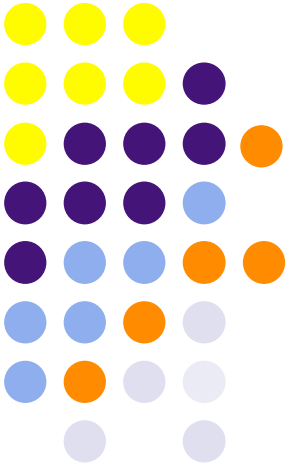
MIDLAB Middleware Laboratory
Università di Roma "La Sapienza" - Dipartimento di Informatica e Sistemistica "Antonio Ruberti"

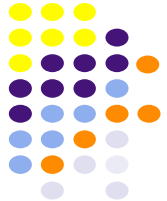
From static to dynamic distributed systems: "the long run"



Roberto Baldoni
Sapienza Università di Roma

53rd Meeting of the IFIP 10.4 WG
Natal, Rio Grande do Norte, 22nd Feb 2008

Context & Motivation: Master Uncertainty in Dist. Sys



- Advent of Complex Distributed Systems (p2p, sensor networks, mobile networks etc…)
  - new problems (information dissemination, content distribution and retrieval, service orchestration and composition, location dependent computing)
  - practical solutions
  - lack of formal framework
- Static Distributed Systems
  - solid theoretical foundations with precise system models (synchronous, asynchronous, eventually synchronous etc.)
  - Precise problem specification (broadcast, failure detectors, consensus, atomic commit etc)
  - provable correct solutions

Uncertainty in Distributed Systems

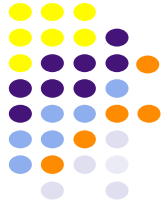
Static Distributed Systems:

- Lack of temporal knowledge
- failures
- unknown communication delays

Dynamic Distributed Systems

- Same as in static distributed systems
- non-monotonic and unknown size of the system,
- neighborhood
- short lifetime as the norm and not an exception

Uncertainty in static distributed Systems:

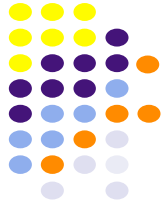


- Unreliable asynchronous distributed systems
  - Lack of temporal global knowledge
  - Process may crash
  - FLP impossibility result
  - Consensus canbe solved (CHT96) through:
    - adding some synchrony to the system (failure derectors)
    - Some process has to be correct
  - Reliable Broadcast can be solved in this model (HT 84) without additional assumptions

What we are talking about…

- Understanding system models of complex dynamic distributed systems
- Understanding basic abstractions (consensus-like, broadcast-like etc) and provide its formal specification in the system model
- Understanding for which abstraction there can be an implementation and based on which assumptions
- Understanding what is a reasonable assumption from a practical point of view to asses technical implementability

Basic Certainties in Static Distributed Systems

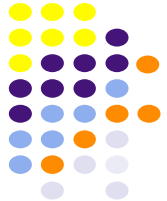


- "I make some assumption on the system model according to the deployment platform"
- "I know the set of processes that will run the computation"
- "I know there is a subset of them that it is correct"
- "I do not know who they are but I know how many they have to be".....by application requirements
  - Do I want to be resilient to byzantine failures?
  - Do I want to be resilient to crash failures?

Assessing assumptions through applications

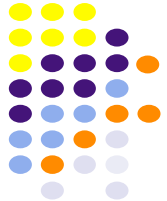
- Air traffic control, Financial systems, Aereospace systems , Egov, Telco service continuity, and many others are examples of static distributed systems
- main characteristics: a predefined setting i.e.,
  - the application knows, directly or indirectly, the set of processes that will participate to the computation.
  - The application also knows if it can exploit synchrony assumptions
- This has a noteworthy consequence: the system can be carefully and "centrally" configured through an appropriate tuning phase in order to get the best performance.
- The application cycle is: Design, deployment, configuration, final deployment, operational
- Can these application be adaptable, scalable etc? Yes they do!

Assessing assumptions through applications



- P2p or mobile applications
- each process autonomously decides to locally run the same distributed application, when joining (it becomes up) and leaving the system
- it is impossible to know the set of processes participating to the computation because it is potentially infinite
- As extremes, in some moments the system could also cease its existence as no process is currently active and at some other moment the system is made of tens, or thousands, of active processes
- The system
  - does not start with a known and pre-defined setting
  - is just the "sum" of all running entities and their local configurations
- each entity has to learn what the system is at run-time in order to successfully reach system goals

