Lecture 11

Byzantine Fault Tolerance

ECE 422: Reliable and Secure Systems Design

UNIVERSITY OF ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- Byzantine failure
- The Two Generals Problem
- Next class: Byzantine General problem

Example on encoding

Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: For each entry in the data (e.g., 0001, 1001):

- Step 1: convert it into a data polynomial
- Step 2: solve the codeword polynomial multiplication

Hint: use the circular shifting property to calculate the codeword polynomial instead. It is much faster.

Example on encoding

Suppose $g(x)$ = $\boxed{(1+x+x^3)}$ for a (7,4) cyclic code

No shifting, 1 shifting, 3 shifting

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: Using the circular shifting property, compute for d(x)·g(x)

For data = {0001000}, d(x)·g(x) = 0001000 + 0000100 + 0000001 = 0001101

Summary of cyclic code

- Any circular shift of a codeword produces another codeword.
- Code is characterized by its generator polynomial g(x), with a degree (n-k), where n = bits in codeword, k = bits in data.
- All calculations are done in mod 2 arithmetic.
  - Multiplication of polynomial for encoding
  - Division of polynomial for decoding
- Cyclic code detects all single errors and all multiple adjacent error affecting (n-k) bits or less.
  - It does not correct the error.

Information redundancy

Information redundancy tolerate faults by adding information to the original data.

- Tolerate faults by means of coding
- Avoid unwanted information changes
- E.g., information loss during data storage or transmission

The term “coding” is used in the context of communication and data storage

Software fault tolerance

Fault tolerance is the ability for systems to continue functioning as a whole despite the faults

- Example of fault tolerance: while a PDF reader may crash when editing a corrupted PDF file, the PDF reader will restart itself after saving the information

The system contains a fail-stop failure.

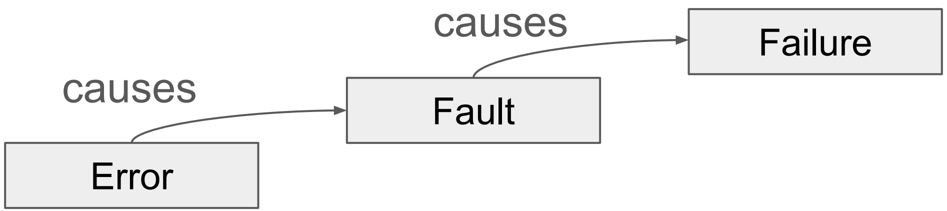
- Upon a fault, the system stops.
- To tolerate this type of behavior:
  - Solution 1: Restart the system (e.g., resetting the states)
  - Solution 2: Failover to redundancy

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- Byzantine failure
- The Two Generals Problem
- Next class: Byzantine General problem

Failure

Failure occurs when the system fails to perform its required function.



A failure is always related to a required function

- Based on the system specifications and functional requirements
- E.g., according to the user requirement, the maximum response time of a server shall be no longer than 15 seconds.
- A failure occurs when the response time exceeds 15 seconds.

Failure

A failure is an event that occurs at a specific point in time.

- It is not always possible to observe when a failure happens
- E.g., server performance degradation
- E.g., dormant faults in codes

The failure may:

- Come in different forms
- Originate from different kinds of faults
- Have different consequences on the system

Failure classification

Failure classification describes the way how a system can fail that is perceived by the rest of the system.

- It aims to answer one question: "What kind of failure are we dealing with when considering the whole system into account."
- It helps understand how to build and design for fault-tolerant systems.

There are four types of failure:

- Timing failure, also known as performance failure
- Omission failure
- Crash failure
- Arbitrary failure, also known as Byzantine failure

Timing failure

Timing failure happens when the system delivers correct results, but outside the expected time interval.

- Failure lies in the amount of time the system took to execute a task
- Not in the results of the task itself

Example: Delayed response from a server

- The server must responds between 5 milliseconds and 15 seconds
- A response exceeds our expected time interval
- This produces a timing failure

Omission failure

Omission failure happens when the system never appears to respond.

- Can also be understood as “infinitely late” timing failure

There are two forms of omission failure:

- Send omission failure: fail to send a response
- Receive omission failure: fail to receive a response

Example: No response to incoming requests

- The client sends a request
- The server fails to respond to the request
    - Failed to send messages
    - Failed to receive messages

Crash failure

Crash failure happens when the system crashes, but is working correctly until it crashes.

- The system experiences omission once and becomes completely unresponsive (i.e., crashes)

Example: Server crashes

- The client sends a request
- The server experiences an omission failure, then fails and stops responding,

Byzantine failure

Byzantine failure happens when the system sends arbitrary responses at arbitrary times.

