THE UNIVERSITY of EDINBURGH

Distributed Systems
Fall 2024

Yuvraj Patel

THE UNIVERSITY *of* EDINBURGH
Careers Service

Welcome to Discover Careers

1 to 3 October 2024, 1pm to 6pm

120 employers
Different organisations each day, across two floors

https://edin.ac/45e5DiL

Sponsored by

goodmoves
Charity and voluntary sector jobs

NatWest Group

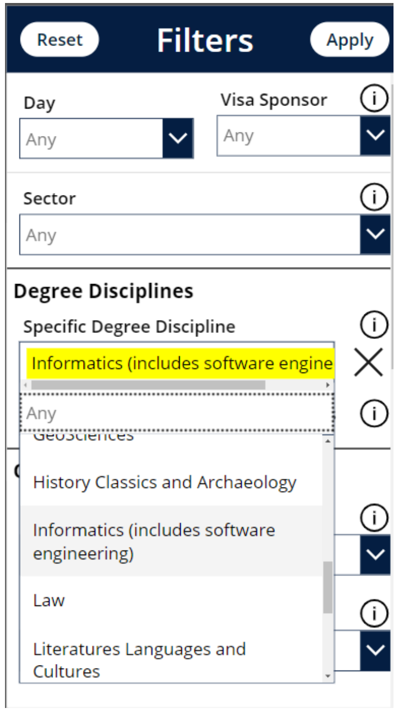
Deloitte.

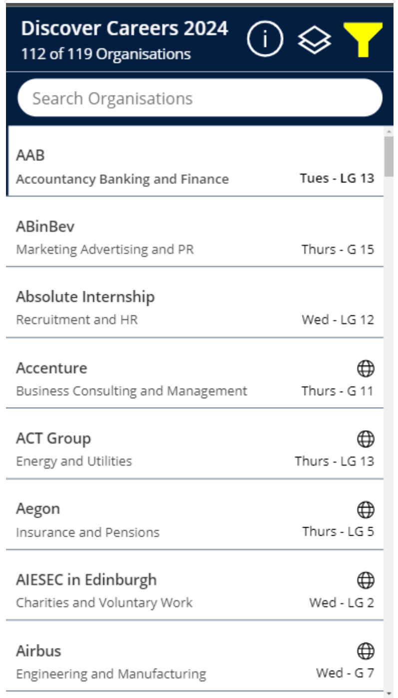
Inspiring futures

Find out more on MyCareerHub

Why should you go?

- 112 of 119 Employers are specifically recruiting for Informatics Degree Disciplines (check the event guide)
- 94 Organisations have open graduate roles, 70 recruit for interns
- Talk to recruiters about: Jobs, Workplace culture, Assessment Process, Tips to make you stand out
- Need Help? Careers Service Staff on hand





Today’s Agenda

Fault Tolerance

- What is a fault?
- What are the system models w.r.t faults?
- Why fault tolerance and how to provide it?

Dependability Requirements

Four main requirements

- Availability – Readiness for usage
- Reliability – Continuity of service delivery
- Safety – Very low probability of catastrophes
- Maintainability – How easy can a failed system be repaired

Reliability vs. Availability

Reliability R(t) of a component C

- Conditional probability that C has been functioning correctly during [0, t) given C was functioning correctly as time T = 0

Traditional metrics

- Mean Time To Failure (MTTF): The average time until a component fails
- Mean Time To Repair (MTTR): The average time needed to repair/replace a component
- Mean Time Between Failures (MTBF): MTTF + MTTR

Reliability vs. Availability (contd...)

Availability A(t) of component C

- Average fraction of time that C has been up-and-running in interval [0, t)
- Desired or long term availability A: A(∞) (component never fails)
- Availability = $\frac{MTTF}{MTBF} = \frac{MTTF}{MTTF+MTTR}$

Availability example

- Two 9's – 3.65 days/year
- Three 9's – 8.7 hours/year
- Four 9's – 52.56 minutes/year
- Five 9's – 5.25 minutes/year

