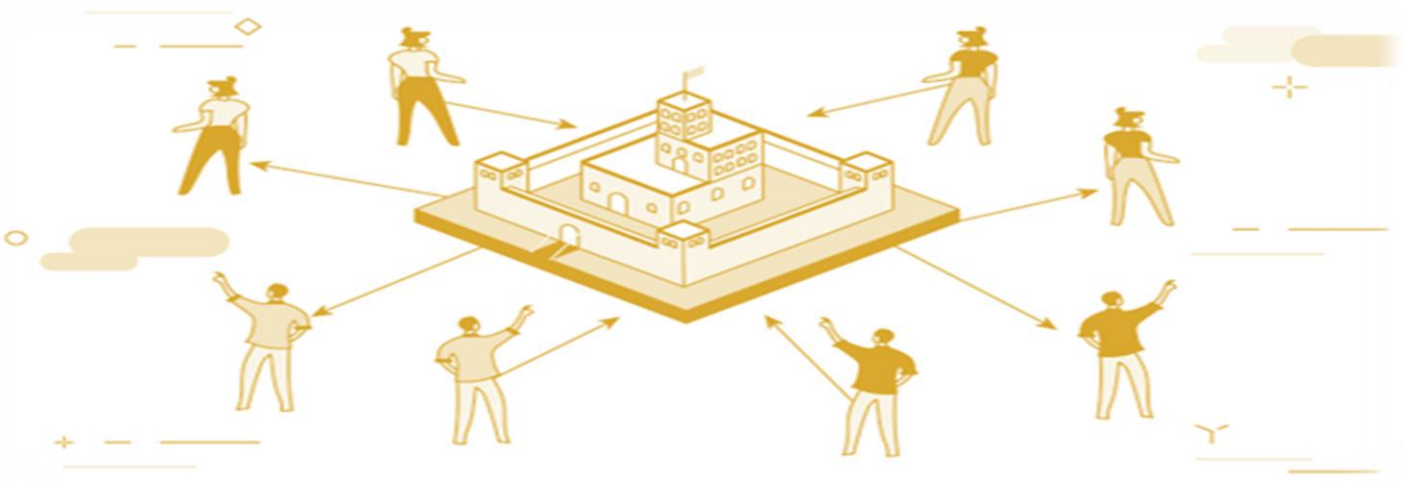
Security: Enhances security by preventing malicious nodes from disrupting the system's operations.

In Blockchain Networks

Consensus and Integrity: BFT is crucial for blockchain networks, where nodes must agree on the state of the ledger despite the presence of potentially malicious participants.

Decentralization: Enables secure and reliable operation in a decentralized environment, which is a core principle of blockchain technology.

Transaction Validation: Ensures that transactions are validated and recorded accurately, maintaining the integrity of the blockchain.



Difference Between Distributed Systems and Blockchain Systems

1. **Structure:**

Distributed Systems: No specific structure; data is stored in multiple locations.

Blockchain Systems: Data is stored in a chain of blocks, each linked to the previous one.

2. **Consensus:**

Distributed Systems: May use simple coordination methods; not always require consensus.

Blockchain Systems: Require robust consensus mechanisms like Proof of Work (PoW) or Proof of Stake (PoS).

3. **Trust:**

Distributed Systems: Often rely on trusted components or central authorities.

Blockchain Systems: Operate without a central authority, relying on cryptographic techniques for trust.

Why Distributed Systems and Blockchain Work Together?

1. **Enhancing Reliability:**

Distributed Systems: Provide redundancy by having multiple nodes perform the same tasks, increasing system reliability.

Blockchain: Adds an additional layer of security and reliability through its consensus mechanisms, ensuring data integrity.

2. Achieving Decentralization:

Distributed Systems: Distribute tasks across multiple nodes to avoid a single point of failure.

Blockchain: Uses decentralization to ensure no single entity controls the system, enhancing security and trust.

3. Ensuring Data Consistency:

Distributed Systems: Use replication and synchronization to keep data consistent across nodes.

Blockchain: Ensures consistency and immutability of data through its chain structure and consensus algorithms.