- Derived from Byzantine faults
- The system responds with different (erroneous) responses each time, and is inconsistent in what kind of response it delivers

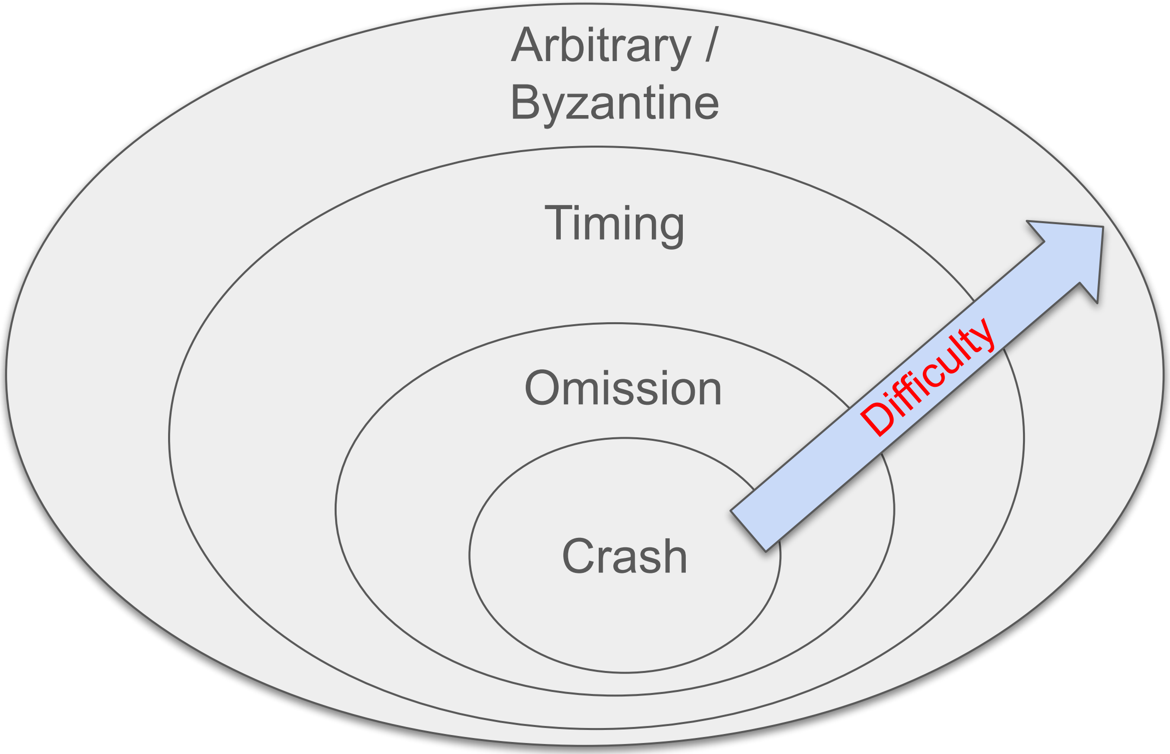
Example: Arbitrary response to the same request

- The client A sends a request
- The server responds with a message A
- The client B sends the same request as client A
- The server arbitrarily responds with a message B

Examples of failure

Type of failures	Example of server's response
Crash failure	Valid responses until it crashes
Omission failure	No response to incoming requests
Timing failure	Delayed response, outside of a specified time interval
Arbitrary failure	Arbitrary response at arbitrary times

Difficulties of failures



Halting failures

Failure classification deal with the situation that we no longer perceive any actions from the system.

- Can we conclude that a system has come to a halt?
- How do we distinguish between crash/omission/timing failure?

There are five categories of halting failures on the reliability of failure detection (from the least to the most severe):

- Fail-stop
- Fail-noisy
- Fail-silent
- Fail-safe
- Fail-arbitrary

Types of halting failures

Let a *client* attempt to detect that the *server* has failed.

Fail-stop: failures that can be reliably detected.

- Crash failure, but reliably detectable
- Assuming that:
  - Non-faulty communication exists between the client and the server
  - There is a worst-case delay on the response from the server.
- E.g., Server stops and the client can detect this failure.

Fail-noisy: failures that are eventually reliably detected

- Crash failure, but eventually reliably detected
- The client will only eventually come to the correct conclusion that the server has crashed.
- E.g., Server's response time slows down, and eventually fails.

Types of halting failures

Fail-silent: failures that cannot be distinguished between crash and omission

- Omission or crash failures
- Assuming that:
  - Non-faulty communication exists between the client and the server
  - The client cannot distinguish crash failures from omission failures.
- E.g., No response from the server.

Fail-safe: failures that cannot do any harm.

- Arbitrary, yet benign failures
- The server fails without doing any ham
- E.g., Server slows down, but remains available.