Terminology

Failure – System(one or more components) as a whole is not working

Error – Part of the component(s) that can lead to a failure

Fault – Cause of an error

Fault Tolerance – System continues to work in the presence of faults

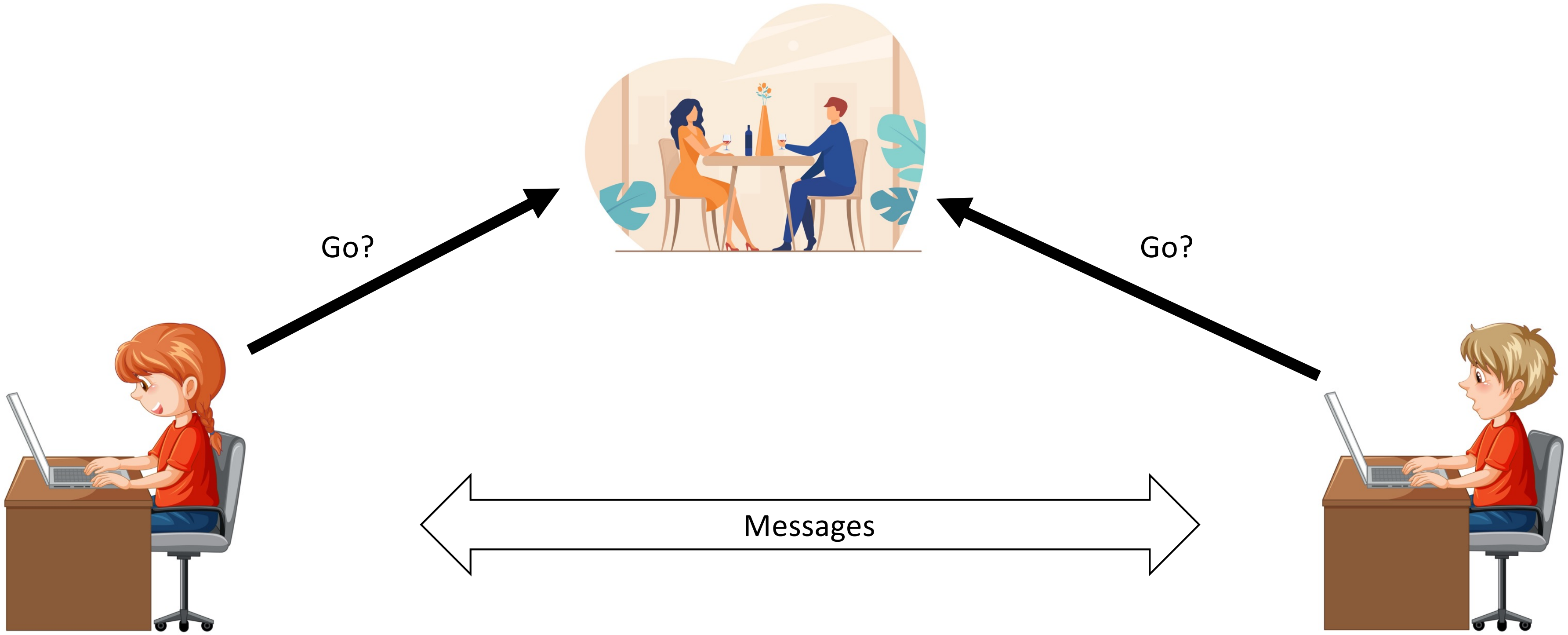
Fault categories

Transient – Occurs once and disappears (Heisenbug)

Intermittent – Occurs many times but in a random fashion

Permanent – Continues to exist until fixed/repaired/replaced (Bohrbug)

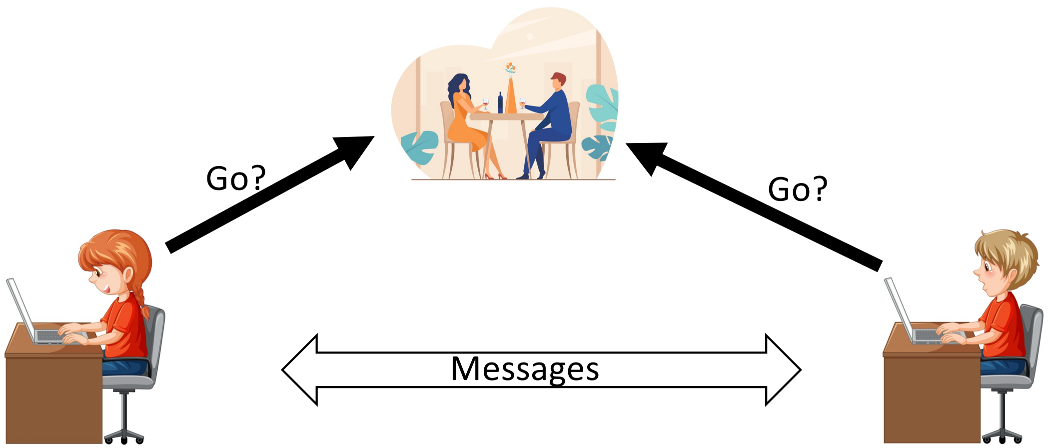
Thought experiment 1: A romantic date...