4. Improving Security:

Distributed Systems: Benefit from having multiple nodes that can detect and mitigate faults or attacks.

Blockchain: Uses cryptographic techniques and consensus to prevent unauthorized changes and ensure secure transactions.

Considerations

Byzantine Fault Tolerance is essential for building reliable and secure distributed systems. It plays a vital role in various high-stakes applications, particularly in blockchain networks, by ensuring consensus and correct operation even amidst complex and unpredictable failures.

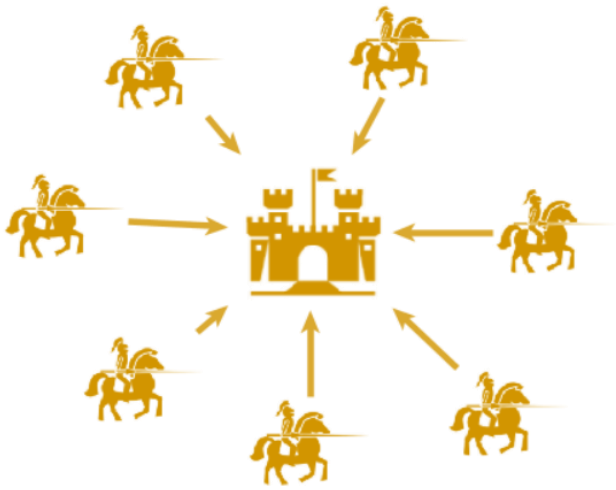
II. problems of Byzantine system

A Byzantine fault is a state of a computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed. It is also known as a Byzantine generals problem or Byzantine failure. The phrase gets its name from an allegory known as the "Byzantine generals problem", which was created to explain a scenario in which the system's participants must agree on a coordinated approach to prevent catastrophic system failure.

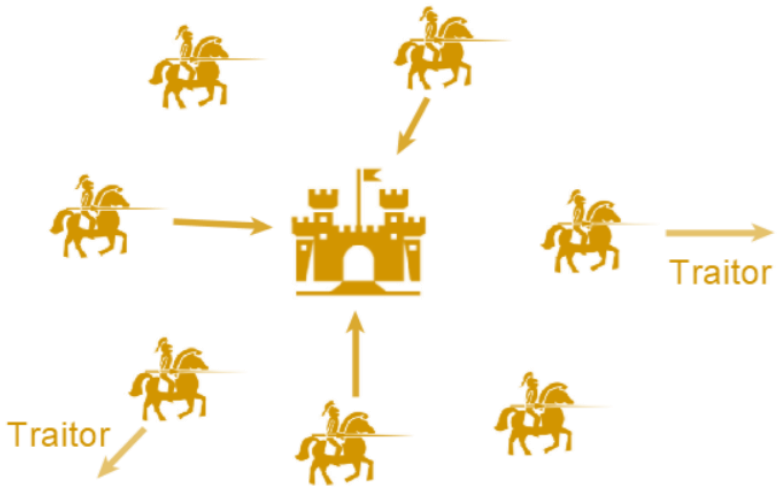
- ✓ **For an example,** A server may inconsistently appear to failure-detection systems as both malfunctioning and working during appear various symptoms to various observers in a Byzantine fault .It is a difficult for the other components to declare it failed and block it from the network since they first need to agree on which component actually failed. A fault-tolerant computer system's resistance to such circumstances is known as byzantine fault tolerance (BFT).

Consider a group of generals attacking a fort as an example of the flaw in its most basic form. The generals must determine whether to advance or retire as a group, some may favour advance while others favour retreat. The most important factor is that all generals come to an agreement on a strategy because a few generals' Random attack would result in a rout.

The issue is made more difficult by the existence of treacherous generals who might not only vote for a poor strategy, but also do so selectively. For an example, the ninth general may send a vote of attack to the remaining generals and a vote of retreat to the four generals who support attacking while the remaining four generals support retreating. While the remainder will attack those who obtained a vote to retreat from the ninth general will retreat. The generals' physical separation from one another and requirement to convey their votes via messengers who might not deliver them or might fabricate votes adds to the difficulty of the situation.