Basic uncertainty in Dynamic Distributed Systems

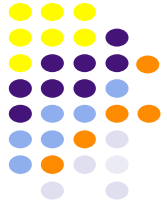


- "each process knows in advance the finite potential set of processes that will run, among this set is always contained the unknown set of correct processes" does not work
- a process that will become up at some arbitrary time and will stay up forever (as the notion of correct process) does not know any finite set that contains the unknown set of processes that will stay up forever.
- reasonably assuming that there exists a set of eventually and permanently up processes in the system (stable core)..otherwise you cannot solve any meaningful problem
- a process should have a way to reach this set without knowing it a-priori.
- we can define useful abstractions only if it is possible to eventually identify among the infinite set of potential running processes those eventually and permanently up.
- identifying the "stable core" among a set of infinite processes is another basic source of uncertainty

Classical Distributed Systems vs Dynamic Distributed Systems

	o Static Distributed Systems:	o Dynamic Distributed Systems
Arrival Model [A04-sigact04]	o Known bound on the number of entities.	o The system is arbitrarily large (finite arrival model with unknown bound or infinite one). Entities freely join and leave the system
locality	o each entity can interact with any other entity in the system (e.g. through TCP connection)	o each entity interacts with an arbitrary small part of the system
failures	o Known bound on the number of failures (usually failures are below a given threshold of processes	o The number of failures is unknown and depends on the arrival model

Uncertainty in Dynamic Distributed Systems



- Eventual Connectivity (i.e., keeping the topology connected) (TB - LADC 06).... *"Coping with dynamics makes **maintaining** even a **basic property** like overlay **connectivity** a very difficult task" Eli Upfal,ACM* FOCS 01

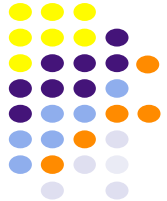
- One-time Query problem solvability (BBRT - PACT 07)

The Connectivity problem

The connectivity problem

- Dynamic Distributed System abstracted by a **dynamic graph** (also called overlay network):
  - Vertices: all system entities
  - Arcs: neighborhood relations
- Static Distributed Systems:
  - Most of the times communication graph is a clique
- The dynamic graph is on top and independent of physical networks
- The graph has to be CONNECTED along the time to allow for example processes to communicate!

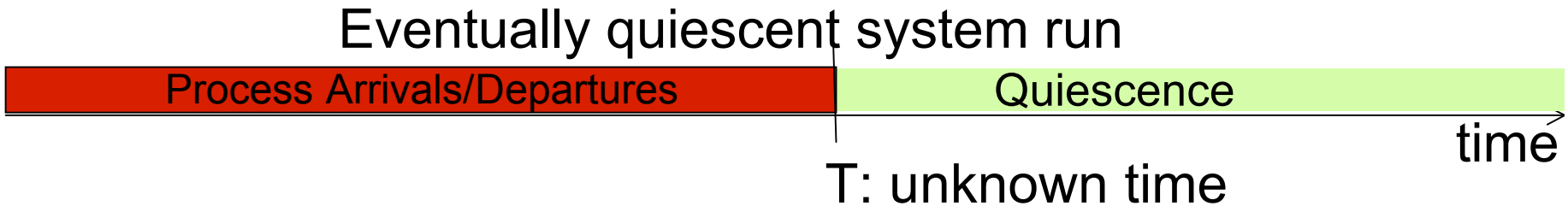
Problems to face for ensuring connectivity



- Adoption of a distributed computing model where the connectivity problem can be solved
- Formal specification of the connectivity problem
- A protocol which satisfies the specification under the proposed model

Computational Model

- Finite arrival model
  - Infinite set of processes P (… i, j, k…)
  - Only **finitely many** might be added to the overlay for any system run
  - Any number of processes can be simultanously added to/deleted from the overlay (i.e., the access to the overlay is not serialized)

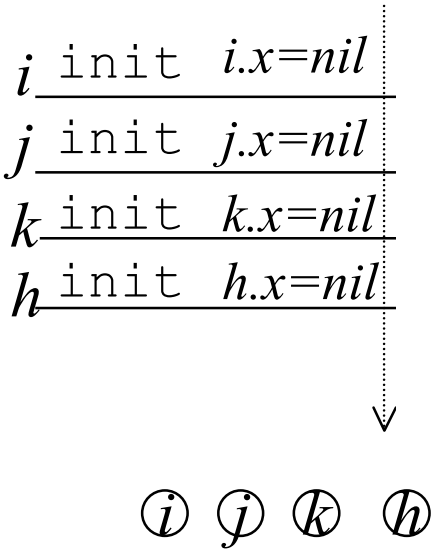


- Other classical assumptions:
  - A pair (i,j) communicate by a reliable link
  - The system is asynchronous
  - A process may be correct or faulty. A faulty process fails by crashing

Connectivity Specs: Overlay Definition

- Each *i* has a set of ***neighbor variables*** *i.X* of $x \in \Pi \cup \{nil\}$
- ***Overlay O(t):***

  the set of pairs (i, i.X): $\exists\ i.x \in i.X \neq nil$ *at time t*
- Any process in the overlay is called ***vertex***
- Process actions: `init, join, leave, stop`