Designed by Freepik

Thought experiment 1: A romantic date…

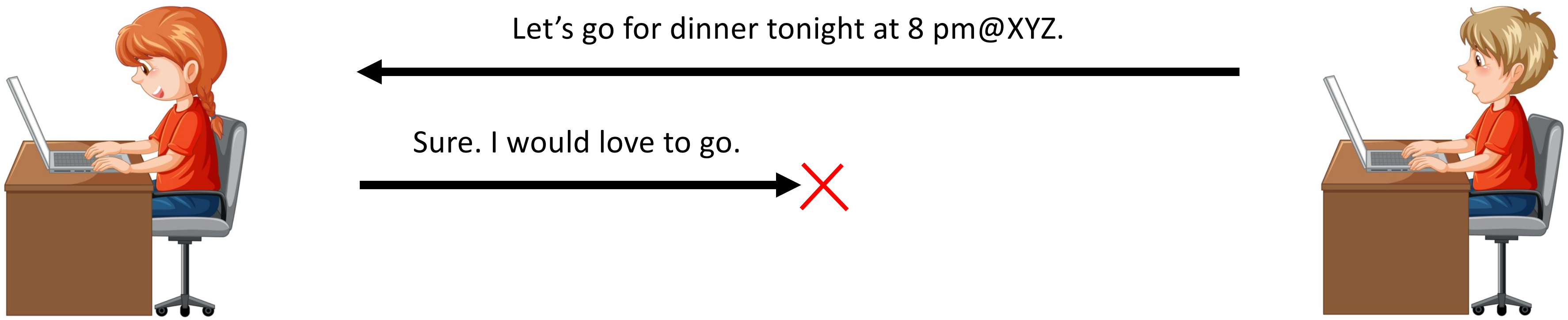
Desired outcome – Boy goes if and only if Girl goes, or vice versa



Boy	Girl	Outcome
Do not go	Do not go	Nothing happens
Goes	Does not go	Poor boy 💔
Does not go	Goes	Poor girl 💔
Goes	Goes	Great romantic date 💞