Coordinated attack leading to victory



Uncoordinated attack leading to defeat

Some Solutions:

1. Pioneering Solutions by Lamport, Shostak, and Pease (1982):

- Reduced the Byzantine Generals' Problem to a "Commander and Lieutenants" scenario.
- Found that the problem is solvable if there are fewer than one-third traitors ($n>3t$) and communication is synchronous.
- Illustrated with a scenario where a traitorous commander creates conflicting orders, making it impossible for lieutenants to determine the traitor.

2. Unforgeable Message Signatures:

- Digital signatures (public-key cryptography) can ensure Byzantine fault tolerance even with an arbitrary number of traitors.

3. Blockchain Application:

- Blockchain relies on consensus to validate transactions in a decentralized, trustless system.
- Consensus mechanisms and cryptographic protocols are essential for maintaining security and trust.

4. Byzantine Agreement in Distributed Systems:

- All non-faulty processors must agree on the same value.
- The goal is to protect against unpredictable failures and malicious attacks.

III. The algorithms &technique

Byzantine Fault Tolerance (BFT) is a property that ensures a distributed system continues to function correctly even if some of its components fail or act maliciously. This concept is crucial in environments where consensus and reliability are essential despite the presence of faulty or potentially malicious nodes.

➢ **First ,let know the difference between technique& algorithm:**

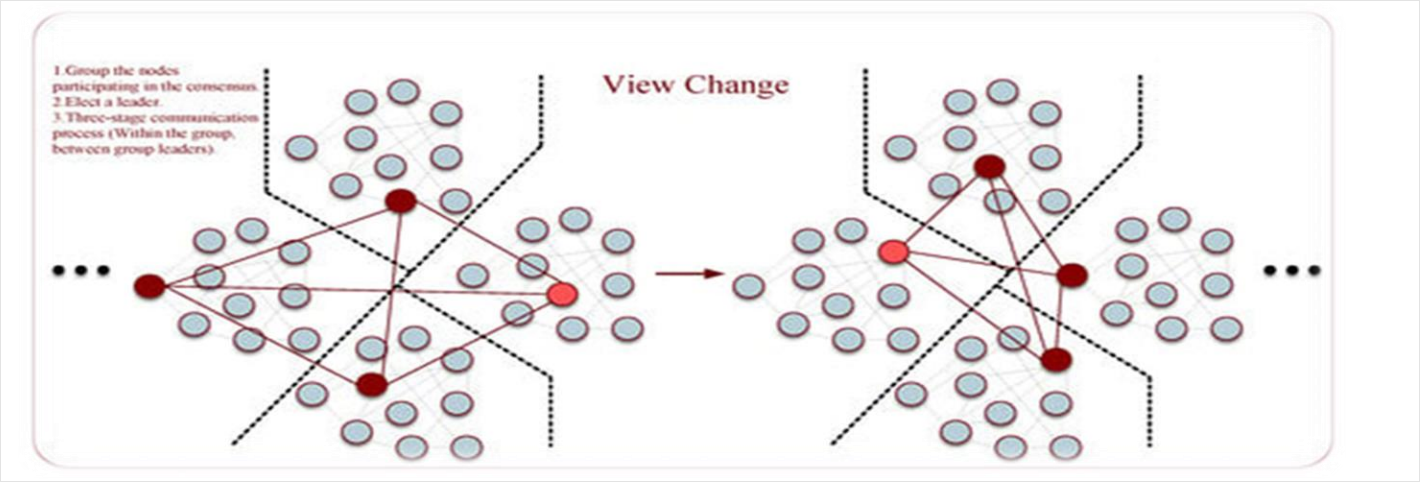
- A technique is a broader method or approach used to achieve a general goal. Techniques can encompass a variety of algorithms are applied to solve problems.

- An algorithm is a step-by-step procedure or a finite set of well-defined instructions to perform a specific task or solve a particular problem.