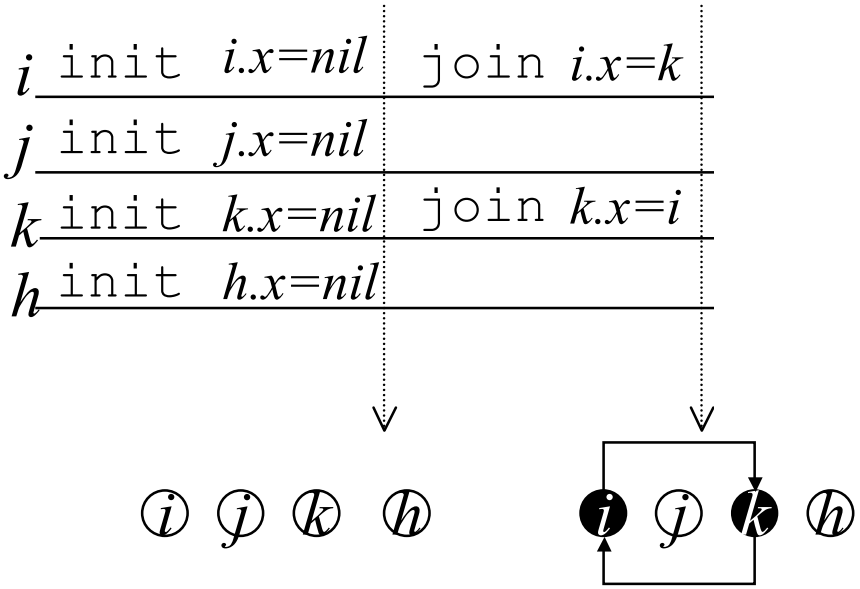


topological overlay representation -- vertices black bullets

Connectivity Specs: Overlay Definition

- Each *i* has a set of ***neighbor variables*** *i.X* of $x \in \Pi \cup \{nil\}$
- ***Overlay O(t):***

  the set of pairs (i, i.X): ∃ i.x ∈ i.X ≠ nil at time t
- Any process in the overlay is called ***vertex***
- Process actions: `init, join, leave, stop`

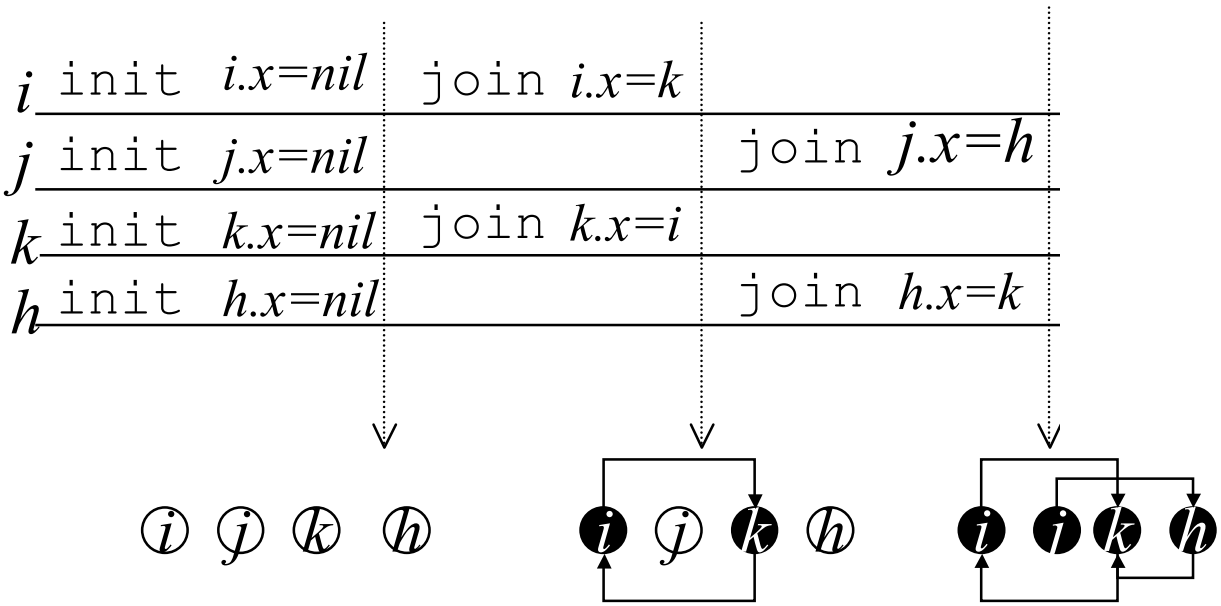


topological overlay representation -- vertices black bullets

Connectivity Specs: Overlay Definition

- Each *i* has a set of ***neighbor variables*** *i.X* of $x \in \Pi \cup \{nil\}$
- ***Overlay O(t):***

  the set of pairs (i, i.X): $\exists\ i.x \in i.X \neq nil$ *at time t*
- Any process in the overlay is called ***vertex***
- Process actions: `init, join, leave, stop`



