

HARSH WINDS OF REALITY

Distributed Data Management Distributed Systems

Thorsten Papenbrock
Felix Naumann

F-2.03/F-2.04, Campus II
Hasso Plattner Institut

DISTRIBUTED SYSTEMS

Introduction

Distributed Systems

I am facing ...

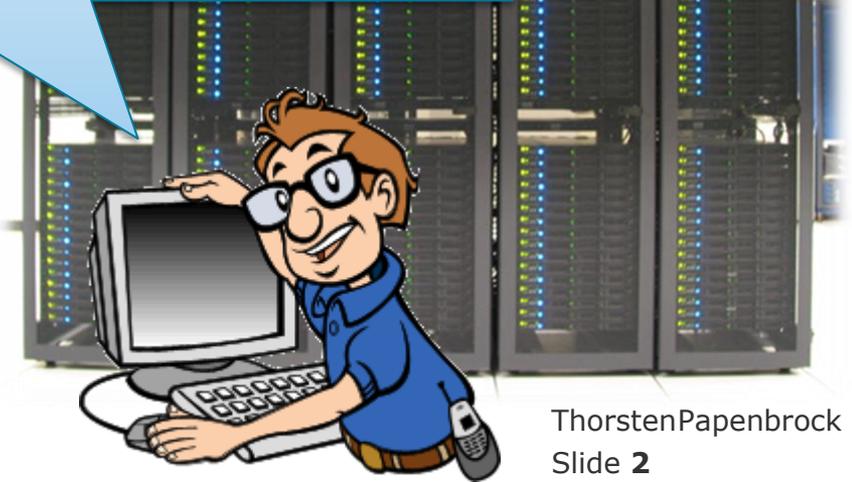
- software bugs
- power failures
- head crashes
- hardware aging
- ...



Non-Distributed System Developer

I am facing everything he faces and ...

- **network** faults
- **clock** deviation
- **partial** (power/network/...) failures
- **nondeterministic** behavior
- ...



Distributed System Developer

Introduction

Distributed Systems

"My system is **predictable**"

"I can debug easily"

"A well operating system **should not have failures**"

"I use parallelism whenever necessary"



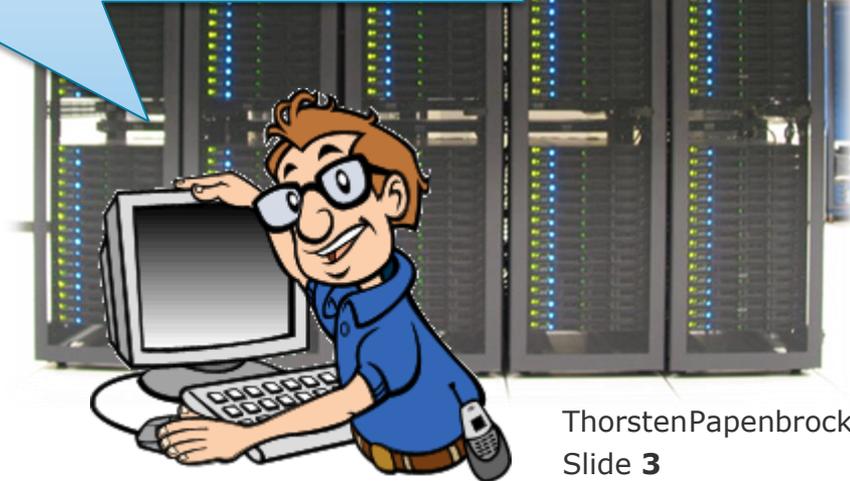
Non-Distributed System Developer

"My system is **predictably unpredictable**"

"Debugging is hard"

"A well operating system **properly deals with its failures**"

"Parallelism is my bread and butter"



Distributed System Developer

Building a reliable system from unreliable components

Given: n number of nodes in the cluster

p likelihood that a node fails (in some arbitrary time interval)

f number of nodes that fail at the same time

Then: Probability of **exactly** f failing nodes can be calculated as (**Binomial distribution**) ...

Number of different ways to pick f nodes in the n node cluster

Failure probability of f independent nodes with p failure likelihood

Well-being probability for the n-f other nodes

$$P(f) = \binom{n}{f} * p^f * (1 - p)^{n-f}$$

$$P(1) = \binom{n}{1} * p^1 * (1 - p)^{n-1}$$

$$= n * p * (1 - p)^{n-1}$$

Distributed Data Management

Distributed Systems

Thorsten Papenbrock
Slide 5

Reliability despite Unreliable Components

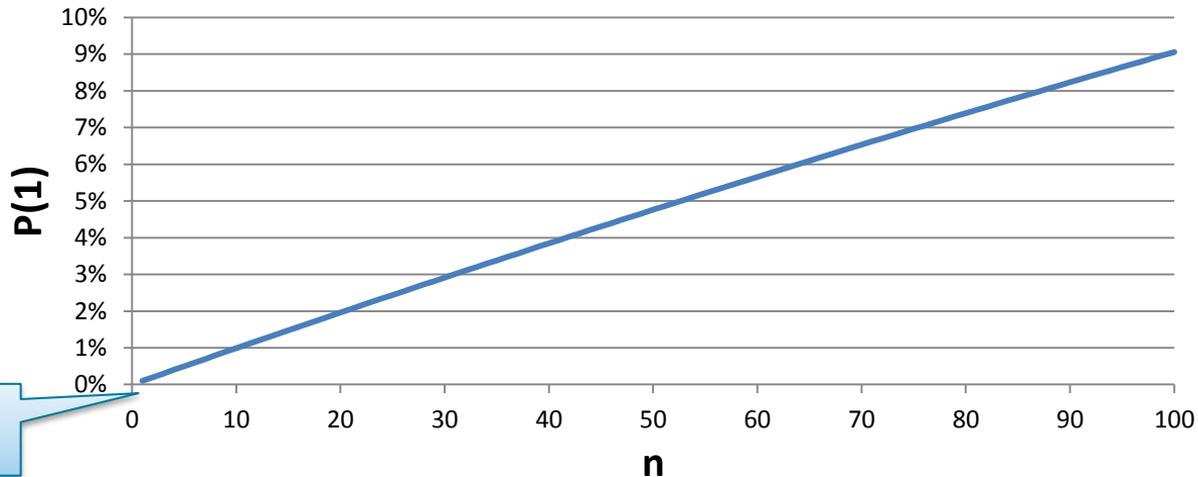
Building a reliable system from unreliable components

Given: n number of nodes in the cluster

p likelihood that a node fails (in some arbitrary time interval)

f number of nodes that fail at the same time

Then:



$p = 0.1\%$

Distributed Data Management

Distributed Systems

ThorstenPapenbrock
Slide 6

Building a reliable system from unreliable components

- With no special fault handling:
 - A distributed system is only as reliable as its weakest/strongest component
- With fault handling:
 - A distributed system is (much) more reliable as its unreliable components

Fault handling examples

- Radio inference on wireless networks:
 - Error-correcting codes allow digital data to be transmitted accurately
- Unreliable Internet Protocol (IP):
 - Transmission Control Protocol (TCP) retransmits missing packages, eliminates duplicates, and reassembles packets in order