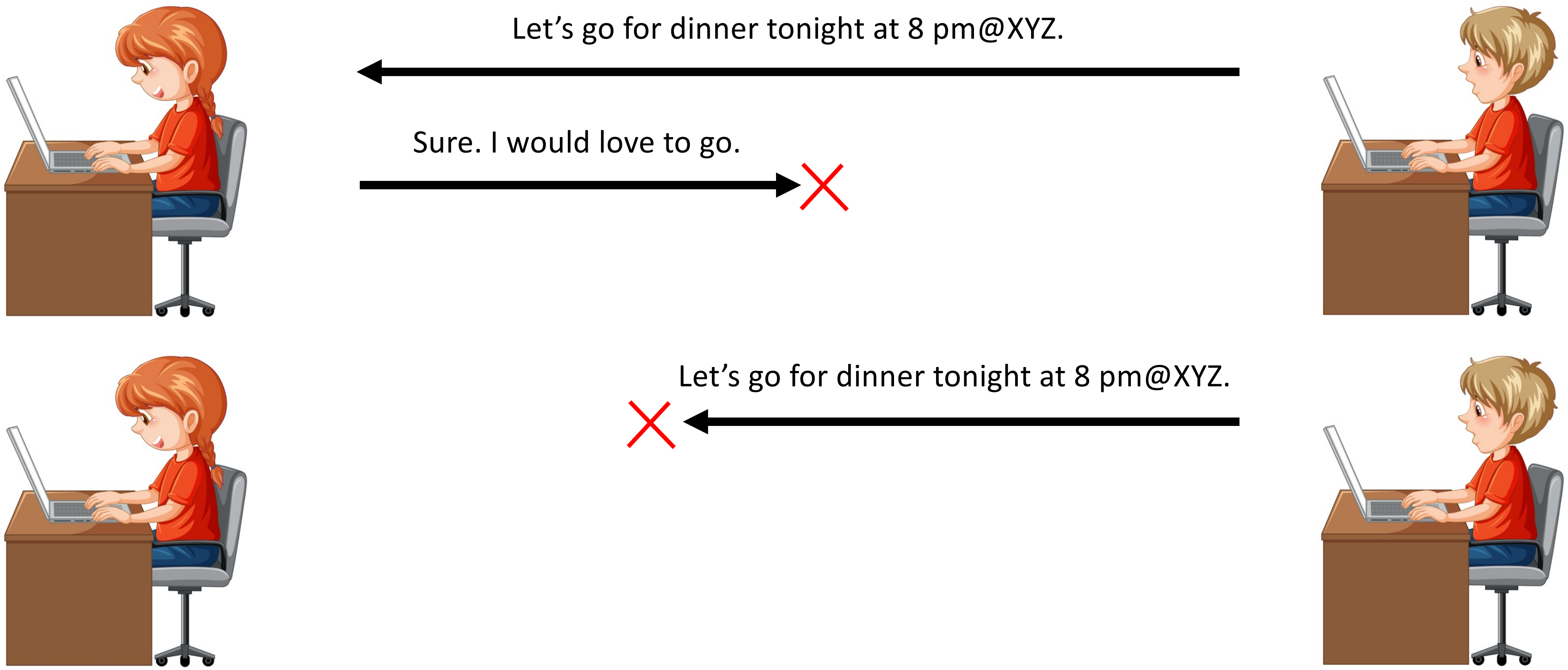
Designed by Freepik

Thought experiment 1: A romantic date...



Designed by Freepik

Thought experiment 1: A romantic date…



Designed by Freepik

Thought experiment 1: A romantic date…

Option 1: From boy's perspective

- Send multiple messages to increase the probability that one will reach
- If none reaches, the boy will go alone assuming the girl would have received atleast one message

Thought experiment 1: A romantic date…

Option 1: From boy's perspective

- Send multiple messages to increase the probability that one will reach
- If none reaches, the boy will go alone assuming the girl would have received atleast one message

Option 2: If boy only goes if it receives a positive response from the girl

- Now the boy is safe
- Girl knows that the boy will only go if her response is received by the boy
- Now, girl's dilemma is the same as the situation as Option 1

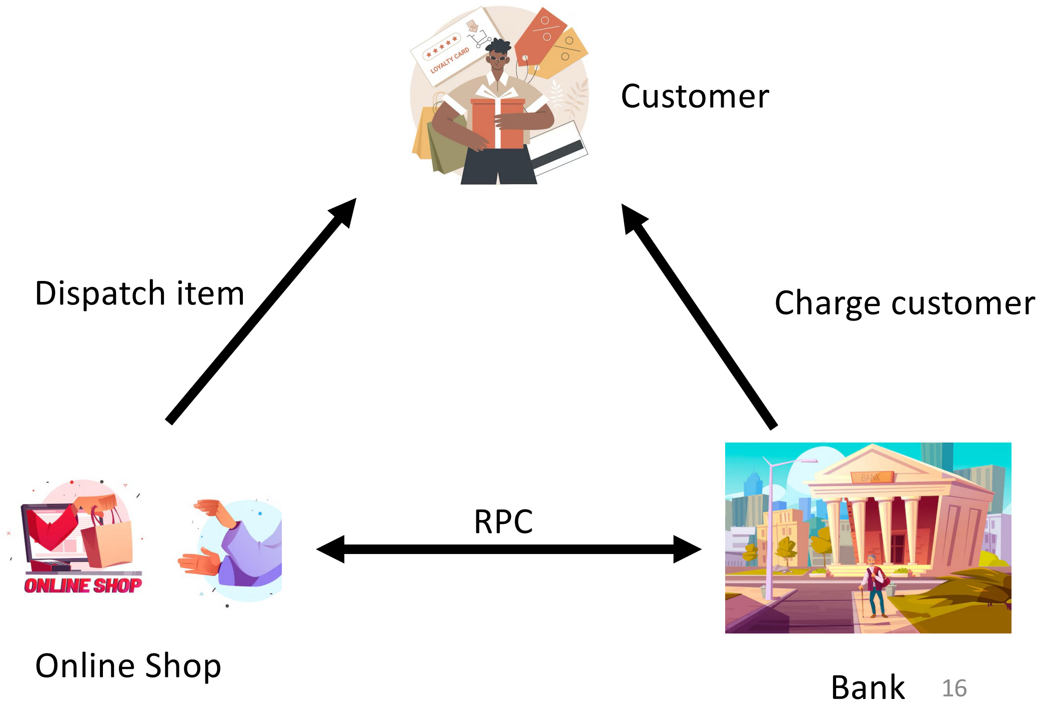
No way for one node to have certainty about second node's state