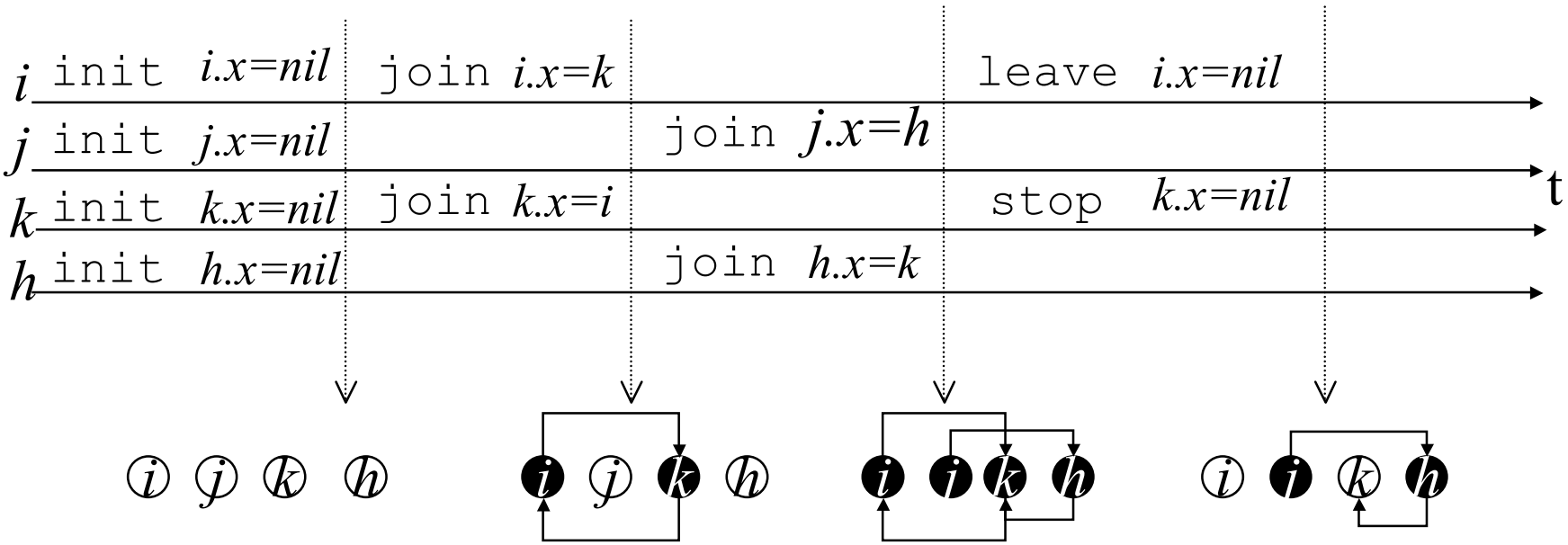
topological overlay representation -- vertices black bullets

Connectivity Specs: Overlay Definition

- Each *i* has a set of ***neighbor variables*** *i.X* of $x \in \Pi \cup \{nil\}$
- ***Overlay O(t):***

  the set of pairs (i, i.X): $\exists\ i.x \in i.X \neq nil$ *at time t*

- Any process in the overlay is called ***vertex***
- Process actions: `init, join, leave, stop`