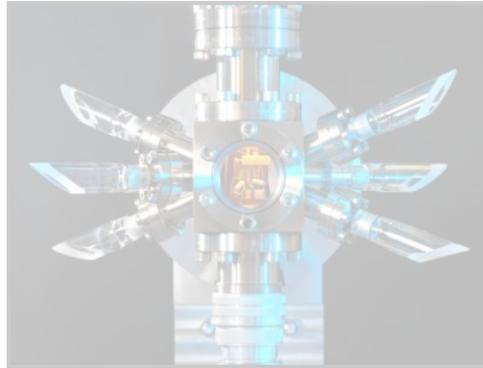
Some easily solvable faults

Unreliable Networks



A shark raiding an undersea cable

Unreliable Clocks



An atomic clock with minimum drift

Knowledge, Truth, Lies



Students communicating their knowledge

Distributed Data Management

Distributed Systems

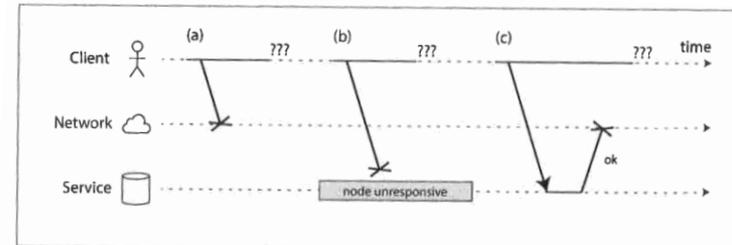
Network

- Physical connection between autonomous, shared-nothing computing nodes
- **Asynchronous** messaging via **packet** binary sequences
- Nodes can send messages but no guarantees as to when/whether it arrives

Sender can't even tell if the packet was delivered ...

Potential failures when sending a message

- Request is **lost** on the network (e.g. cable unplugged)
- Request is **waiting** in a queue and delivered later (e.g. recipient overloaded)
- Remote node is **unavailable** (e.g. recipient crashed or updates)
- Response is **delayed** on the network (e.g. network overloaded)
- Response is **lost** on the network (e.g. network switch misconfigured)



Unreliable Networks

Detecting Faults

Using the operating system

- If a process on a node crashes, but the operating system (OS) still runs:
 - OS can close or refuse TCP connections to notify clients with an error
 - OS can trigger failover scripts to explicitly notify certain clients

Using the network switch

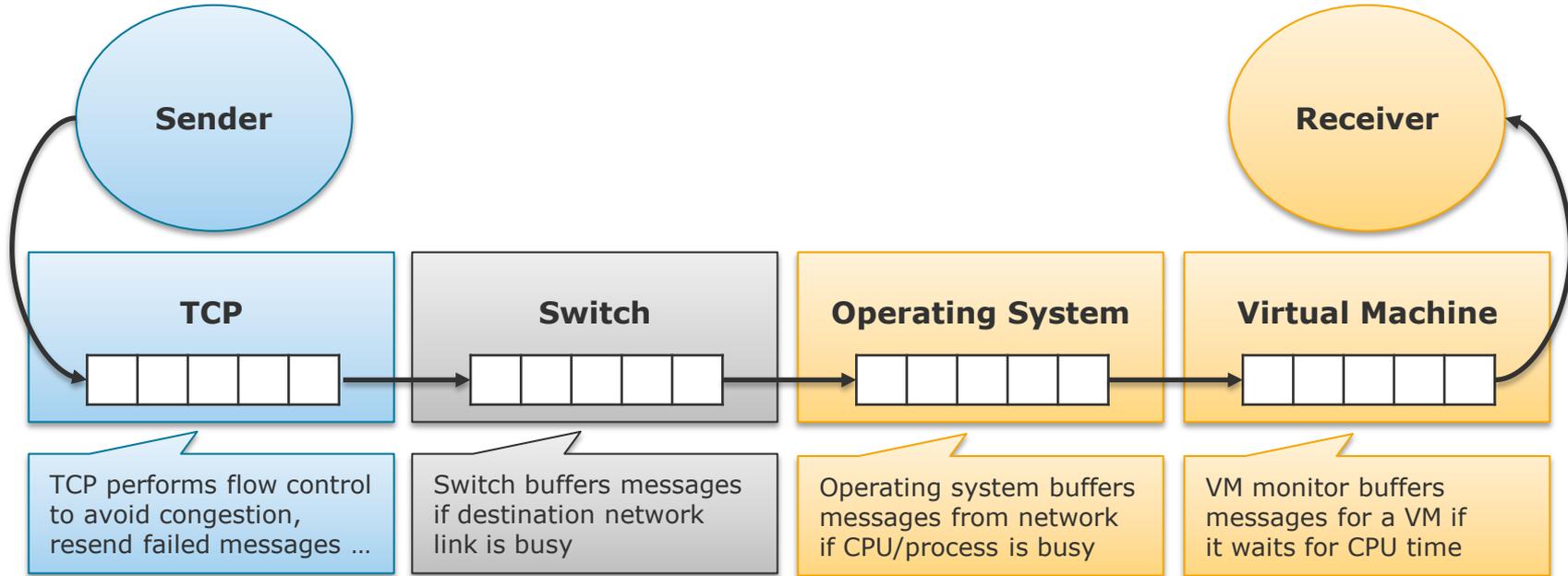
- If the client has access to the network switch:
 - Switch can detect link failures on hardware level (e.g. detect if remote is powered on)

Using timeouts

- Log the sending time for each message
- Messages are declared lost if their recipient does not answer within a certain timeout
 - Most universal fault detection mechanism

Unreliable Networks

Queues on the Network



- Many reasons for a packages being delayed (**query congestion**)
- Even if the receiver could guarantee a **processing time** for messages, the network cannot guarantee a **transmission time** for messages

Issues

- How to set the timeout?
 - Too long (**conservative**): program waits wastefully long before triggering fault handling
 - Too short (**aggressive**): more false message loss reports each triggering fault handling
- How to handle failures?
 - Resend message } → Messages might get **handled multiple times!**
 - Reroute message } → Messages might **worsen overload** if this caused the timeout!
 - Escalate as system error

Analytical Systems

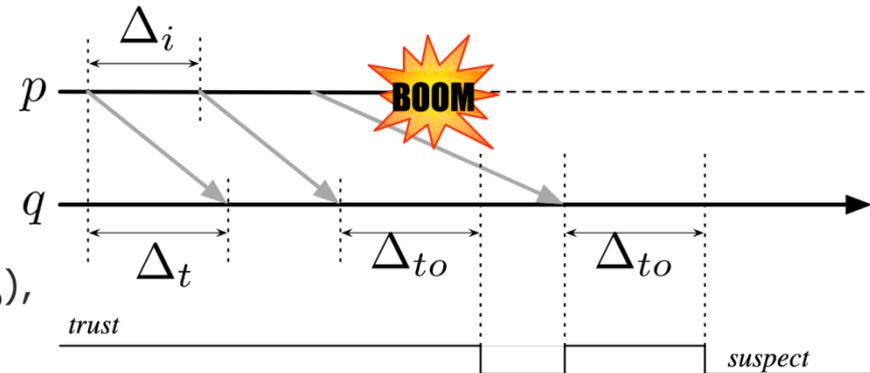
- Nodes with **high CPU load** due to analytical calculations
- Network with **high traffic** due to data-intensive nature
 - Overall high system load makes timeouts hard to predict

Note that we cannot know:

- What caused the error?
- Has a message been worked on?