Techniques of (BFT):

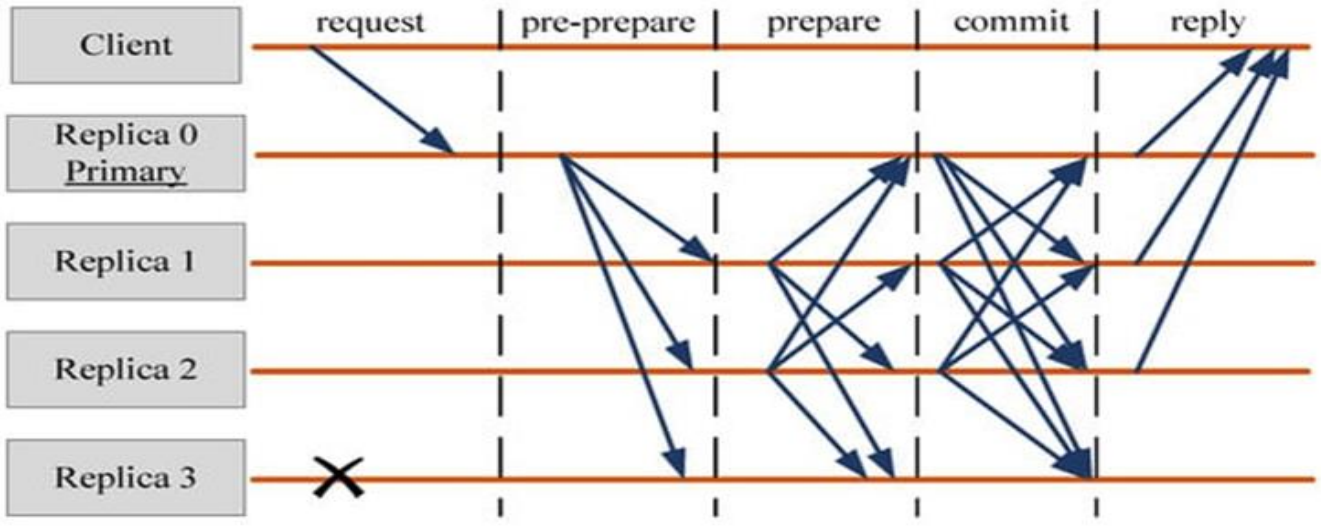
- **Redundancy:** A technique that involves duplicating critical components or functions to increase reliability. This could involve various algorithms to manage the redundant components.
- **Load Balancing:** A technique to distribute workload across multiple resources. It can use different algorithms like Round Robin, Least Connections, or Hashing.
- **Error Correction:** Techniques such as Forward Error Correction (FEC) involve using algorithms like Reed-Solomon or Hamming codes

Algorithms of (BFT):



➢ Practical Byzantine Fault Tolerance (PBFT):

- **Description:** One of the most widely used BFT algorithms, designed to provide high performance in practical systems.
- **Process:** Involves multiple phases (pre-prepare, prepare, commit) to ensure consensus among nodes. It tolerates up to f faulty nodes in a system of $3f+13f+1$ nodes.
- **Use Cases:** Blockchain systems, distributed databases.



➢ Federated Byzantine Agreement (FBA):

- **Description:** Used by systems like the Stellar blockchain. Nodes form quorums based on trusted sets.
- **Process:** Consensus is achieved through overlapping quorums of nodes that trust each other.
- **Use Cases:** Payment networks, decentralized networks.

➢ Tendermint:

- **Description:** A consensus algorithm used in blockchain systems that provides BFT.
- **Process:** Combines Byzantine Fault Tolerance with Proof-of-Stake mechanisms, using rounds of voting to achieve consensus.
- **Use Cases:** Blockchain platforms, such as Cosmos.

➢ Raft:

- **Description:** Although primarily designed for crash fault tolerance, extensions of Raft can handle Byzantine faults.
- **Process:** Simplifies consensus by dividing the process into leader election, log replication, and safety.
- **Use Cases:** Distributed databases, cluster management.

➢ HoneyBadgerBFT:

- **Description:** An asynchronous BFT protocol designed for high-performance and scalable systems.
- **Process:** Operates in an asynchronous environment, making it suitable for wide-area networks.
- **Use Cases**: Cryptocurrencies, secure multiparty computation.

