Characterization of Distributed Systems

Marco Aiello - RuG - a.y. 2014/15

(based on http://www.cdk4.net)

What is a distributed system?

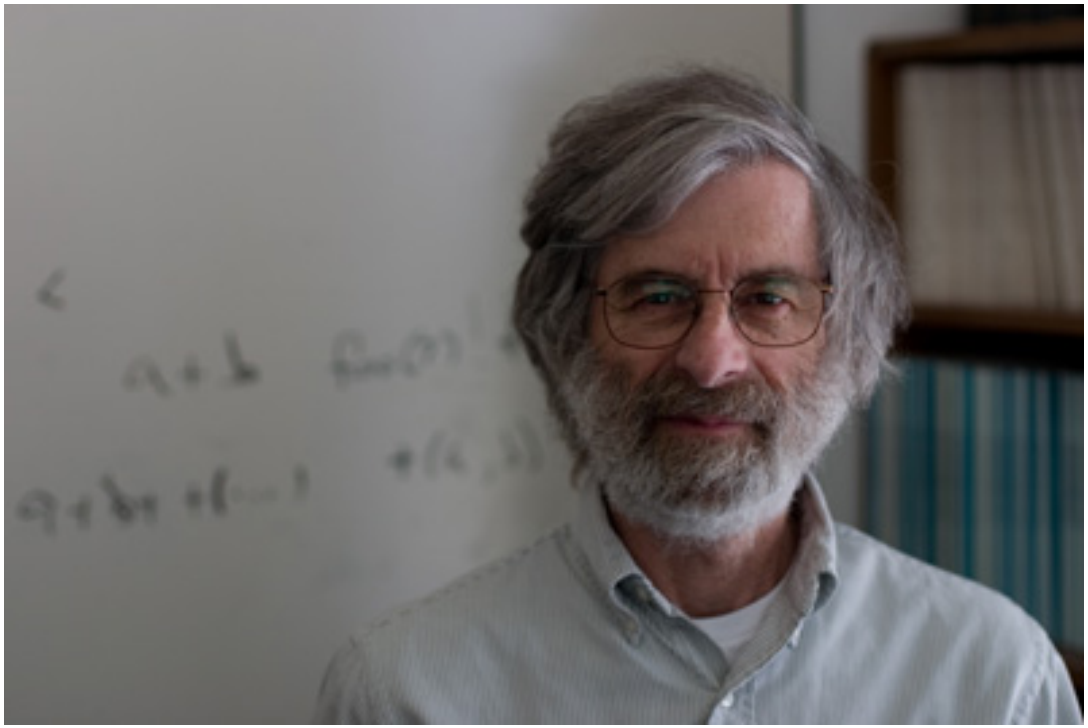
A distributed system is a collection of autonomous hosts that are connected through a computer network. Each host executes computations and operates a distribution middleware, which enables the components to coordinate their activities via message-passing in such a way that users perceive the system as a single, integrated computing facility.

university of groningen

Distributed what?

"A distributed system is one in which the failure of a machine you have never heard of can cause your own machine to become unusable"

Leslie Lamport



Facts from the definition

1. Hosts
2. Network
3. Distributed computations
4. Middleware
5. Message-passing
6. Perceive single and integrated

- Concurrency of components
- Lack of global clock
- Independent failures of components

