# Designing Data Intensive Applications

## CHAPTER - 8

## The Trouble with Distributed Systems

Subscribe for such content
YouTube - MsDeep Singh

### Faults and Partial Failures

↳ Single computer in-general works as per written software.
↳ In distributed system, there is possibility some parts are partial failure working.

### Unreliable Networks

Assuming a shared nothing architecture, things that could go wrong→ (Each machine has its own memory/disk and not accessible to others.)

↳ Request is lost / waiting in queue.
↳ Remote machine failed / temporarily unresponsive
↳ Response took more time than expected.

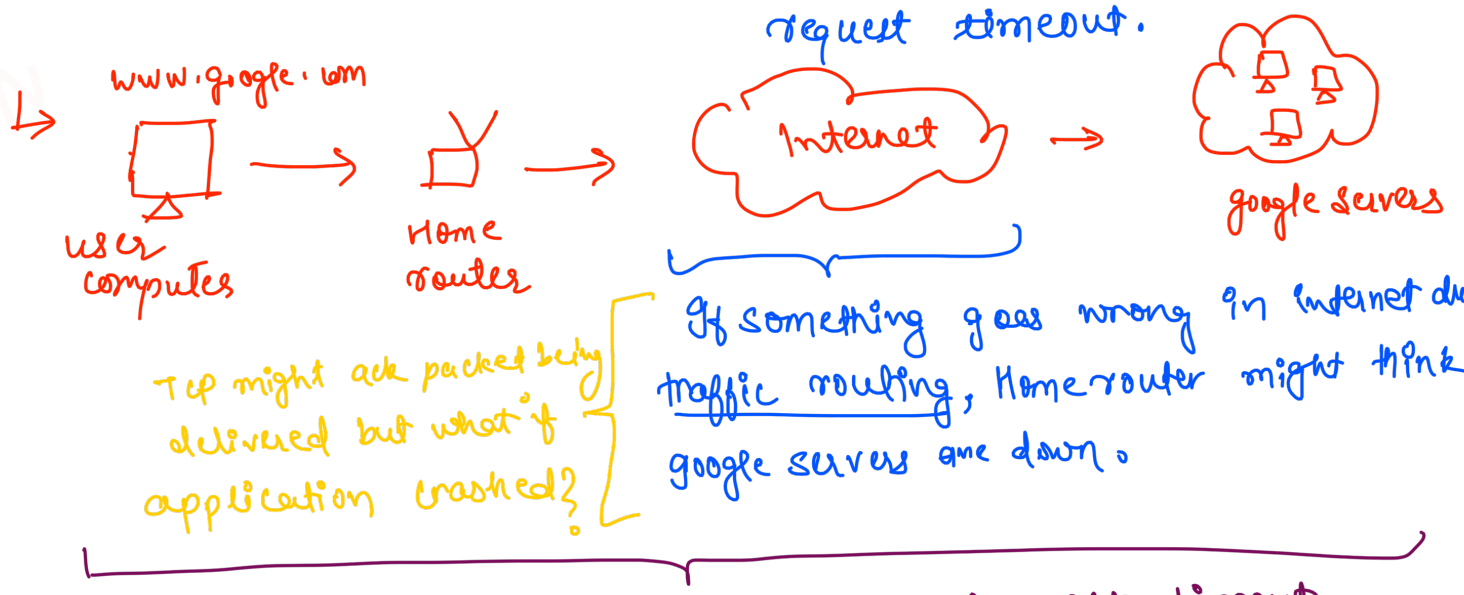
} add timeout so as request don't keep on waiting.

↑ have occurred due to any network faults.

### Detecting Network Faults

Automatic detection
↳ Load Balancer don't send request to faulty node
↳ If leader fails in DB, follower is promoted to leader.

What if we're not able to figure out node is dead or not?
↳ eg. Node OS is running but node process crashed.
another node should takeover before




If something goes wrong on internet during traffic routing, Home router might think google servers are down.

TCP might ack packet being delivered but what if application crashed?

General recommendation is to retry after timeout.
Exponential backoff

## Timeouts and Unbounded Delays

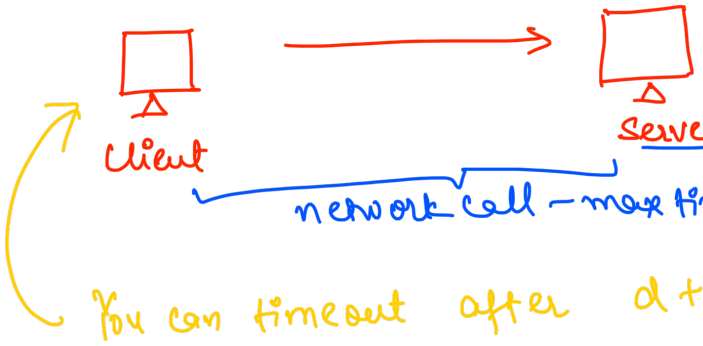
for asynchronous networks, there is no upper limit for packet to be delivered

what should be timeout value?

not too big, not too small.

users have to wait long

node may be actually operational but slow to respond.
eg due to overload




You can timeout after $d + r + d = 2d + r$

## Network Congestion and queueing

requests more than threshold has to wait to be served.

until the requests are served, they are kept in queue

queueing can also be at client end

TCP flow control
backpressure

## TCP vs UDP

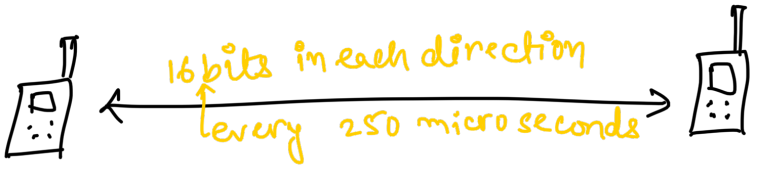
↳ perform flow control
↳ retry – retransmit lost packets

→ dont perform #1 and #2
→ we don't want delayed data
eg. live cricket match.