Types of halting failures

Fail-arbitrary: arbitrary, with malicious failures

- Server may fail in any possible way; failures may be unobservable.
- Arbitrary where a server is producing output that it should never have produced, but which cannot be detected as being incorrect.
- E.g., The server may be working with other servers to produce intentionally wrong answers; it still responds to the client.

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- Byzantine failure
- The Two Generals Problem
- Next class: Byzantine General problem

Byzantine failure

Byzantine failure: a node/component may fail arbitrary due to:

- Exhausted resources
  - E.g., null response from the server due to resource exhaustion
- Conflicting information from different parts of the system
  - E.g., Malicious attack providing conflicting information
- … and more

Byzantine failure

Byzantine failure: a node/component may fail arbitrary due to:

- Exhausted resources
- Conflicting information from different parts of the system

Why would nodes/components fail arbitrarily?

- Software bugs in the code
- Hardware failures
- Malicious attack on the system

What makes Byzantine failure important?

Byzantine failure provides an abstraction of issues in real-world applications

Example 1: Bitcoin

- Byzantine failure as the results of malicious nodes in the network
- A malicious node can cause failure in the network

Example 2: Aircraft flight control system (e.g., Boeing 777, 787)

- Byzantine failure as the results of potential hardware failures
- Hardware components can fail arbitrarily based on the temperature at which they operate

Solution? Designing Byzantine fault-tolerant systems for reliability

The Two Generals Problem

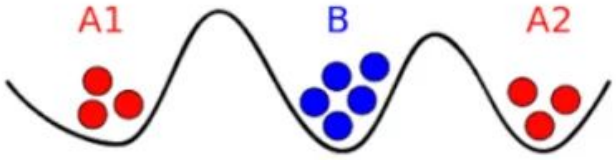
The two generals problem is a consensus problem where two Generals (General A1 and General A2) need to coordinate attack on the army B.

- If both Generals attack at the same time, they win the battle.
- If neither Generals attack, they will try to coordinate again the next day.
- But if only one of the two Generals attacks, they lose the battle.

There is only one way to communicate between the two Generals:

- By sending messengers on a high-risk path through enemy lines.

What messages can the generals send to reach a consensus on whether or not to attack?



The Two Generals Problem

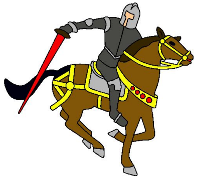
Let General A1 be the one who decides to attack the next morning.

- Send the message to General A2 about the attack.
- Make sure that General A2 got the message.

What kind of message can General A1 send to General A2 to make sure that a consensus is reached?

Solution: TCP 3-way handshake

A1



A2

Scenario 1: If the message was delivered to A2, everything is fine.

A1 wants to attack

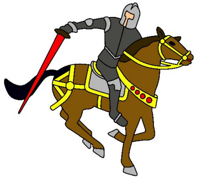
If you respond, I'll attack!

A1 will attack;
B wants to attack

Chris Colohan's talk illustrated this slide.

Solution: TCP 3-way handshake

A1



A2

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

Scenario 2: If the messenger was captured by B, A1 will not receive any response from A2, they will not attack.

Solution: TCP 3-way handshake

A1

A2

Scenario 1: If the message was delivered back to A1, everything is fine.

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

Solution: TCP 3-way handshake

A1

A2

Scenario 2: If the messenger was captured by B, A2 will not receive any response from A1, they will not attack.

A1 wants to attack

If you respond, I'll attack!

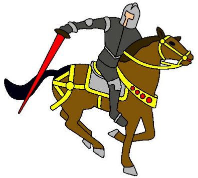
A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

Solution: TCP 3-way handshake

A1



A2

Scenario 1: If the message was delivered back to A2, everything is fine. They will both attack.

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

We'll attack!

A1 will attack;
A2 will attack

Solution: TCP 3-way handshake

A1

A2

Scenario 2: If the message was not delivered back to A2, A1 is committed to attack but A2 isn't.

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

We'll attack!

A1 will attack;
A2 will attack

Solution: TCP 3-way handshake

A1

A2

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

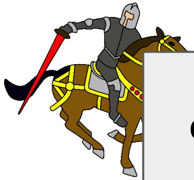
We'll attack!

A1 will attack;
A2 will attack

There is no solution to the two general problem.

Solution: TCP 3-way handshake in practice

A node in a distributed system



Untrusted communication channel

A node in a distributed system

A1 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

A1 will attack;
A2 will attack

We'll attack!

A1 will attack;
A2 will attack

Assumption: Failure is not 100% Byzantine.

The Two Generals Problem

The two generals problem is a classic computer science problem that remains unsolvable.

- It remains unsolvable as there is a need to be a last acknowledgement from General A1 to A2.
- And this starts a never-ending cycle of acknowledgement as the message may get lost.
- This is why the problem is unsolvable.

Next class: The Byzantine Generals Problem