Challanges

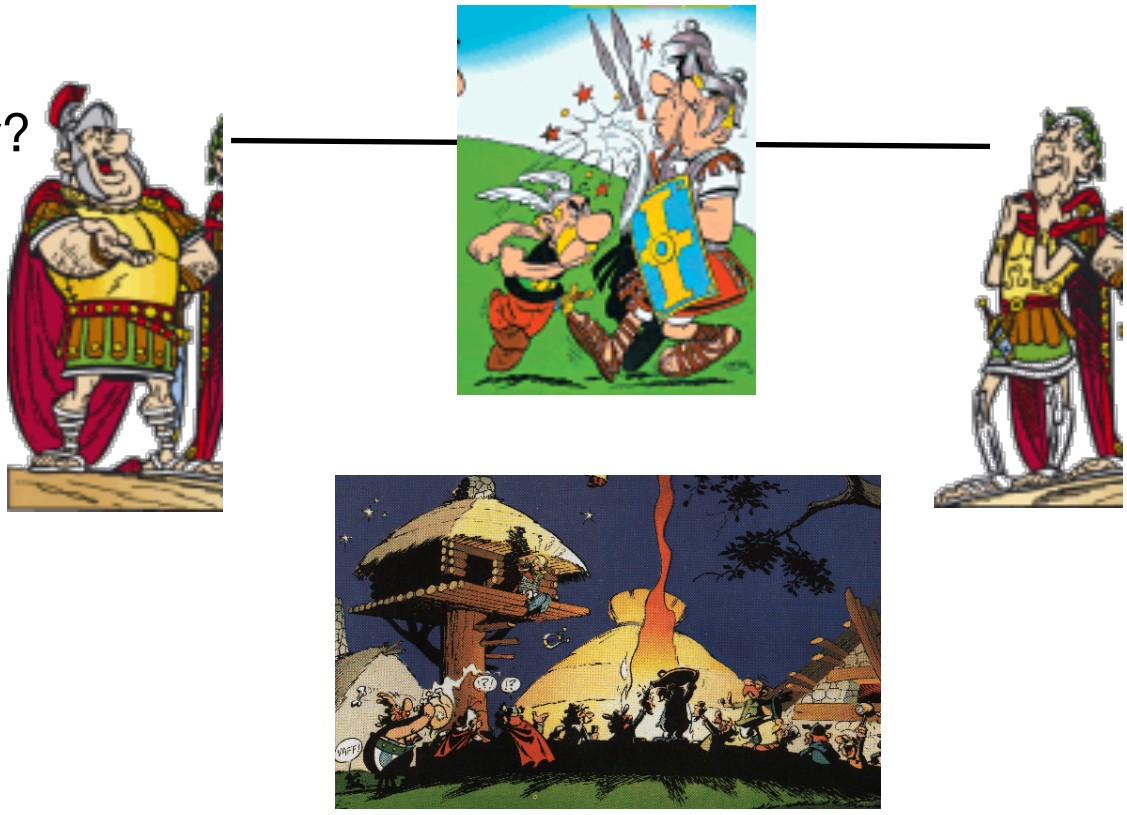
1. Heterogeneity
2. Openness
3. Security
4. Scalability
5. Failure handling
6. Concurrency
7. Transparency

Two generals problem

- Two Generals need to coordinate an attack against an enemy. If they attack individually, they will loose, if they attack together they will win.

- But the enemy lies in the middle and can intercept the coordination messages and avoid delivery

- Can the generals defeat the enemy?



Two generals

- **Theorem:** there is no non-trivial protocol that guarantees that the generals will always attack simultaneously

- **Proof:** Ab absurdum, suppose there is one such protocol that does the job in the minimum number of steps $n>0$.

  Consider the last message sent, the n-th. The state of the sender cannot depend on its receipt, the state of the receiver cannot depend on its arrival, so they both do not need the n-th message. So we would have a protocol with n-1 messages. But that contradict the hypothesis

- **Fact:** A solution requires reliable message delivery.

Challenges

1. Heterogeneity
2. Openness
3. Security
4. Scalability
5. Failure handling
6. Concurrency
7. Transparency

university of groningen

Heterogeneity

- **Levels**:
  - Network
  - Computing hardware
  - Operating systems
  - Programming languages
  - Implementations by different developers

- **Middleware**: software layer which abstracts from the above providing a uniform computational model

university of groningen

Openness

- The degree to which a computer system can be extended and re-implemented
  - Publishing interfaces
  - Standardization

Security

Security components:

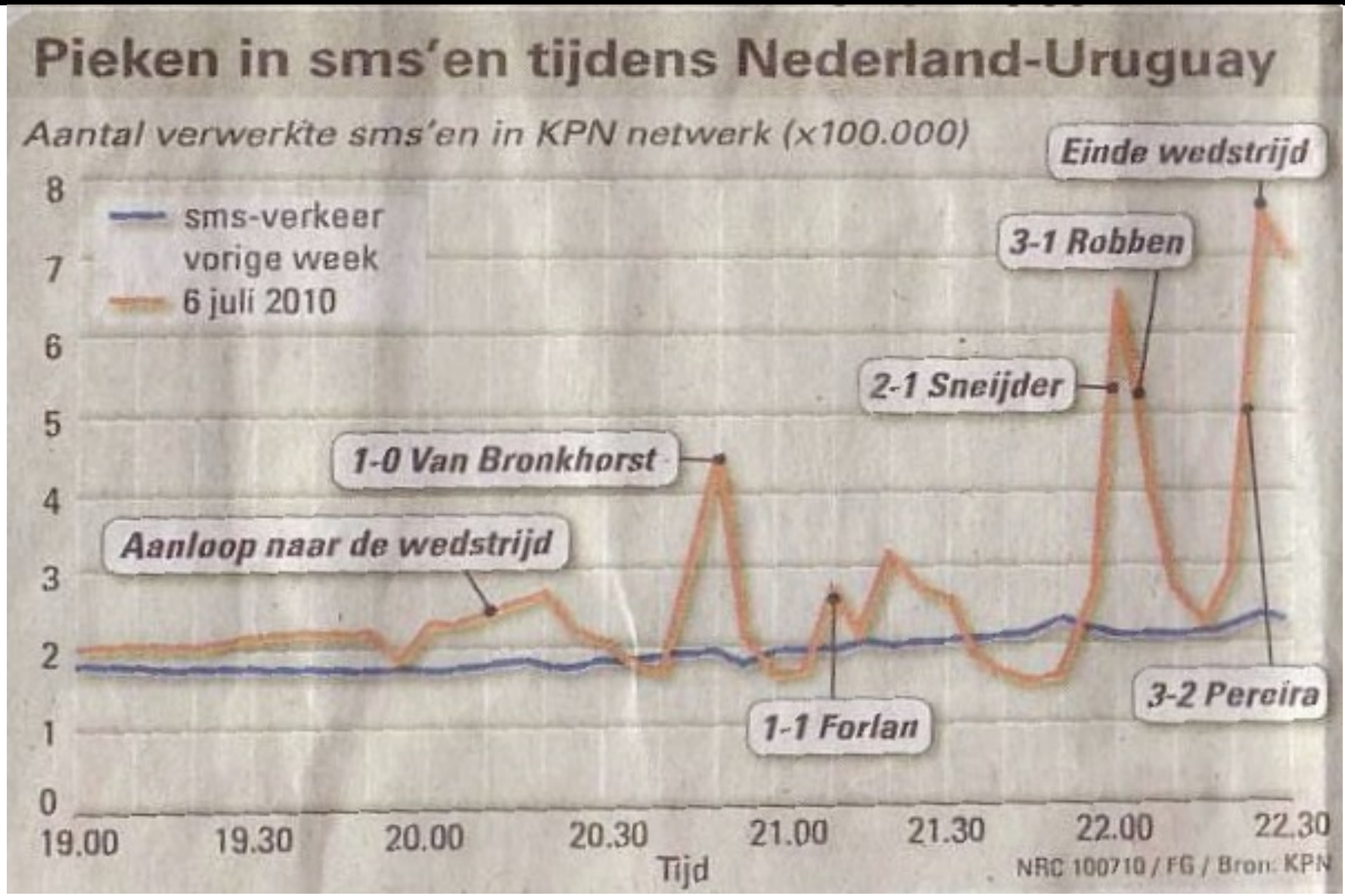
- **Confidentiality**: disclosure of the contents of a message to a party different from the intended receiver, e.g., packet sniffing
- **Integrity**: corruption of the transimitted contents by a third party , e.g., man in the middle attack
- **Availability**: intereference with a communication, e.g., denial of service attack

university of groningen

Scalability

- A distributed system is scalable if operates effectively and efficiently independently from the number of resources and users connected to it
- Challenges:
  - Control the costs of physical resources
  - Control performance loss
  - Control available software resources
  - Control performance bottlenecks

Why scalability is important



13

university of groningen

Failure handling

- Failure detection (e.g., message checksum)
- Failure masking (e.g., email retransmission)
- Fault tolerance (e.g., array of servers)

- Detected failures may be recovered and masked
- Redundancy may improve fault tolerance

university of groningen

Concurrency

- Is at the heart of any distributed system in which there are more than two parties acting simultaneously

Transparency

- *Access transparency*: enables local and remote resources to be accessed using identical operations.
- *Location transparency*: enables resources to be accessed without knowledge of their location.
- *Concurrency transparency*: enables several processes to operate concurrently using shared resources without interference between them.
- *Replication transparency*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- *Mobility transparency*: allows the movement of resources and clients within a system without affecting the operation of users or programs.
- *Performance transparency:* allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.