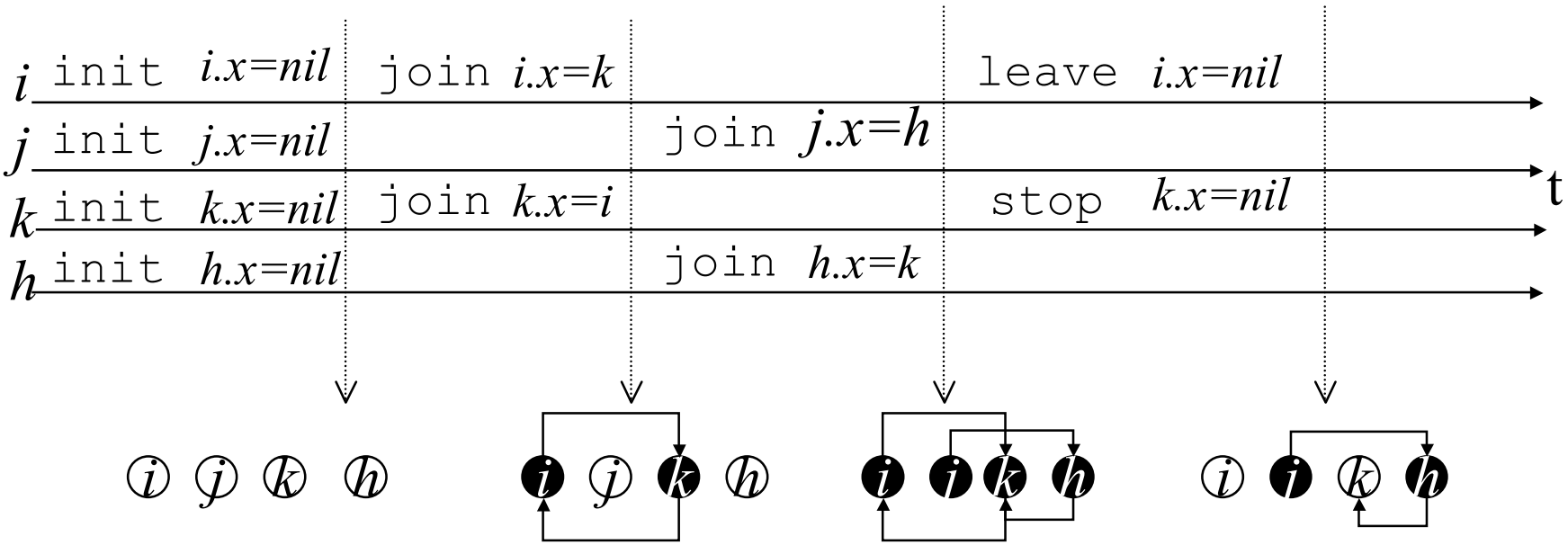
topological overlay representation -- vertices black bullets

Connectivity Specs: Overlay Definition

- Each *i* has a set of ***neighbor variables*** *i.X* of $x \in \Pi \cup \{nil\}$
- ***Overlay O(t):***

  the set of pairs (i, i.X): $\exists\ i.x \in i.X \neq nil$ *at time t*

- Any process in the overlay is called ***vertex***
- Process actions: `init, join, leave, stop`



topological overlay representation -- vertices black bullets

Connectivity Specs: Basic Definitions

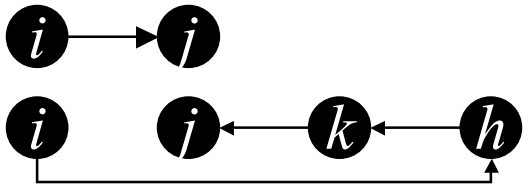
direct path: $i \rightarrow_t j$

for each i,j: (i, i.X), (j, j.X) ∈ O(t)

iff at time t one of the following conditions hold

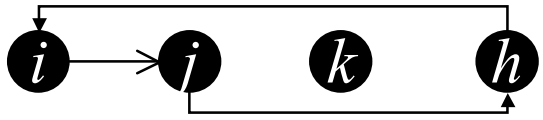
– $\exists\, i.x \in i.x = j$

– $\exists\, k \in \Pi : (i \rightarrow t\ k)$ AND $(k \rightarrow t\ j)$

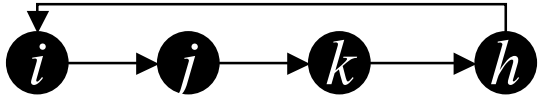


i and j are strongly connected iff:

$i \rightarrow_t j$ *AND* $j \rightarrow_t i$



O(t) is strongly connected *at time t iff any pair of vertices is strongly connected at time t*



Connectivity specification

- Eventual Strong Connectivity

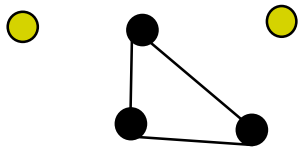
  overlay eventually strongly connected

- No Lockout

  if a correct $i \in \Pi$ executes `join` *at time t, then* $\exists t', j$:
  $t' > t$ *AND* $(i, i.X) \in O(t')$

Impossibility of Avoiding Partitions

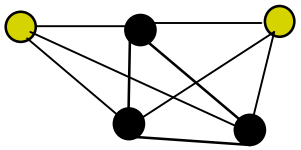
- Starts from the impossibility of maintaining a clique over time when the ***access to the overlay is not serialized***
  - Two concurrent joining processes (i,j) may not see each other when connecting to the overlay O(t)



O(t)

Impossibility of Avoiding Partitions

- Starts from the impossibility of maintaining a clique over time when the ***access to the overlay is not serialized***
  - Two concurrent joining processes (i,j) may not see each other when connecting to the overlay O(t)



O(t') with t'>t

Impossibility of Avoiding Partitions

- Starts from the impossibility of maintaining a clique over time when the ***access to the overlay is not serialized***
  - Two concurrent joining processes (i,j) may not see each other when connecting to the overlay O(t)

O(t'') with t''>t'

- Conclusion
  - Due to the arbitrary number of concurrent join and leave operations, no protocol can ensure connectivity during perturbed periods even if considering k-connected graph structures (for any finite k)

Impossibility: operative follow up

overlay partition during perturbed period is a matter of life……

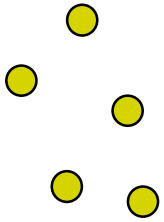
…….. any overlay connectivity protocol can achieve connectivity only in quiescent periods……..