- The only way of knowing something is to communicate it

Two General's problem

Previous situation is called two general's problem

Real world situation (contrived)

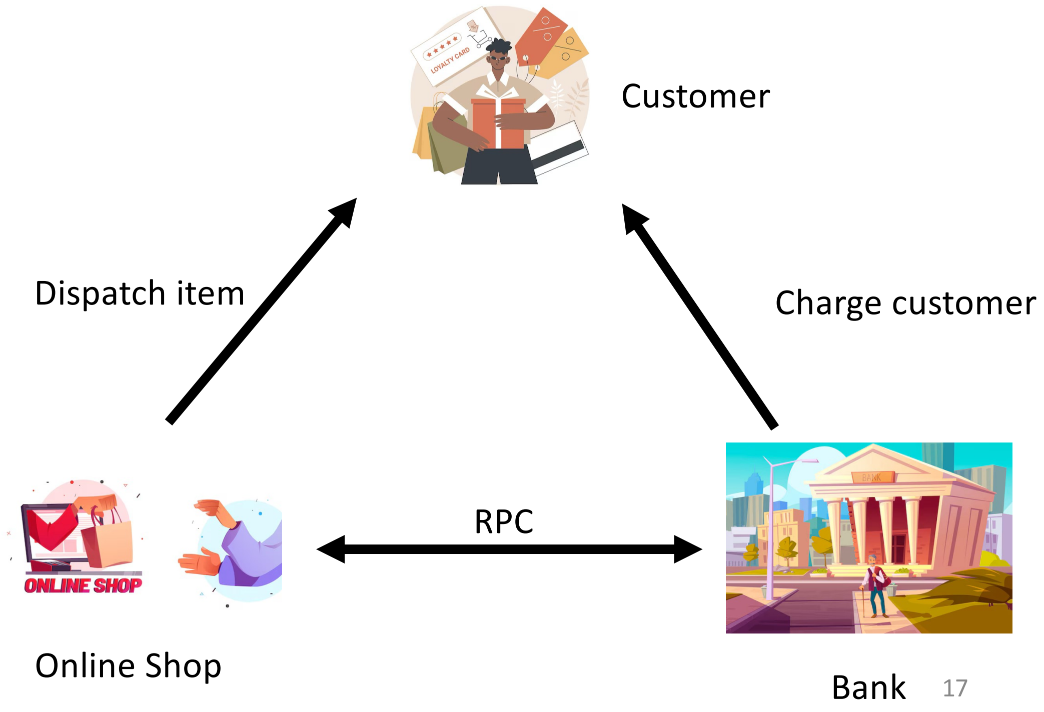


Two General's problem

Previous situation is called two general's problem

Real world situation (contrived)

Bank	Online Shop	Outcome
No charge	No dispatch	Nothing
Charge	No dispatch	Customer complaint
No charge	Dispatch	Shop loses money
Charge	Dispatch	Customer happy



Thought experiment 2:

Thought experiment 2:

Thought experiment 2:

Problem – Byzantine general's problem

- Up to f nodes (friends) might behave maliciously
- Honest nodes (friends) don't know who the malicious ones are
- The malicious nodes (friends) may collude
- Nevertheless, honest nodes (friends) must agree on a plan

Theorem

- Need 3f + 1 nodes (friends) in total to tolerate f malicious nodes (friends)
- Cryptography may help

Key Message

- How do you make sure that multiple entities, which are separated by distance, are in absolute full agreement before an action is taken?

System Models

Two thought experiments

- Two generals problem – a model of networks
- Byzantine generals problem – a model of nodes

Nodes and network both can be faulty

System model captures our assumptions about how nodes and the network behave

- Abstract description of the properties
- Implementation may vary depending on the technology/language used
- Network, Node, Timing behavior

Network Behavior

Network behavior

- Reliable – Message received if it is sent; Messages can be re-ordered
- Fair-loss – Messages may be lost, duplicated, or reordered
- Arbitrary – Malicious adversary may eavesdrop, modify, drop, spoof, replay

Fair-loss → Reliable

Arbitrary → Fair-loss

Node Behavior

Node Behavior – Halting behavior

- Fail-stop – Crash failures, can reliably detect failures
- Fail-noisy – Crash failures; eventually reliably detect failure; noisy behavior
- Fail-silent – Omission or crash failures; clients cannot tell what went wrong
- Fail-safe – Arbitrary; yet benign failures (no harm)
- Fail-arbitrary (Byzantine) – Arbitrary, with malicious failures