The traditional heartbeat method

- The **monitored process p** sends periodical heartbeat messages to the **server process q**
- Δ_i : the heartbeat send interval of p
- Δ_t : the initial wait time
- Δ_{to} : the timeout
- Upon receiving the first heartbeat (Δ_t), p measures the time to the next heartbeat (Δ_{to}), which is then set as the timeout
- Problems:
 - **Static timeout**: query congestion might **naturally delay** heartbeats on higher load
 - **Initialization**: if the second heartbeat is delayed, Δ_{to} is **set too large**
 - **Binary trust**: client is either **trusted** or **suspected**



The accrual failure detector method

- **Accrual failure detector:**
 - German: “anwachsende Fehler Erkenner”
 - Output a suspicion level for each node instead of binary trust or fixed timeout
 - **Suspicion level:**
 - Measure describing the probability that node p has failed at time t
 - Defined as a continuous function for p over t : $susp_level_p(t) \geq 0$
 - Properties
 - **Asymptotic completeness:** If p is faulty, $susp_level_p(t) \rightarrow \infty$
 - **Eventual monotony:** If p is faulty, $susp_level_p(t)$ monotonically increases
 - **Upper bound:** If p is correct, $susp_level_p(t)$ has an upper bound
 - **Reset:** If p is correct, $susp_level_p(t) = 0$ for some $t > t_0$
- Used to adjust **load balancing** and **timeout expectations**

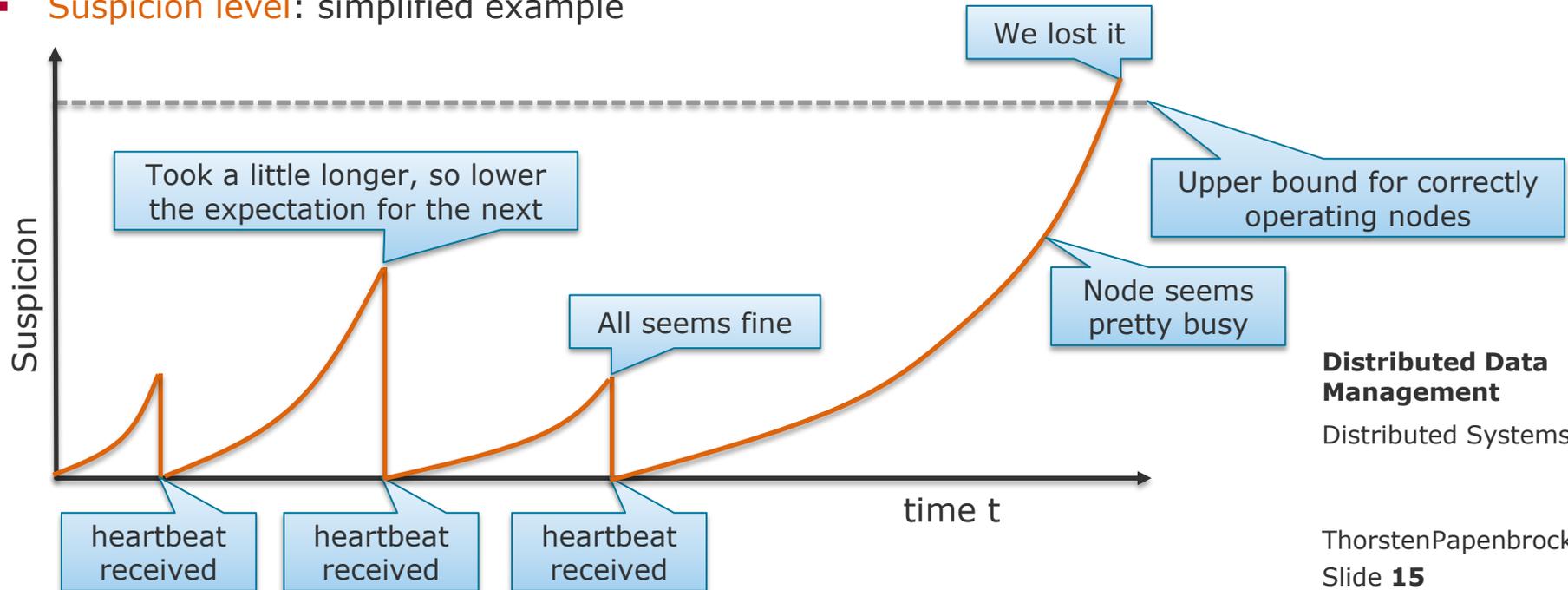
Trust is interpreted from the **development of suspicion**

i.e., whenever a heartbeat arrives

Defining Timeouts Experimentally

The accrual failure detector method

- **Suspicion level:** simplified example



The accrual failure detector method

▪ Suspicion level interpretation:

▪ Example interpretation algorithm:

- Initialize two dynamic thresholds T_{high} and T_{low} to the same arbitrary values >0 and start trusting a node
- **S-transition:**
 - Whenever $\text{susp_level}_p(t)$ crosses T_{high} upwards, $T_{\text{high}} = T_{\text{high}} + 1$ and suspect p
- **T-transition:**
 - Whenever $\text{susp_level}_p(t)$ crosses T_{low} downwards, $T_{\text{low}} = T_{\text{high}}$ and trust p

➤ The longer the algorithm monitors $\text{susp_level}_p(t)$, the better T_{high} captures real node failures

▪ Suspicion dynamically **adjusts to the current latency and load**

➤ T_{high} becomes a fix threshold that is robust against load changes

Unreliable Networks

Defining Timeouts

Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama,
"The φ Accrual Failure Detector",
Japan Advanced Institute of Science and Technology, School of Information Science,
Technical Report IS-RR-2004-010, May 2004

The φ accrual failure detector

- A concrete implementation of the accrual failure detection method
- Implemented in Akka, Spark, Flink, Cassandra, Riak, ZooKeeper, ...
- φ (Phi):
 - Suspicion level: $\varphi_p(t) = \text{susp_level}_p(t)$
 - Comparable: if $\varphi_p(t) > \varphi_q(t)$, p is more likely to fail at time t than q , i.e.,
 p differs more clearly from its usual timing than q
 - Useful for fault detection and load balancing
- General idea:
 - Continually measure response times (jitter) and availability of nodes via heartbeat
 - Calculate $\varphi_p(t)$ based on p 's heartbeat history