➢ Algorand:

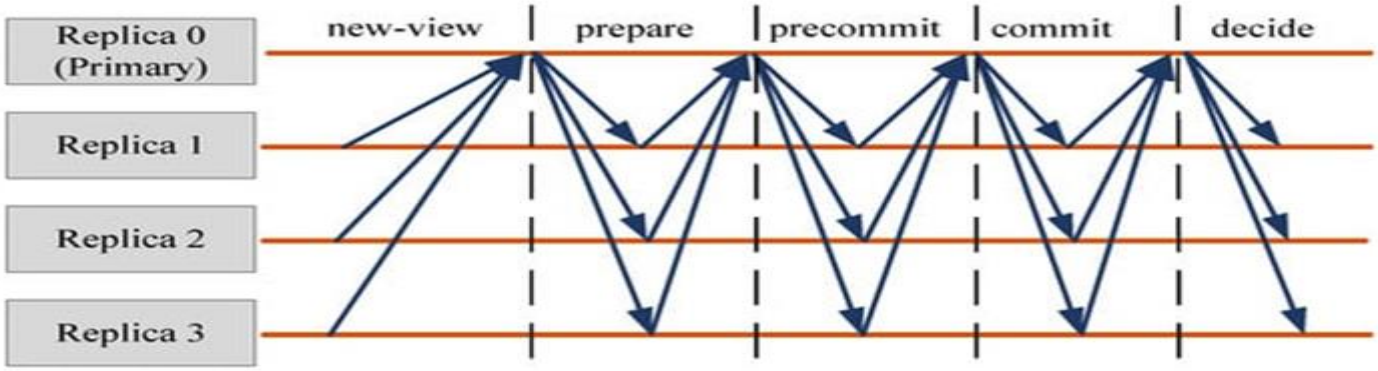
- **Description:** A consensus algorithm that achieves Byzantine agreement using a cryptographic sortition mechanism.
- **Process:** Randomly selects a small committee of nodes to propose and vote on blocks.
- **Use Cases:** Blockchain platforms, digital currencies.

➢ SBFT (Scalable Byzantine Fault Tolerance):

- **Description:** An optimized version of PBFT for better scalability.
- **Process:** Reduces the number of communication rounds and message complexity.
- **Use Cases:** Large-scale blockchain systems, high-throughput applications.

➢ HotStuff:

- **Description:** A BFT consensus protocol that improves the efficiency and simplicity of previous protocols.
- **Process:** Uses a linear communication complexity in the normal case, making it highly efficient.
- **Use Cases:** Blockchain systems, distributed databases.



➢ Chain-Based Byzantine Fault Tolerance (Chain-BFT):

- **Description:** Combines features of both chain-based consensus (like Nakamoto consensus) and BFT.
- **Process:** Utilizes a blockchain structure to achieve consensus with Byzantine fault tolerance.
- **Use Cases:** Hybrid blockchain platforms, secure voting systems.

Why These Algorithms and Techniques are Used?

- ✓ **Reliability:** BFT algorithms ensure that the system can continue to operate correctly even when some nodes are compromised. This is crucial for maintaining reliability and trust in distributed systems.
- ✓ **Security:** By addressing the challenges posed by malicious nodes, BFT enhances the security of the system, preventing attackers from disrupting consensus.
- ✓ **Decentralization:** BFT algorithms are essential for decentralized systems, where no single point of control exists, and consensus must be achieved collectively.
- ✓ **Scalability:** Techniques like quorum-based protocols and FBA allow BFT systems to scale efficiently, accommodating growth in the number of nodes without significant performance degradation.

➢ Practical Applications can apply these algorithms in it:

- **Blockchain and Cryptocurrencies:** BFT is a foundational concept in blockchain technology, ensuring that all participants agree on the state of the ledger despite the presence of malicious actors.
- **Distributed Databases:** Ensuring data consistency and reliability in distributed databases where nodes may fail or act maliciously.

IV. Main Results and the Tight Upper Bound on Fault Tolerance

In 1982, Leslie Lamport, Robert Shostak, and Marshall Pease published a seminal paper titled "The Byzantine Generals Problem."

The Byzantine Generals Problem can be summarized as follows:

- A group of Byzantine generals, each commanding a portion of the army, must agree on a common plan (attack or retreat) to ensure a coordinated action.