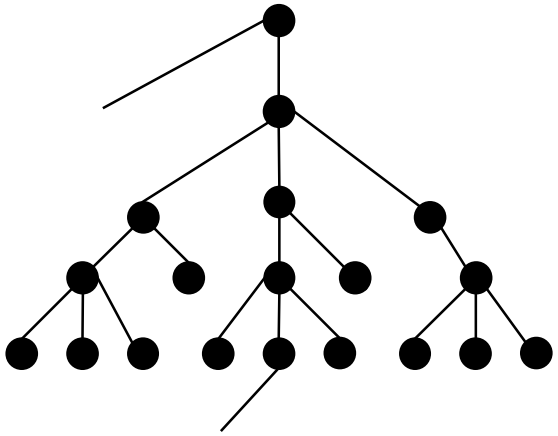
…….. but no protocol can detect when quiescence begins……..

…….. thus, any protocol has to do its best during the perturbed period to face partition occurrences: ***any protocol must detect any possible partition and subsequently recover it***

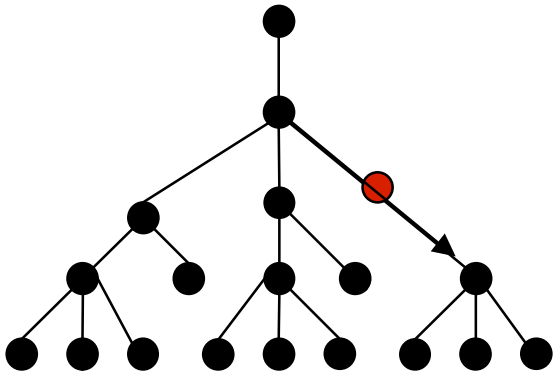
The Tree Algorithm

The algorithm arranges nodes in a rooted tree topology



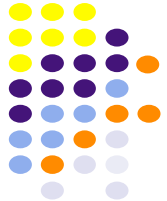


new nodes add as leaf

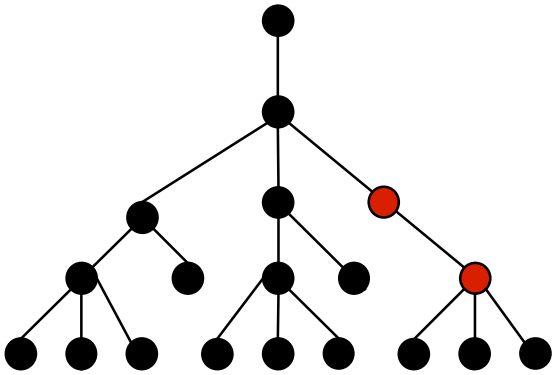


before leaving, creates a new weak connection

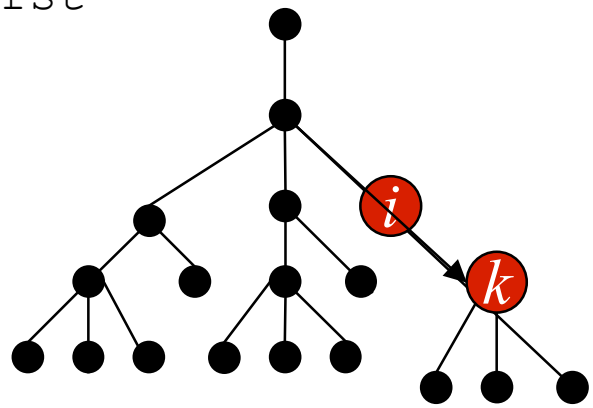
The Tree Algorithm - concurrent leaves



Conflicting leaves

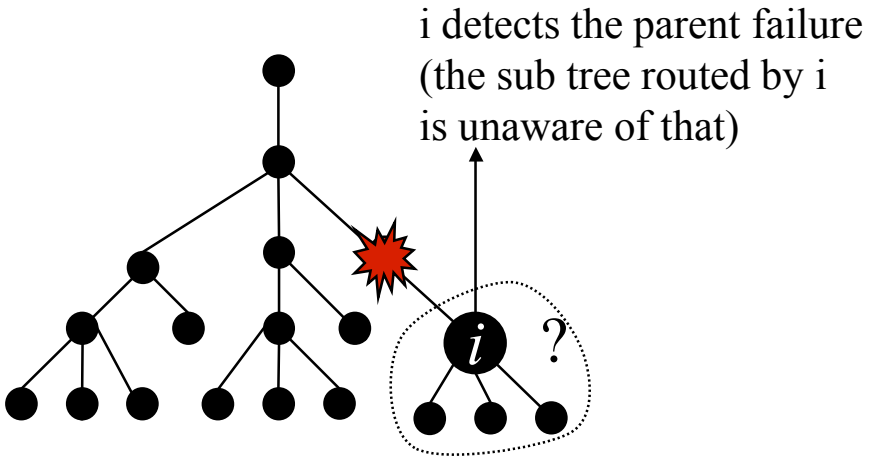


vertex closest to the root leaves first

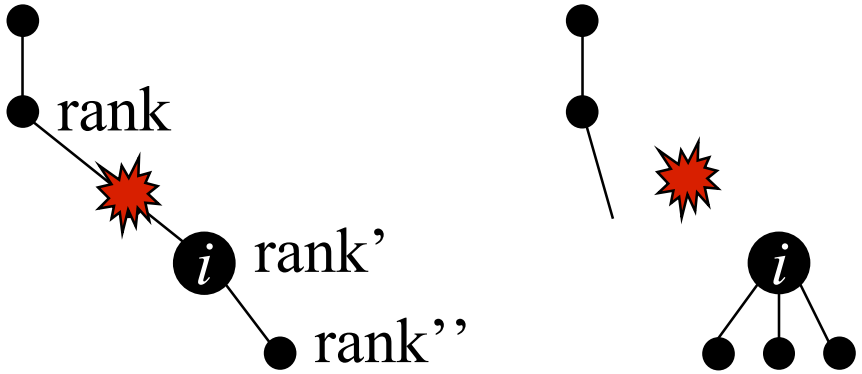


The Tree Algorithm - dynamic restoring

Failures

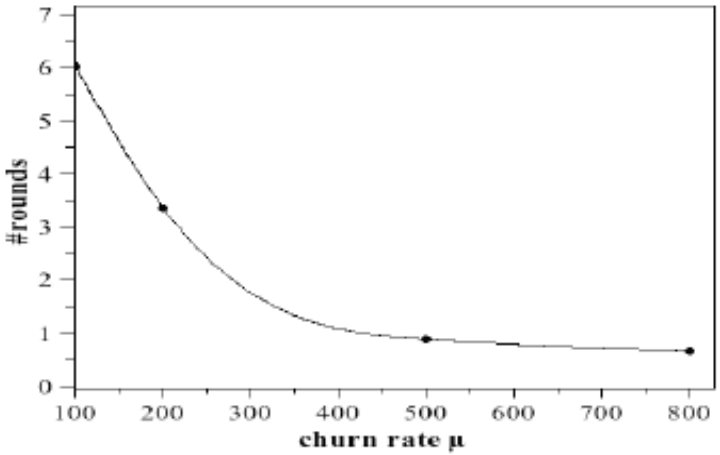


rank usage to avoid a reconnection to its subtree:rank'<rank''



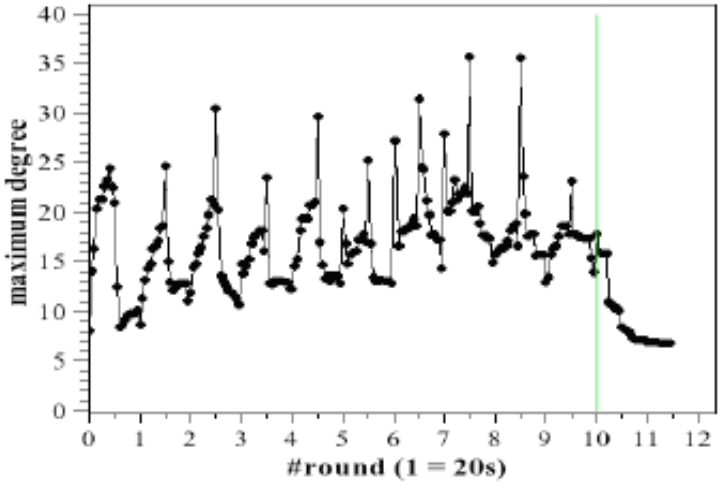
Performance Evaluation of the "extended" Tree Algorithm

- dynamics modeled as churn,
  - churn rate μ defined as μ joins and μ leaves in each round (Hopcroft, PODC05)



- star effect for the Tree algo
  - 1000 nodes
  - convergence to a star

- introduction of a "dynamic rebalancing mechanism"
  - contends with the star-effect
  - overlay converges to a tree with height 8 and maximum degree 7

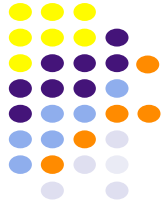


One-time Query problem solvability

System Models

- Assume connectivity
- A model is denote as $M^{N,D}$ where
  - N is the number of processes
    - bounded in all runs (b)
    - bounded in each run but unbounded considering the runs'union (n)
    - Unbounded (∞)
  - D is the diameter of the graph
    - bounded and known, (b)
    - bounded and unknown (n)
    - Unbounded (∞)
- Example of models: $M^{b,b}$, $M^{b,n}$ ..... $M^{\infty,b}$ $M^{\infty,\infty}$

One-time Query problem (BGGM – SIGMOD04)



- A process (node) issues a query in order to aggregate data that are distributed among a set of processes (nodes).
- The issuing process does not know
  - (i)if there exist nodes holding a value matched by the query,
  - (ii) where these nodes are and
  - (iii) how many they are.
- The One-Time Query problem informally requires:
  - TERMINATION: The query, issued by a node *pi terminates*
  - VALIDITY: The query has to aggregates *at least* all the values held by the nodes that are in the system and are connected to *pi* during the whole duration of the query (query time interval).