Unreliable Networks

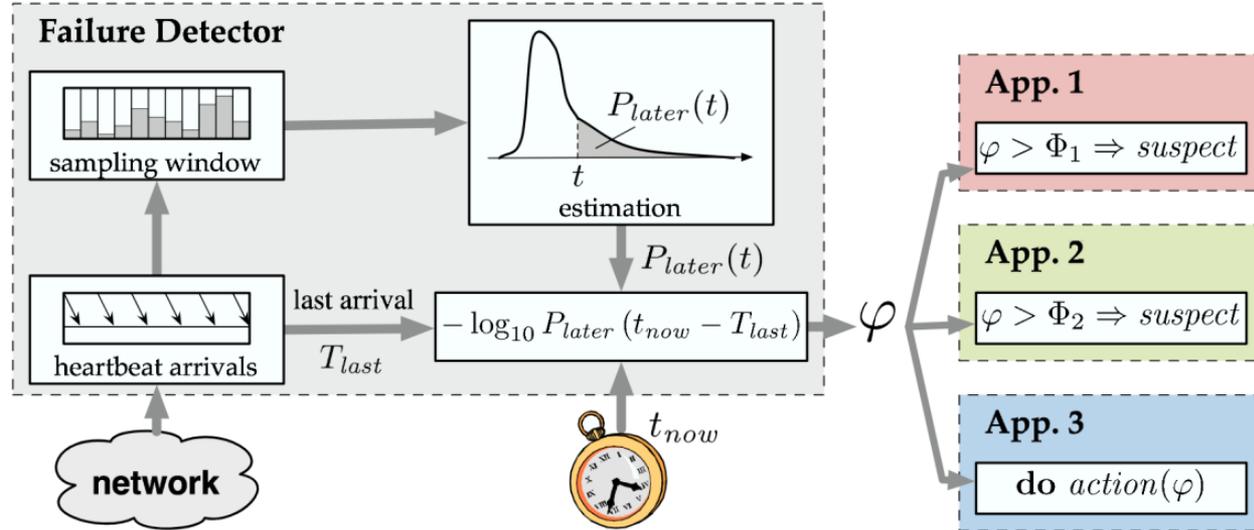
Defining Timeouts

Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama,
"The φ Accrual Failure Detector",
Japan Advanced Institute of Science and Technology, School of Information Science,
Technical Report IS-RR-2004-010, May 2004

The φ accrual failure detector

Variables

- T_{last} : Arrival time of most recent heartbeat
- t_{now} : Current time
- P_{later} : Probability that a heartbeat will arrive more than t time units after the previous one



Heartbeat arrivals

- Heartbeats arrive with a sequence number to restore their send order

Sampling window

- Stores the arrival times in a fixed sized window (last x heartbeats per node)
- Pre-calculates the **arrival intervals**, **sum**, and **sum of squares** of all samples

Unreliable Networks

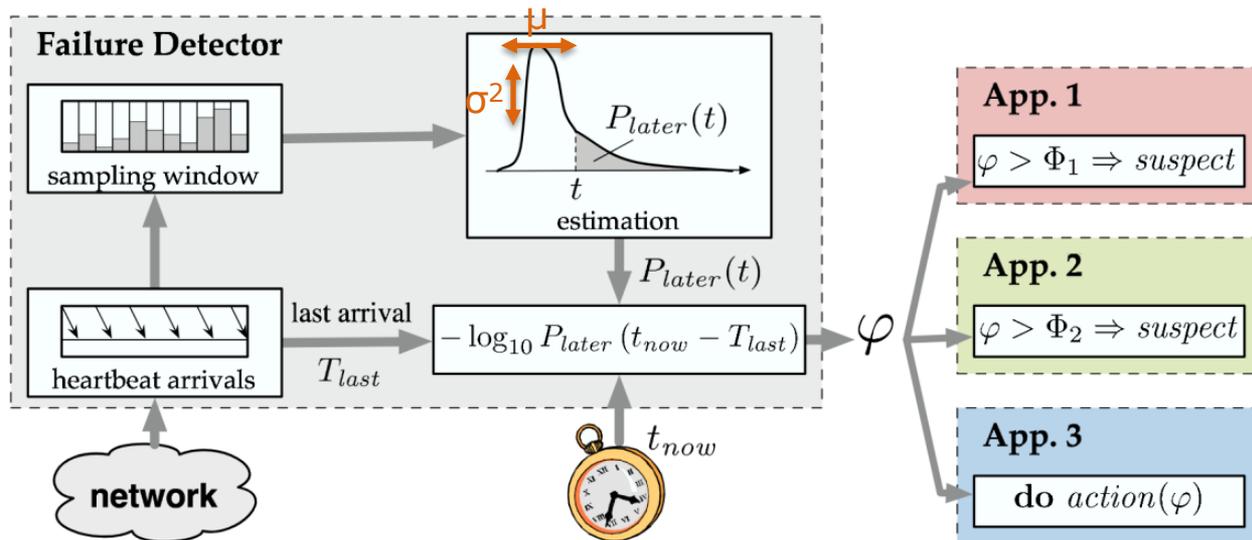
Defining Timeouts

Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama,
"The φ Accrual Failure Detector",
Japan Advanced Institute of Science and Technology, School of Information Science,
Technical Report IS-RR-2004-010, May 2004

The φ accrual failure detector

Variables

- T_{last} : Arrival time of most recent heartbeat
- t_{now} : Current time
- P_{later} : Probability that a heartbeat will arrive more than t time units after the previous one



Estimation

1. Calculate the **mean** μ and the **variance** σ^2 for the samples
2. Calculate $P_{later}(t)$:

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_t^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

Unreliable Networks

Defining Timeouts

Naohiro Hayashibara, Xavier Défago, Rami Yared, and Takuya Katayama,
 "The φ Accrual Failure Detector",
 Japan Advanced Institute of Science and Technology, School of Information Science,
 Technical Report IS-RR-2004-010, May 2004

The φ accrual failure detector

Variables

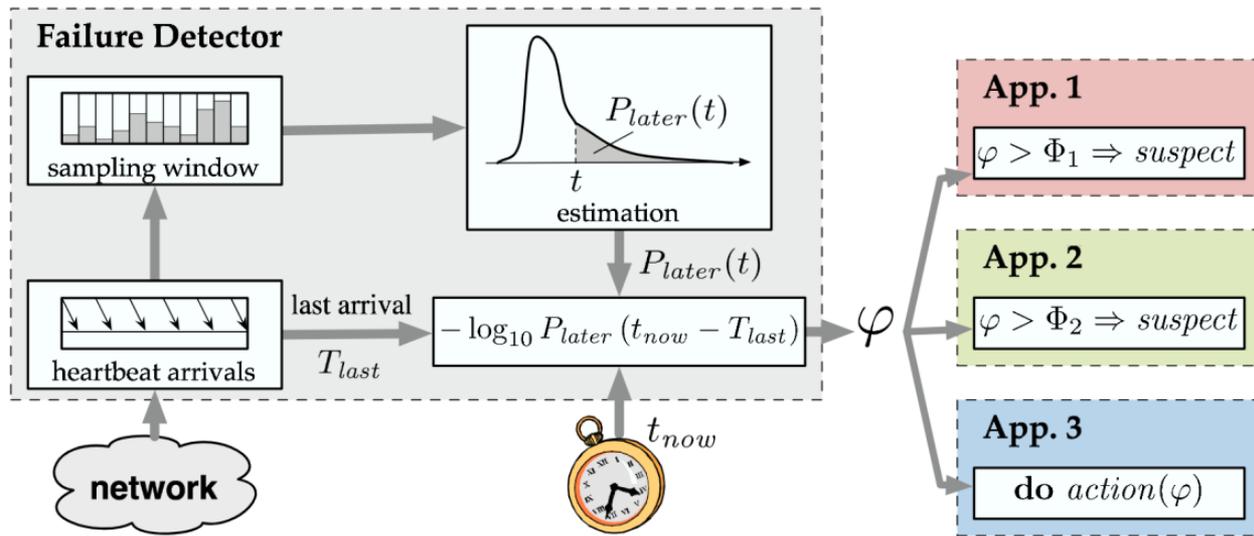
- T_{last} : Arrival time of most recent heartbeat
- t_{now} : Current time
- P_{later} : Probability that a heartbeat will arrive more than t time units after the previous one