- Some generals may be traitors and attempt to prevent the loyal generals from reaching a consensus by sending conflicting or false messages.

Lamport et al. made the following assumptions for their analysis:

1. **Synchronous Communication****: The system operates synchronously, meaning messages are delivered within a known, bounded time.

2. **Reliable Communication**: Messages between generals are delivered reliably without corruption.

3. **Authentication** (in some scenarios) Messages can be authenticated to ensure they are not forged.

Main Results and the Tight Upper Bound on Fault Tolerance

The most critical result from Lamport et al.'s paper is the establishment of the tight upper bound on fault tolerance in a distributed system. The key finding is:

Consensus is achievable if and only if $n \geq 3f+1$, where n is the total number of generals and f is the maximum number of traitors.

This bound means that to tolerate f faulty generals, the total number of generals must be at least $3f+1$

Examples

1) Traitor: To tolerate 1 traitor (f=1), the system needs at least 4 generals (n=4).
2) Traitors: To tolerate 2 traitors (f=2), the system needs at least 7 generals (n=7).
3) Traitors: To tolerate 3 traitors (f=3), the system needs at least 10 generals (n=10).

This result demonstrates that the proportion of traitors must be less than one-third of the total number of participants in the consensus process. If this condition is met, the system can guarantee that the loyal generals will reach an agreement despite the presence of malicious actors.

Significance of the Result

The $n \geq 3f+1$ bound is significant because it provides a concrete and rigorous criterion for designing fault-tolerant distributed systems. It shows the necessity of redundancy and the extent to which a system must be over-provisioned to ensure reliability.

This finding has profound implications for various applications, including:

Distributed Databases: Ensuring consistency and reliability across multiple replicas.

Blockchain Technology: Achieving consensus in decentralized networks.

Fault-Tolerant Systems: Designing systems that can withstand arbitrary (Byzantine) faults.

Theoretical Bounds on Fault Tolerance Under Modified Assumptions

In the Byzantine Generals Problem, the classical result by Lamport et al. establishes that

$n \geq 3f+1$ is the necessary and sufficient condition for achieving consensus in the presence of f Byzantine faults in a synchronous system with reliable communication. However, this bound can be explored and modified under different assumptions.

1. Authentication and Cryptographic Techniques

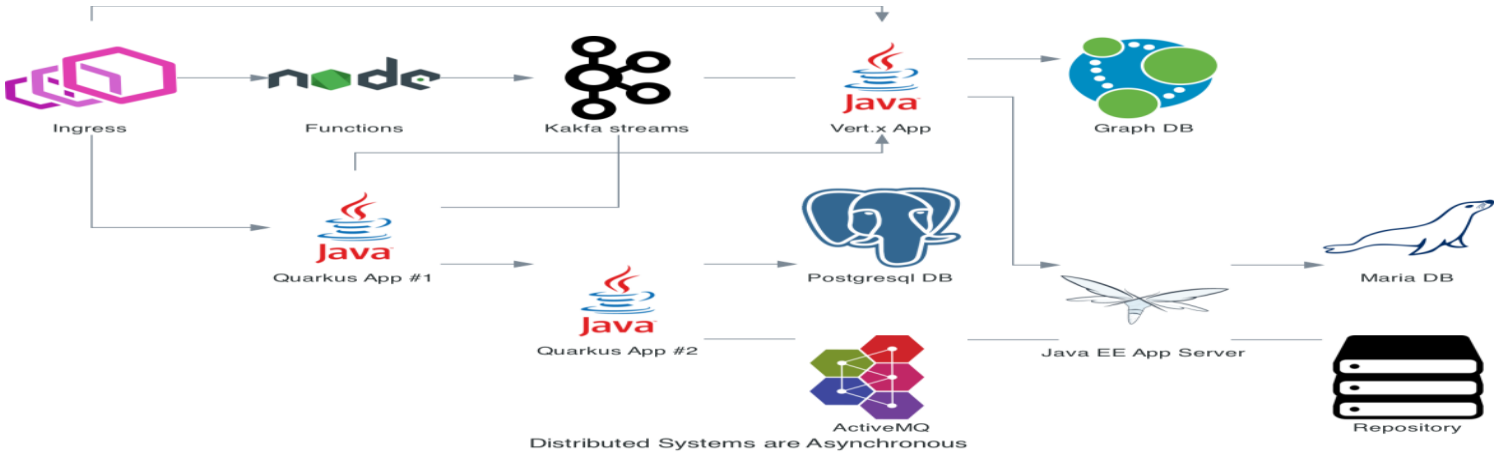
When messages can be authenticated using digital signatures, the assumption is that a message purportedly from a specific sender can be verified to be genuine and unaltered.

