Byzantine Fault Tolerance in a Distributed System

- Byzantine Faults
- Byzantine General’s Problem
Byzantine General’s Problem

(Lamport had to pick a group that would not offend anybody today)

- A Byzantine Fault is an incorrect operation (algorithm) that occurs in a distributed system that can be classified as:
  
  Omission Failure – a failure of not being present such as failing to respond to a request or not receiving a request.
  
  Execution Failure or Lying – a failure due to sending incorrect or inconsistent data, corrupting the local state or responding to a request incorrectly.

- Examples
  
  Round off errors passed from one function to another and then another, etc.
  
  Corrupted system databases where the error is not detected
  
  Compiler errors
  
  An undetected bit flip producing a bad message

- This is a worse case model since the Byzantine Fault can generate misleading information causing a maximum of confusion.
• The Byzantine General’s Problem is an illustrative example of Byzantine Faults

• A number of different armies want to besiege an enemy city. Success requires agreement on a common plan amongst the disbursed army; communication is only by messengers.

  Complication: There may be traitors who send out conflicting messages to sow confusion.

  Challenge: Find an algorithm that ensures the armies come to an agreement and attack at the same time.

• Real World Relationships

  Generals → processors

  Traitors → faulty processors or faulty system components (including software)

  Messengers → processor communications/system data bus
Byzantine General’s Problem Scenario

Several armies, each under the command of a general, are camped outside a city which they plan to attack.

One of the generals will issue the order *attack* or *retreat*.

One or several of the generals may be traitors. To win the battle all loyal generals must attack at the same time.

The traitors will attempt to fool the loyal generals so that not all of them attack at the same time.

To stop the traitor’s malicious plan, all generals exchange messages directly with each other.

**Problem**

How do replicated units reach agreement on a non-replicated value in the presence of malicious faults? Simple voting algorithms cannot handle the ‘malicious’ faults.
Impossibility to reach agreement on sensor values

Voting algorithm: Use median value.
Computer C sends value = 9 to Computer A and value = 12 to Computer B
A uses $median\{9, 10, 11\} = 10$, B uses $median\{10, 11, 12\} = 11$

C’s malicious behaviour causes A and B to work with inconsistent values.
Solution*

To reach agreement between the loyal generals in the presence of \( m \) faults

- There must be at least \( 3m + 1 \) processors to deal with \( m \) faults (traitors)
- Each processor must be connected to each other through at least \( 2m + 1 \) communication paths
- \( m + 1 \) rounds of messages must be exchanged
- The processors must be synchronized within a known skew of each other

Such a system is called **Byzantine Resilient** which is a fully fault tolerant system.

* for Unprotected or Oral messages versus Signed messages
Tolerating One Byzantine Fault

Unit 2 uses $v'_1 = \text{majority}\{v_1, v_{31}, v_{41}\}$

Number of messages for agreement on one value is: $3 + 2 \cdot 3 = 9$ messages

A majority decision involving 4 units (e.g. 4 sensor values) requires: $9 \cdot 4 = 36$ messages
Tolerating Two Byzantine Faults

Unit 2 uses $v'_1 = \text{majority}\{v_1, v'_3, v'_4, v'_5, v'_6, v'_7\}$

$v'_4 = \text{majority}\{v'_4, v'_34, v'_54, v'_64, v'_74\}$

Number of messages for agreement on one value is $6 + 6(5 + 5.4) = 156$ messages