A DISTRIBUTED OPERATING SYSTEM FOR PERMISSIONED BLOCKCHAINS

HYPERLEDGER FABRIC

BYZANTINE GENERALS' PROBLEM

- An agreement problem
- A group of generals wants to attack a city
- They must to know when to attack or when to retreat
- They cannot see or hear each other
- What do they need? –> A PLAN
- We should build a network (nodes, communication channels, protocols)

BYZANTINE GENERALS' PROBLEM

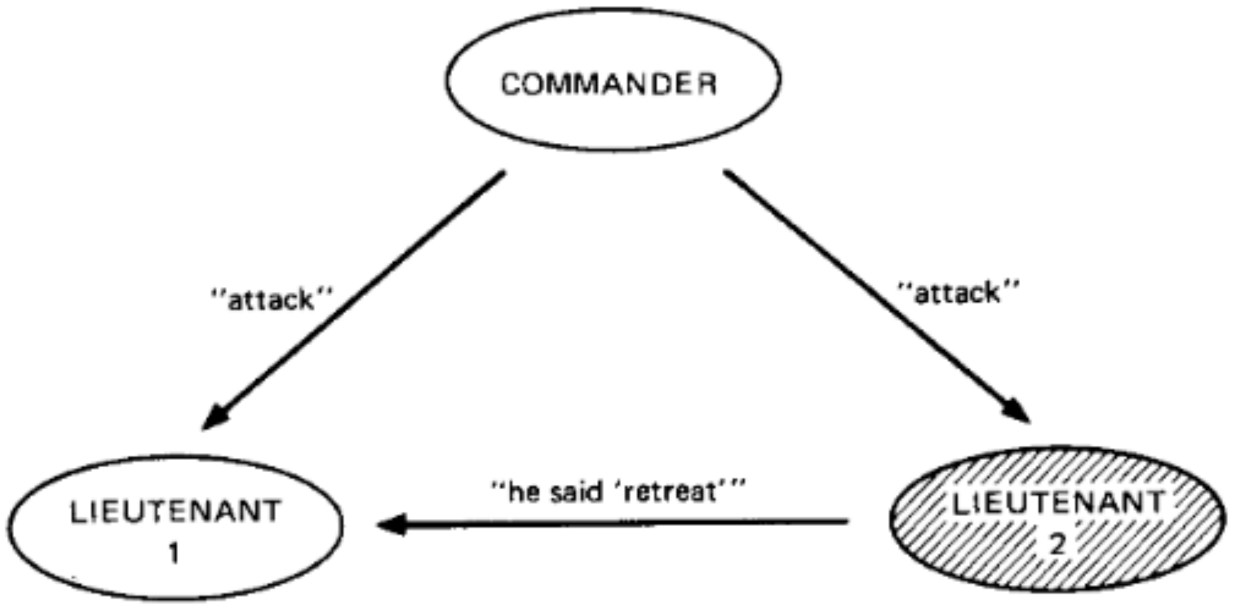

Fig. 1. Lieutenant 2 a traitor.