φ calculation

3. Calculate φ using P_{later} and the time since p 's last heartbeat:

$$\varphi(t_{now}) \stackrel{\text{def}}{=} -\log_{10}(P_{later}(t_{now} - T_{last}))$$

P_{later} gets increasingly smaller; $-\log_{10}$ turns small in very large values



Interpretation by application:
 E.g. failure detection with T_{high} and T_{low} where $T_{high} = \Phi$

Unreliable Networks

Ignoring Timeouts

TCP vs. UDP

- User Datagram Protocol (UDP) does not use timeouts
 - No guarantee of delivery, ordering, or de-duplication
 - **Preferable** if outdated messages are worthless:



video streaming



gaming



sensor processing



VoIP calls

ThorstenPapenbrock
Slide 21

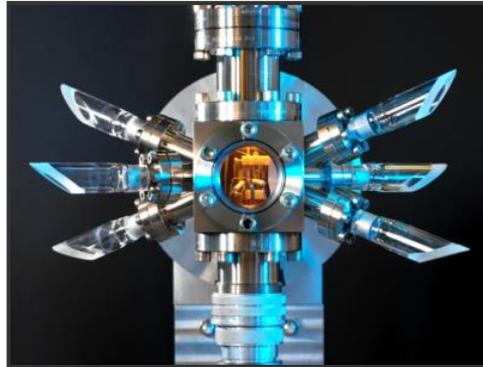
- **Problematic** for most analytical use cases!

Unreliable Networks



A shark raiding an undersea cable

Unreliable Clocks



An atomic clock with minimum drift

Knowledge, Truth, Lies



Students communicating their knowledge

Distributed Data Management

Distributed Systems

Unreliable Clocks

Clocks vs. Networks

Unreliable clocks

Often cause
silent, creeping failures and data loss



Unreliable networks

Usually cause
noticeable crashes and failures



Both need to be considered in application logic!

Distributed Data Management

Distributed Systems

Thorsten Papenbrock
Slide 23

Unreliable Clocks

About Clocks

Computer clocks

- Actual hardware devices: **quarz crystal oscillator**
- Not perfectly accurate and not in sync with other clocks

Clock usage in distributed systems

1. Measure **duration** e.g.:
 - Has this request timed out yet?
 - What's the 99th percentile response time of this service?
 - How long did the user spend on this page?
2. Measure **points in time** e.g.:
 - When was this heartbeat send?
 - When does this cache entry expire?
 - What's the timestamp of this error message?



**Distributed Data
Management**

Distributed Systems

ThorstenPapenbrock
Slide **24**

Unreliable Clocks

About Clocks

Kinds of clocks

a) Time-of-day clock:

- Returns the current time according to some calendar (e.g. millis since 01.01.1970 UTC)
- Example: `clock_gettime(CLOCK_REALTIME)` (Linux)
`System.currentTimeMillis()` (Java)
- Can be changed completely (e.g., synchronized via NTP)
- Used to measure points in time



b) Monotonic clock:

- A constantly forward moving clock with no reference point (specific values are meaningless)
- Example: `clock_gettime(CLOCK_MONOTONIC)` (Linux)
`System.nanoTime()` (Java)
- Can be speeded up or slowed down (e.g., by 0.05% via NTP)
- Used to measure durations (time intervals)



Unreliable Clocks

Unreliability

Clock drift

- Natural deviation of clock speeds due to ...
 - machine temperature
 - gravitation
 - aging and abrasion
- Unavoidable even if clocks get synchronized frequently



Illusion of synchronized clocks

- **Clock drift**: 17 sec drift for clocks synchronized once a day (Google)
- **Back-shifts**: clocks being forced to sync to past times
- **Network delay**: no synchronization can work around network delay
- **Leap seconds**: necessary time adjustment due to earth rotation
- **Virtualization**: VMs use virtualized clocks that pause if VM has no CPU time

Use Libraries for Time-Calculations!



Tom Scott

The Problem with Time & Timezones - Computerphile

1,383,370 views

39K 374 SHARE

<https://www.youtube.com/watch?v=-5wpm-gesOY>

Distributed Data Management

Distributed Systems

ThorstenPapenbrock
Slide 27

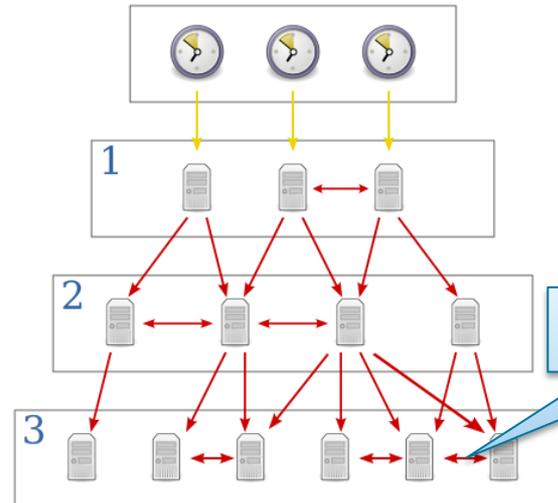
Unreliable Clocks Synchronization

David
L. Mills



Network Time Protocol (NTP)

- Most popular **clock synchronization protocol** for packet-switched, variable-latency data networks
- Assumption:
 - Some nodes (server) have very precise clocks (atomic, GPS, ...)
- Protocol:
 - Nodes with less precise clocks synchronize their clocks with these reference clocks directly or indirectly
 - The closer a node is to the reference clocks, the more precise it can (potentially) sync its clock



Internet protocol suite

Application layer

BGP • DHCP • DNS • FTP • HTTP • IMAP •
LDAP • MGCP • NNTP • **NTP** • POP •
ONC/RPC • RTP • RTSP • RIP • SIP • SMTP
• SNMP • SSH • Telnet • TLS/SSL • XMPP •
more...

Transport layer

TCP • UDP • DCCP • SCTP • RSVP •
more...

Internet layer

IP (IPv4 • IPv6) • ICMP • ICMPv6 • ECN •
IGMP • IPsec • *more...*

Link layer

ARP • NDP • OSPF • Tunnels (L2TP) • PPP
• MAC (Ethernet • DSL • ISDN • FDDI) •
more...