✓ Authenticated Byzantine Agreement: With authenticated messages, the bound can be relaxed. It is possible to achieve consensus with $n \geq 2f+1$ under these conditions because authentic messages prevent impersonation by Byzantine nodes.

2. Asynchronous Systems

In an asynchronous system, there are no guarantees on message delivery times. This means that messages can be delayed indefinitely, leading to potential issues in achieving consensus.

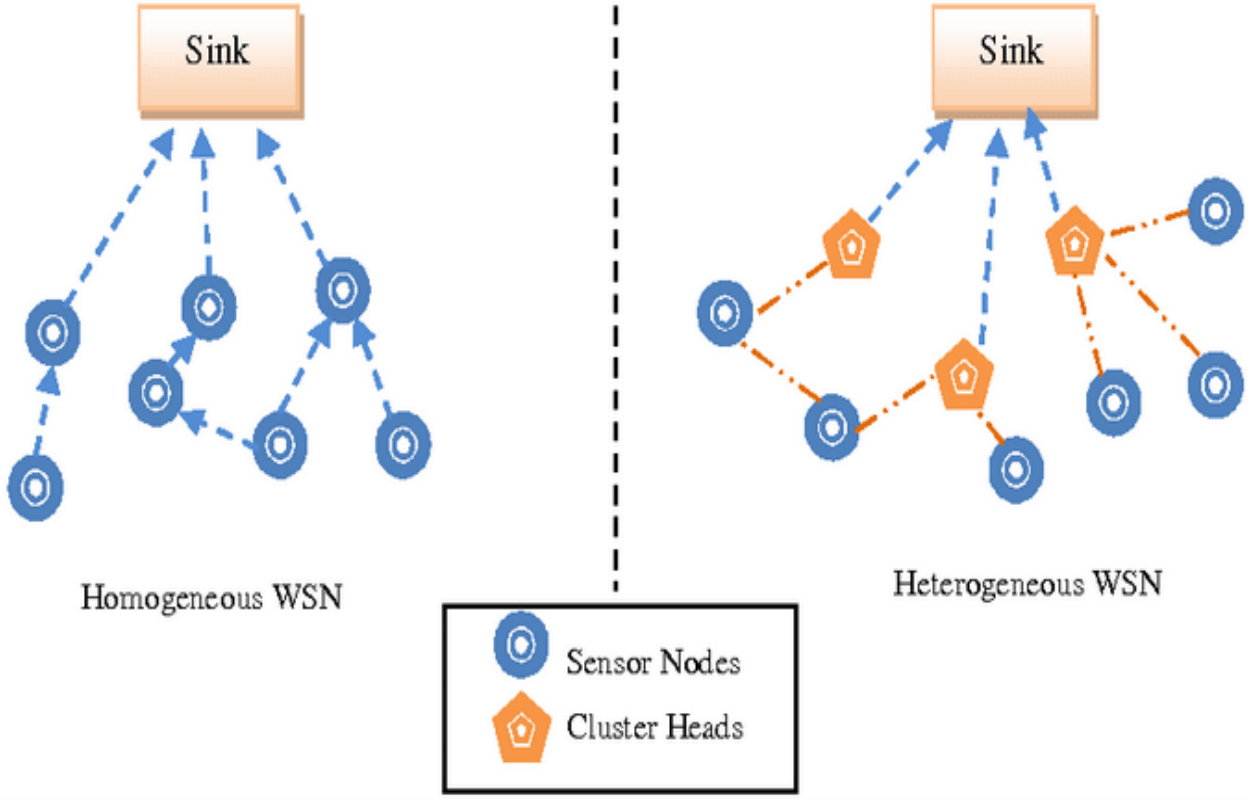
✓ Fischer-Lynch-Paterson (FLP) Impossibility Result: The FLP theorem states that in a completely asynchronous system, it is impossible to achieve consensus if even a single node can fail (Byzantine or otherwise). This highlights the fundamental difficulty of consensus in asynchronous systems without additional assumptions.