One-Time Query problem solvability

- (BGGM – SIGMOD04) provides an algorithm that work only in monotonous network where only edges can be removed due to leaving of processes. So it does not solve the problem in any $M^{*,b}$ model.

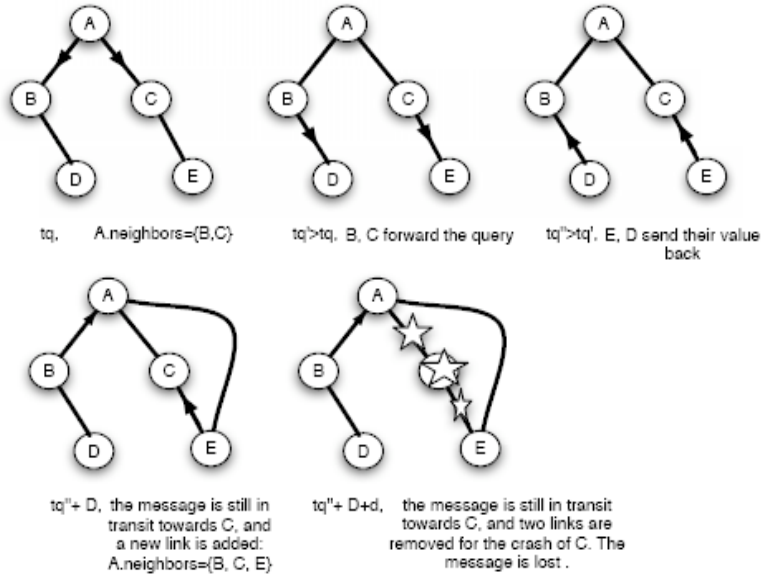


Fig. 2. Bug Example

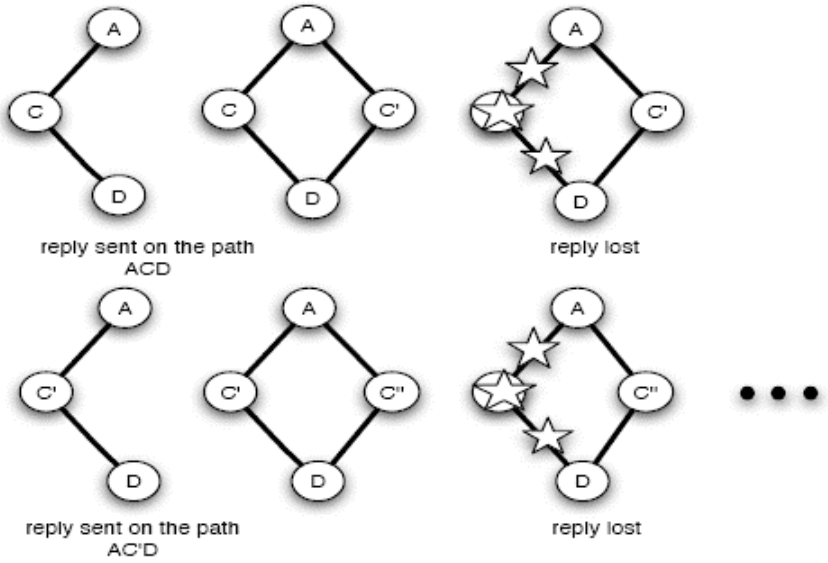


Fig. 3. Bad Pattern of Graphs Changing

One-Time Query problem (revised)

SPEC:

- Termination: *query*(*Q*) completes in a finite time.
- Dynamic validity: For each run, *query*(*Q*) will compute the result including in *V* at least the values held by each process that, during the whole query interval, remains connected to the querying process through a subgraph of the graph *G* that represents the network at the time the query is started.

Revised One-Time Query problem solvability

- The problem is solvable $M^{\infty,n}$ using perfect failure detectors

- The problem is impossible to solve in $M^{\infty,\infty}$ due to a race among messages that arrives at a process p and the joining of new processes.

The few lines to remember

- Understanding the model of a dynamic distributed systems is yet an open problem
- Even very simple problems like basic connectivity become complex……
- …and even assuming connectivity is always hard to find algorithms that are able to solve simple problems in such weak system models