# Synchronous vs Asynchronous networks

## Fixed line Telephone network

→ fixed guranteed bandwidth is allocated (16 bits)
→ This is Synchronous network.
↳ no queueing - 16 bits reserved
↳ bounded delay → latency is ~fixed.




↳ This concept is not used in TCP.

Should we use packet switching over internet?
no → we don't have exact bandwidth requirements.
you'll end up wasting → depends on use-case.
bandwidth or allocating less bandwidth.

Should not be used for bursty traffic.

## Unreliable Clocks

↳ Clock is used at multiple places in system. example - Calculate latency of operation.
↳ As there are multiple machines in network, time calculation on each machine might vary.
for synchronization of clocks, mostly Network Time Protocol (NTP) is used

### ① Time of Day Clocks

↳ current date and time according to some calendar. (wall-clock time)
Example JAVA - System.currentTimeMillies()
↑return no of second/ms since epoch (January 1, 1970 midnight UTC)

↳ In scenarios of local clock is too far ahead of NTP, it will jump back to previous point in time.
↑ not a good solution for calculation of elapsed time

### ② Monotonic Clocks → guaranteed to always move forward

JAVA — System.nanoTime()
↳ Not dependent on synchronization between different nodes.
↳ NTP will not cause to jump back or forward.
↑ no synchronization is required.

Example on How NTP can go wrong?

> NTP daemon is misconfigured on servers.
↳ firewall is blocking NTP traffic.

Clock offsets between servers should be monitored.

↳ In case clock on one node drift too much from others → node can be declared dead.

↑ will depend on business your servers are spawned for.

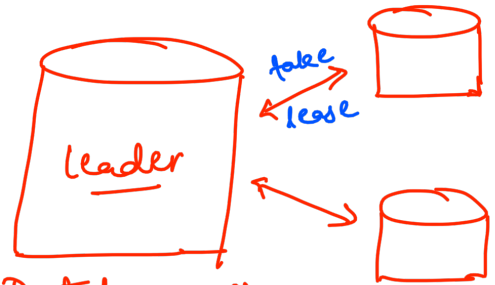
Example - Last write wins algo on DB cluster.

For scenarios where ordering of events is important, Logical Clocks can be used.

why? → based on incrementing counters, it helps in identifying relative order of events.

instead of time elapsed.

## Example → How System Pause can cause issues




Database Cluster
single leader per partition

who is leader → A node in coordination can take lease for certain timeout

↑ needs to be updated once timeout expires

How? via synchronized clocks

If nodes don't update lease, it is assumed dead and another node should takeover.

In this perfect setup, what could go wrong?

① lease time depends on time taken from another machine.

what if clocks are out of sync → can cause issues

② To solve #1 - use monotonic clocks

What if there is unexpected pause in System

ⓐ example due to Garbage Collector (GC)

ⓑ OS access disk for accessing memory pages (virtual memory)

↳ The thread is paused as I/O takes place

If lot of time is spend in doing I/O rather than actual processing → Thrashing.

③ Can we treat GC as

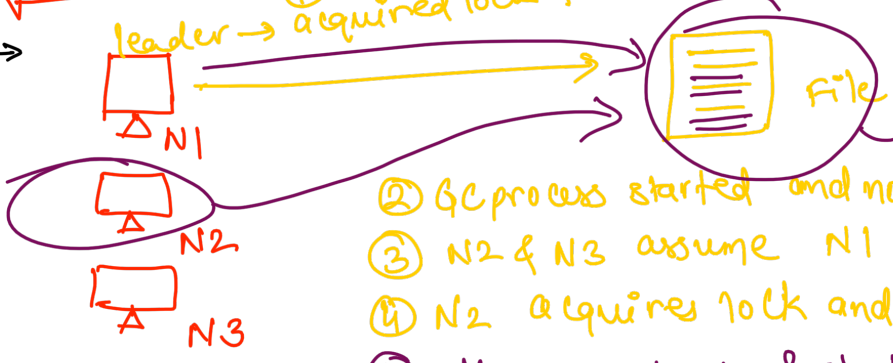
planned outage } → no requests served by node in this time.

② Use GC only for short-lived objects and restart processes periodically.

→ will help in reducing impact on application.

## Fencing Tokens

Problem →




② GC process started and node goes to pause state.

③ N2 & N3 assume N1 is dead

④ N2 acquires lock and start writing to file

⑤ N1 comes back & start writing to file. corrupts the file

→ with every lock, keep a counter which is incremented every time a lock is granted for write.

In above scenario

N1 is granted 10
N1 paused
N2 granted 11

→ as N1 tries to write back with token as 10 it is rejected.

what if N1 update the token at its end and send 11 instead of 10.

Byzantine Faults

Safety and liveness → response is sent by server if certain conditions are met.

↳ example - quorum of nodes.

Even if entire system crash / network fail, algorithm should always return correct result.

See you in next chapter...

Happy Learning ☺