3. Homogeneous vs. Heterogeneous Nodes

When nodes have different capabilities or roles, the fault tolerance bounds can be adjusted.

- ✓ Bound: In a system where nodes have varying roles (e.g., some are more trusted than others), the consensus protocol can be designed to leverage this hierarchy, potentially reducing the required number of nodes. However, precise bounds would depend on the specific protocol and the role definitions.



V. BFT Applications

Due to the importance of the BFT in dealing with Byzantine errors, these are some of the areas in which the BFT is used:

1. Computing:

- BFT mechanisms use components that repeat an incoming message (or just its signature) to other recipients of that incoming message.
- All these mechanisms make the assumption that repeating a message blocks the propagation of Byzantine symptoms.
- For systems that have a high degree of safety or security criticality, these assumptions must be proven to be true to an acceptable level of fault coverage(effeciancy).
- When providing proof through testing, one difficulty is creating a sufficiently wide range of signals with Byzantine symptoms. Such testing will likely require specialized fault injectors.

2. Aviation:

- Some aircraft systems, such as [the Boeing 777 Aircraft Information Management System (via its ARINC 659 SAFEbus network), the Boeing 777 flight control system, and the Boeing 787 flight control systems] use BFT; because these are real-time systems, their BFT solutions must have very low latency.
- For example: SAFEbus can achieve BFT within the order of a microsecond of added latency. The SpaceX Dragon considers Byzantine fault tolerance in its design.

3. Blockchain Technology:

- BFT algorithms are widely used in blockchain networks to achieve consensus among a decentralized network of nodes.
- They help ensure the integrity of transactions and prevent double-spending of digital assets.
- Examples of blockchain networks using BFT algorithms : Hyperledger, Stellar, Cosmos, and Ripple.

4. Finance and Banking:

- BFT algorithms are critical for ensuring the security and reliability of financial transactions in decentralized systems.
- They help reduce the risk of fraudulent activities and unauthorized transactions, making them ideal for applications such as: cross-border payments, asset tokenization, and smart contracts.

5. Internet of Things (IoT):

- BFT algorithms can be used to secure communication and data exchange among IoT devices in a distributed network.
- They help prevent malicious actors from manipulating sensor data or disrupting the operation of connected devices, ensuring the integrity and reliability of IoT applications.

6. Cloud Computing:

- BFT algorithms can be employed in cloud computing environments to ensure the availability and consistency of data across multiple servers.
- By utilizing BFT algorithms, cloud service providers can enhance the fault tolerance and reliability of their infrastructure, reducing the risk of data loss or downtime.

7. Supply Chain Management:

- BFT algorithms can be applied in supply chain management systems to verify the authenticity and provenance of products as they move through the supply chain.
- By utilizing BFT algorithms, organizations can ensure transparency and trust among stakeholders, reducing the risk of counterfeiting and fraud.

8. Voting Systems:

- BFT algorithms can be used to secure online voting systems and ensure the integrity of election results.
- By leveraging BFT algorithms, voting platforms can prevent tampering with votes and guarantee the accuracy and fairness of the electoral process.

VI. Conclusion

The Byzantine fault tolerance system is a crucial aspect of ensuring reliability and security in distributed networks, particularly where nodes may fail or act maliciously. By addressing Byzantine faults, systems can maintain consensus and continue functioning correctly even in the presence of arbitrary failures. This resilience is vital for applications requiring high reliability and trustworthiness, such as blockchain technology, financial systems, and critical infrastructure. Ultimately, implementing robust Byzantine fault tolerance mechanisms helps create systems that are not only fault-resistant but also capable of sustaining operations under adverse conditions, thereby reinforcing the integrity and robustness of the network.

Practical Byzantine Fault Tolerance