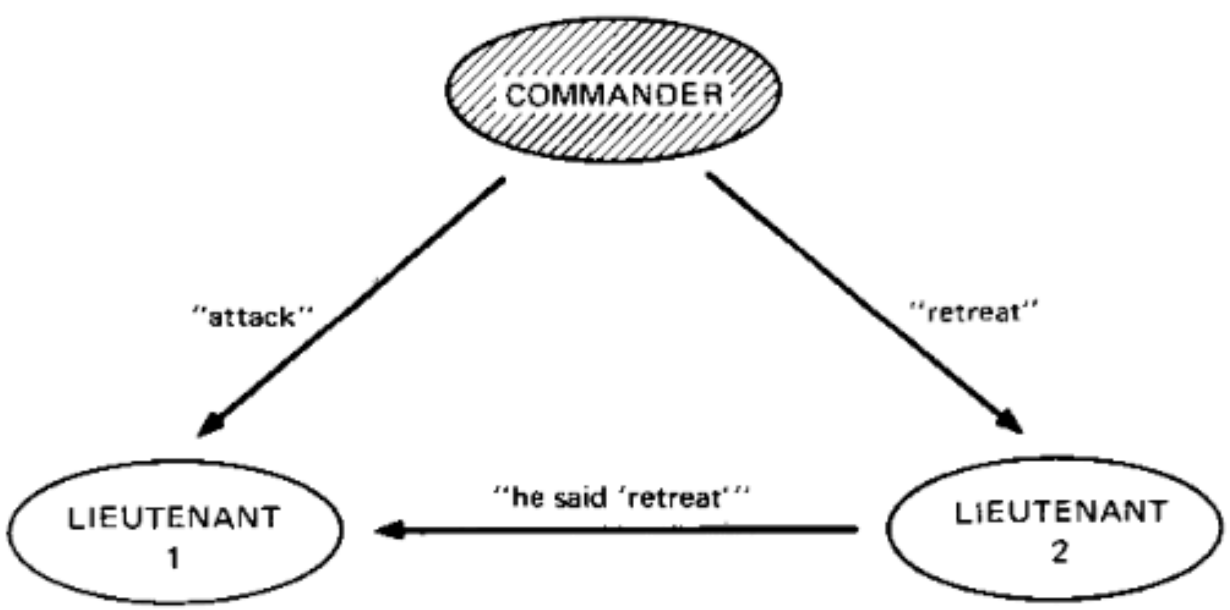

Fig. 2. The commander a traitor.

… take me to your commander

BYZANTINE GENERALS' PROBLEM IN PRACTICE

- Aircraft systems (Boeing 777 and 787 flight control systems)
- Spacecrafts (SpaceX Dragon flight system)
- Bitcoin

BLOCKCHAINS

- Simply, a blockchain is a huge ledger for recording transactions
- It is maintained within a distributed network of mutually untrusting peers (the generals)
- Every peer maintains a copy of the ledger
- They validate and order the transactions through a consensus protocol
- Blockchains have emerged with Bitcoin

BLOCKCHAINS

- In a public (*permissionless*) anyone can participate without a specific identity
- *Permissioned* blockchains run a blockchain (of course) among a set of *known*, *identified* participants. This is a way to secure the interactions among a group of entities that have a common goal but which do not fully trust each other
- Blockchains may execute arbitrary, programmable transaction logic in the form of smart contracts (as exemplified by Ethereum)

SMART CONTRACTS

- A smart contract functions as a trusted distributed application and gains its security from the blockchain and the underlying consensus among the peers
- Many existing smart contracts blockchains follow the blueprint of State-Machine Replication and implement so-called active replication: first, the transactions are ordered and propagated to all peers and second, each peer executes the transactions sequentially

SMART CONTRACTS

- Prior permissioned blockchains suffer from many limitations:
  - Smart contract must be written in a *fixed, non-standard, or domain-specific language*
  - The sequential execution of transaction by all peers *limits performance*
  - Transaction must be *deterministic*, which can be difficult to be ensured programmatically
  - Every smart contract runs on *all peers*, which is at odds with *confidentiality*

OK, LET'S SEE FABRIC

- Fabric introduces a new blockchain architecture aiming at *flexibility, scalability and confidentiality*
- Fabric support the execution of distributed applications written in *standard programming languages*
- Fabric is the first *distributed operating system* for permissioned blockchains

FABRIC

- The architecture of Fabric follows the *execute-order-validate* paradigm

FABRIC

- This design departs radically from the *order-execute* paradigm
- It combines the two approaches to replication, *passive* and *active*:
  - Every transaction is executed (endorsed) only by a subset of peers, which allows for parallel execution (passive)
  - The transaction's effects on the ledger state are only written after reaching consensus of a total order among them (active)

FABRIC

- This *hybrid replication design*, which mixes passive and active replication in the Byzantine model, and the *execute-order-validate* paradigm, represent the most innovation in Fabric architecture

ORDER-EXECUTE ARCHITECTURE

- All previous blockchain systems follows *order-execute architecture*
- Let's take Ethereum for example:
    1. every peer assembles a block containing valid transactions
    2. the peer tries to solve the puzzle
    3. if the peer is *lucky* and solves the puzzle it disseminates the block to the network
    4. every peer receiving the block validates the solution *and* all transaction in the block
- Simply said, every peer repeats the execution of the lucky peer from its first step
- If this is not enough, all transactions must be executed sequentially

LIMITATIONS OF ORDER-EXECUTE ARCHITECTURE

- Sequentially execution
  - It limits the effective throughput that can be achieved
  - Since the throughput is inversely proportional to the execution latency, this may become a performance bottleneck
  - A *Denial-of-Service* attack could simply introduce smart contracts that take very long time to execute

LIMITATIONS OF ORDER-EXECUTE ARCHITECTURE

- Non-deterministic code
  - This is usually addressed by programming blockchains in domain-specific languages (e.g. Ethereum Solidity) that are expressive enough for their applications but limited for deterministic execution
  - Only one non-deterministic contract created with malicious intent is enough to bring the entire blockchain to a halt

LIMITATIONS OF ORDER-EXECUTE ARCHITECTURE

- Confidentiality of execution
  - Many permissioned systems run all smart contract on all peers
  - However, many intended use cases for permissioned blockchains require confidentiality