V • T • E

sanity checking

Unreliable Clocks

Synchronization

Network Time Protocol (NTP)

- Synchronization Algorithm:
 - Client nodes regularly poll server nodes and calculate:

1. time offset

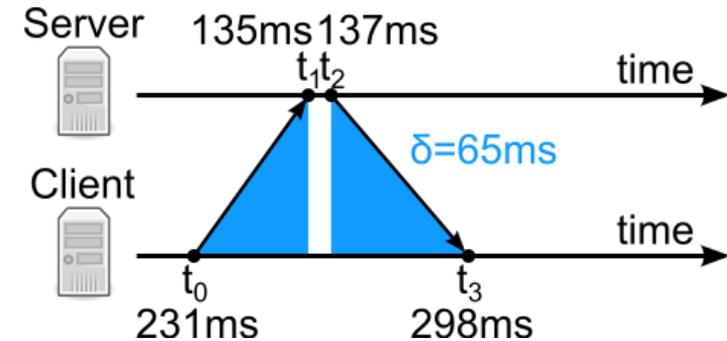
t_1 and t_3 include transmission time so it is added and deleted

$$\theta = \frac{\text{Offset send} + \text{Offset receive}}{(t_1 - t_0) + (t_2 - t_3)}$$

2. round-trip delay

2
Because we calculated the offset twice!

$$\delta = (t_3 - t_0) - (t_2 - t_1)$$



t_0 , t_1 , t_2 , and t_3 are timestamps attaches to the sync message

- θ and δ are passed through statistical analysis removing outliers
- Client then **gradually** adjusts its local clock using θ

Network Time Protocol (NTP)

- Most popular **clock synchronization protocol** for packet-switched, variable-latency data networks
- Computers synchronize their time with a group of servers
- Servers get their time from more accurate time sources

Confidence in local time t

- Estimation about the deviation between local and system time
 - A client's local time t can be expected to be $t + \text{uncertainty}$
 - **Uncertainty** \approx **own expected clock drift since last NTP-sync** + **network round-trip time** + **server's uncertainty**
- Systems that rely on synchronized clocks try to estimate uncertainty and incorporate it in their application logic

Internet protocol suite

Application layer

BGP • DHCP • DNS • FTP • HTTP • IMAP • LDAP • MGCP • NNTP • **NTP** • POP • ONC/RPC • RTP • RTSP • RIP • SIP • SMTP • SNMP • SSH • Telnet • TLS/SSL • XMPP • *more...*

Transport layer

TCP • UDP • DCCP • SCTP • RSVP • *more...*

Internet layer

IP (IPv4 • IPv6) • ICMP • ICMPv6 • ECN • IGMP • IPsec • *more...*

Link layer

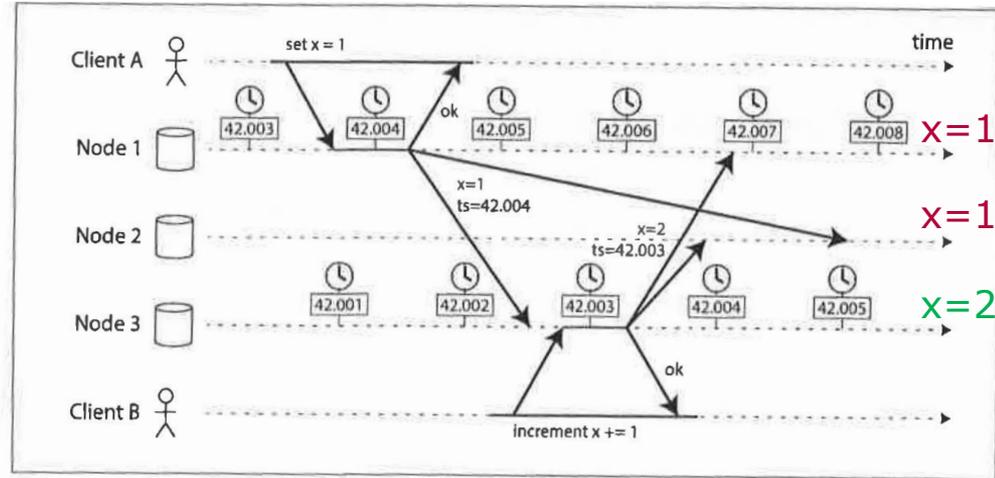
ARP • NDP • OSPF • Tunnels (L2TP) • PPP • MAC (Ethernet • DSL • ISDN • FDDI) • *more...*

V • T • E

Unreliable Clocks Risks

Synchronized clocks in distributed DBMSs

- Used often when messages require a global ordering
- **Last-Write-Wins (LWW)**:
 - Writes get a timestamp from the first node that sees them
 - During change propagation, newer writes overwrite older writes
 - If clocks are out-of-sync, newer writes might get **overwritten/dropped**
- **Snapshot isolation**:
 - Transactions get a timestamp from the node that opens them
 - During transaction processing, transactions only see older changes
 - If clocks are out-of-sync, snapshots might be **inconsistent**



**Distributed Data
Management**

Distributed Systems

ThorstenPapenbrock
Slide 31

Unreliable Clocks

Risks

Synchronized clocks in distributed DBMSs

- Used often when messages require a global ordering
- **Single-leader lease:**
 - In single-leader replication, the leader obtains a lease with a timestamp
 - **Lease:**
 - Kind of a lock with timeout that can be hold by only one node
 - If leases timeout expires, the leader needs to renew the lease
 - If leader fails and does not renew, another leader can be elected
 - If clocks are out-of-sync, leader might hold lease for too long (two leader **brain split**)
 - If the leader **pauses and resumes** in a critical section, it might **process writes without permission**

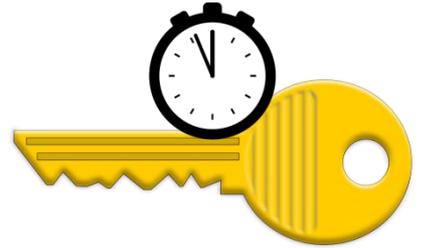
```
while (true) {  
    request = getIncomingRequest();  
  
    if (lease.expiryTimeMillis -  
        System.currentTimeMillis < 10000) {  
        lease = lease.renew();  
    }  
  
    if (lease.isValid()) {  
        process(request);  
    }  
}
```

Better not pause here!

Remember:
no mutexes,
semaphores, ...
in distributed
systems!

Leases

- Necessary if a system requires that there is only one of some thing:
 - One node with a **certain permission** for a particular resource
 - One node with a **particular role** in the system (e.g. leader)
- Obtaining a **lease** grants exclusive rights for a certain time
- Assumption:
 - One node (lock service/server/authority) assigns locks/leases
- **Fencing token**:
 - A number that increases every time a lock is assigned
 - Handed to the lease owner as part of the lease
 - Lease owner must issue the fencing token with every action
 - **Locked resource (!)** checks if fence token is up-to-date (e.g. newest)
 - Reject if other node possesses newer fence token