Cannot convert from one model to another

Timing assumptions

Synchronous systems

- Process execution speeds and message delivery times are bounded
- Can detect omission and timing failures

Asynchronous systems

- No assumptions about process execution speed or message delivery times
- Cannot reliably detect crash failures

Partially Synchronous systems

- Assume system to be synchronous; no bound-on time when asynchronous
- Can normally detect crash failures

Types of failures

Crash failure – Halts, but was working fine until it halts

Omission failure – fails to respond to incoming requests

- Receive omission – Fails to respond to incoming requests
- Send omission – Fails to send message

Timing failure – Response lies outside a specified time interval

Response failure – Response is incorrect

- Value failure – Value of the response is wrong
- State-transition failure – Deviates from the correct flow of control

Arbitrary failure – May produce arbitrary responses at arbitrary times

Failure detection

Perfect failure detection labels node as faulty if and only if crashed

Typical implementation for crash-stop

- Send message; Await response; Declare node crashed after a certain timeout
- Problem – Node may be unresponsive, crashed and recovered, crashed, lost message, delayed message

Reliable failure detection is practically impossible unless synchronous system and crash-stop behavior

Practical approach

- Use heartbeat to detect crash
- Adaptive timeout;
- Temporary to Eventual

RPC + Fault Tolerance

Five different scenarios

- The client is unable to locate the server
- The server crashes after receiving a request
- The client crashes after sending a request
- The request message from the client to the server is lost
- The reply message from the server to the client is lost

Need to handle failure differently

Client Cannot Locate The Server

Reason

- Servers are down; Mismatch in client-server stub versions

Solution

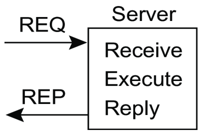
- Use exception to handle such scenarios
- Not all programming language offers exception/signal handling

Server Crashes After Receiving Message

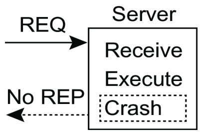
Many reasons to fail – software or hardware failures

Three scenarios

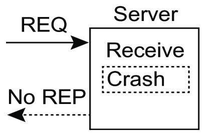
Normal



Crash before sending message



Crash before executing message



Cannot distinguish between the last two scenarios – hard to detect

Two solutions from server's perspective

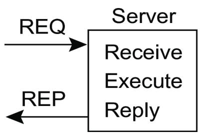
- At least once semantics – Server guarantees it will carry out an operation at least once, no matter what
- At most once semantics – Server guarantees it will carry out an operation at most once

Server Crashes After Receiving Message

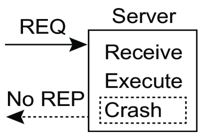
Many reasons to fail – software or hardware failures

Three scenarios

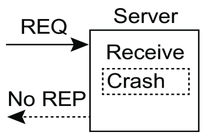
Normal



Crash before sending message



Crash before executing message



Upon server recovery, clients can either

- Always reissue a request – Document may be processed twice
- Never reissue a request – Document may never be processed
- Reissue message if no acknowledgement received from the server
- Reissue message if acknowledgement received from the server

Lost Request Message

Reason

- Network related failure

Solution

- Implement timeout on client stub
- Wait until timeout
- Retransmit message after timeout and no reply/acknowledgement received

If message lost, server would not distinguish between original and retransmission

If server is down, switch to “Cannot locate server”

If request not lost, server could detect retransmission and deal accordingly

Lost Reply Message

Reason

- Network related failure
- Server crashed or is slow/stuck for a while

Solution

- Like lost request message – retransmission
- No way to distinguish lost request, lost reply, server being slow
- Server executing retransmission message may not be possible
  - Idempotent operations – Safe to repeat as often as necessary with no damage done
  - Non-idempotent operations – Unsafe to repeat the execution, can cause correctness issues
- Idempotent operations – Read, Overwrite;
- Non-idempotent operations – Append, Create, Delete
- Detect non-idempotent operations using duplicate filtering

Client Crash

Many reasons to fail – software or hardware failures

Orphans – Server may execute the operation for nothing

Solution

- Terminate the orphan either by client or server
- Expiration
- Termination not good – May lead to other problems (locks not released, incorrect results, etc.)

Checkpoint and Restore – Better option is to not do anything and attempt to restore the client to the state before the crash