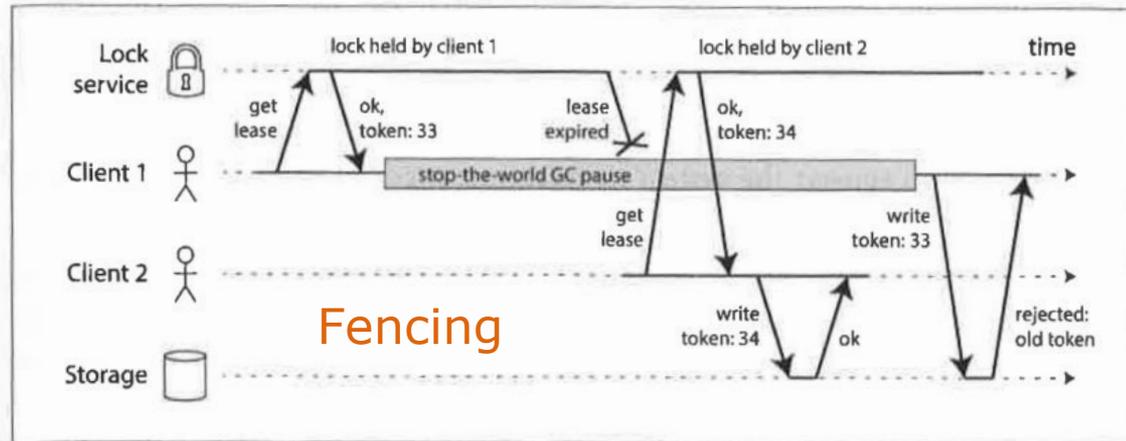
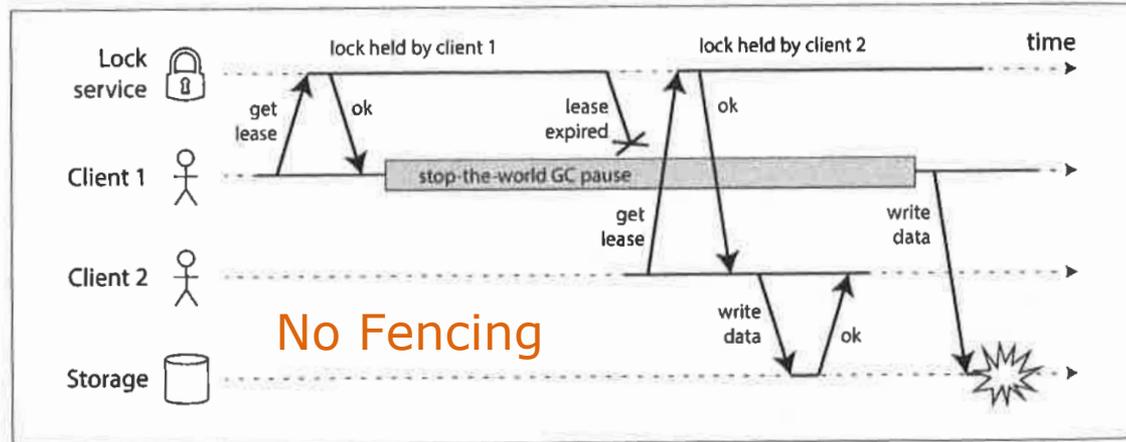


To counter the problem:
A node **wrongly thinks**
that it has the lock!

Unreliable Clocks Locking

Leases

- Example:



Overview

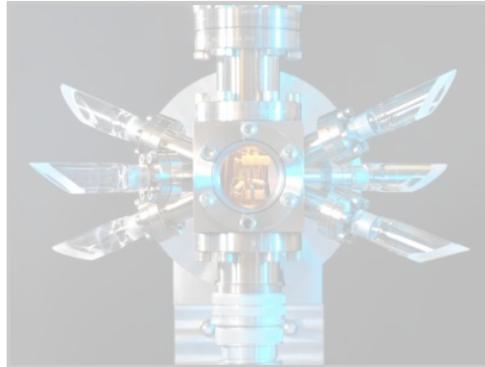
Distributed Systems

Unreliable Networks



A shark raiding an undersea cable

Unreliable Clocks



An atomic clock with minimum drift

Knowledge, Truth, Lies



Students communicating their knowledge

Distributed Data Management

Distributed Systems

Knowledge

- A node can know nothing about other nodes for sure
 - Can only make guesses based on received messages

Truth

- Truth can be defined by majority or authority
- Examples for majority:
 - A node loses its connection to the network, but is still alive
 - The majority sees the node disappear and will declare it dead (although the connection and not the node was faulty)
 - A change propagation message gets lost on the network
 - The majority holds an outdated value that is declared valid (although the most recent value is on the node issuing the change)



Knowledge, Truth, and Lies

Determining Facts

Remember quorum reads and writes
(quorum consistency)
from chapter replication!

Quorum

- Minimum number of nodes that must agree on a statement to be accepted
- Typically (but not necessarily) the majority ($>50\%$ of the nodes)
 - A majority quorum is safe, because there can be only one majority



Knowledge, Truth, and Lies

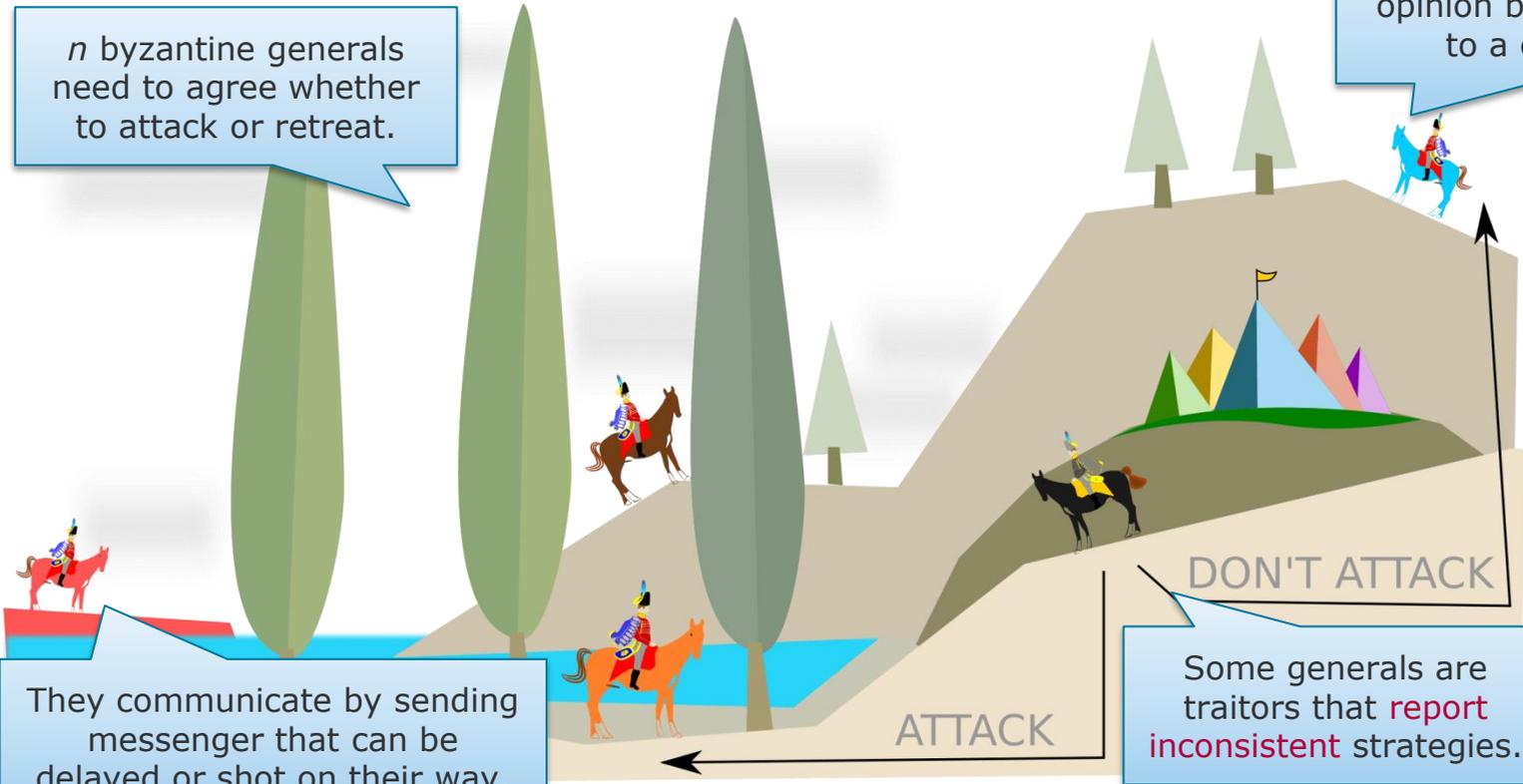
Lies: Byzantine Problem

n byzantine generals need to agree whether to attack or retreat.

Every general has an own opinion but would agree to a consensus.

They communicate by sending messenger that can be delayed or shot on their way.

Some generals are traitors that report inconsistent strategies.

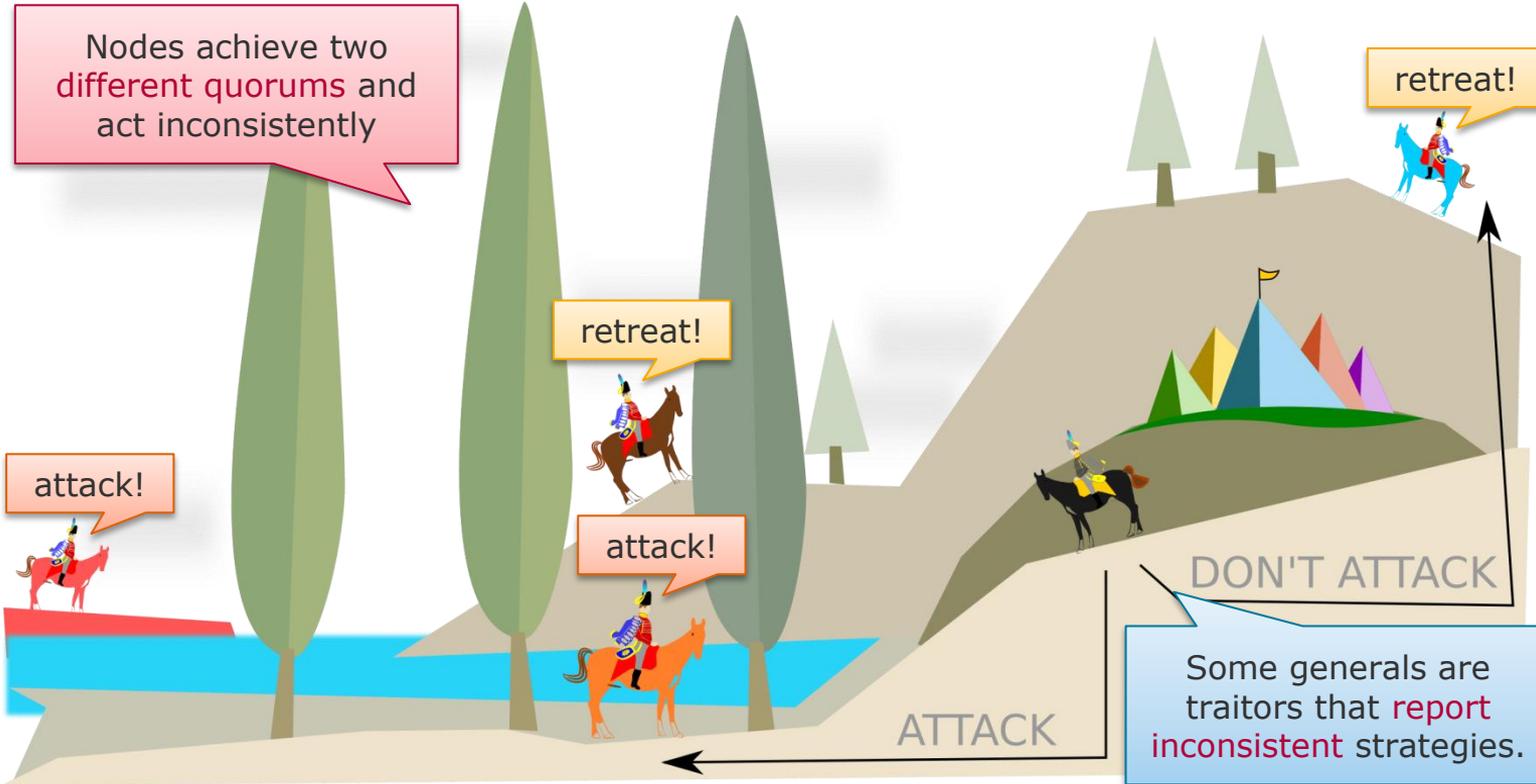


Distributed Data Management
Distributed Systems

ThorstenPapenbrock
Slide 38

Knowledge, Truth, and Lies

Lies: Byzantine Fault



Weak Lies

- Nodes accidentally send invalid information (with no bad intention):
 - outdated, miss-calculated, damaged, lost, ...
- Reasons:
 - software bugs, signal interference, misconfiguration, hardware faults, software update ...
- Protection:
 - **checksums** (e.g. TCP), **redundancy** (e.g. NTP), **quorums** (e.g. Cassandra), **sanity checks** (application), ...

Byzantine Lies

- Nodes systematically send invalid information (usually with bad intention)
- Reasons:
 - hardware faults, security compromises, malicious attacks, ...
- Protection:
 - **complicated, often inefficient consensus protocols**
 - hardware-based, multiple-consensus-rounds, consensus-hierarchies, proof of work ...

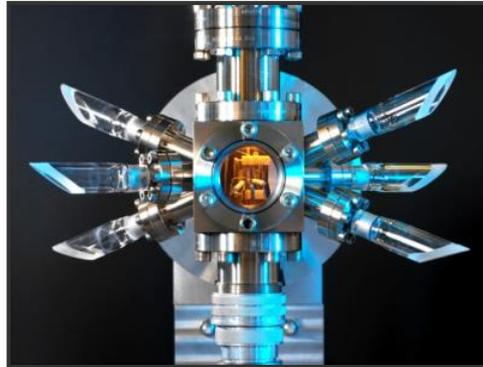
Distributed Systems Summary

Unreliable Networks



A shark raiding an undersea cable

Unreliable Clocks



An atomic clock with minimum drift

Knowledge, Truth, Lies



Students communicating their knowledge

Unreliable Networks

- Messages can be lost, reordered, duplicated, and arbitrarily delayed

Unreliable Clocks

- Time is approximate at best, unsynchronized, and can pause

Distributed Data Management

Distributed Systems

Thorsten Papenbrock
Slide 41

The φ accrual failure detector

- Suppose we observed the following heartbeat intervals (in s):
 - 14, 34, 15, 11, 17, 10, 35, 29, 28, 21
- Furthermore, assume we haven't received a heartbeat for 31s now.
- Use the φ accrual failure detector to estimate the probability P_{later} that the heartbeat will still arrive and give the value of φ .
- In reality, the heartbeat intervals follow a Gaussian distribution with the parameters mean $\mu=15.0$ and variance $\sigma^2=100.0$.
By what factor did we misjudge the probability of P_{later} ?



HARSH WINDS OF REALITY

CLOCK ERROR

PROCESS PAUSES

UNBOUNDED NETWORK DELAY

DISTRIBUTED SYSTEMS