Election in Dynamic Networks

Agenda

- The bully algorithm
- Election in an ad hoc wireless network

References

Chapter 6.4 of “Distributed Systems by M. van Steen and A. S. Tanenbaum”

The Bully Algorithm

Proposed by Garcia-Molina in 1982

The idea: an entity tries to elect itself as the leader. “The biggest guy in town wins”

Assumptions

- The system is **synchronous**
- Entities may **fail at any time**, including **during execution** of the algorithm
- A **failure detector** detects failed entities
- Message delivery is reliable
- The topology is a **complete graph**: each entity knows its own id and that of every other entity, but do not know which entities are **dead or alive**

Synchronous Systems

Strong assumption on the local clocks of the entities and on the communications delays

Synchronized clocks => all clocks are incremented simultaneously

Bounded communication delays => there is an upper bound on the communication delays experienced by entities (**T**)

How Estimate Possible Failures?

In a synchronous system we can estimate if a entity fails by using heartbeat messages

- The entity x sends to all its neighbors a message
- If a neighbor does not reply by a given interval of time (threshold), it is considered failed

  Threshold = the easiest choice a multiple of **T**

  Local timer + timeout event

The Kind of Messages

There are three kinds of messages

1. **Election**: sent to announce/start an election
2. **Answer (alive)**: the response to the election message
3. **Coordinator**: sent by the leader to notify its victory

The Algorithm

An entity x starts the protocol (many initiators are possible) when it discovers that

- The current coordinator failed or
- x has just restarted (recover from a failure)

The Algorithm

1. If x has the highest id number, it sends the **Coordinator** (notification) message to all other entities in the network
2. Otherwise, it broadcasts an **Election** message to all other entities with higher process IDs than itself
3. If x receives no answer after sending an **Election** message, then it broadcasts a notification message to all other nodes and becomes the leader

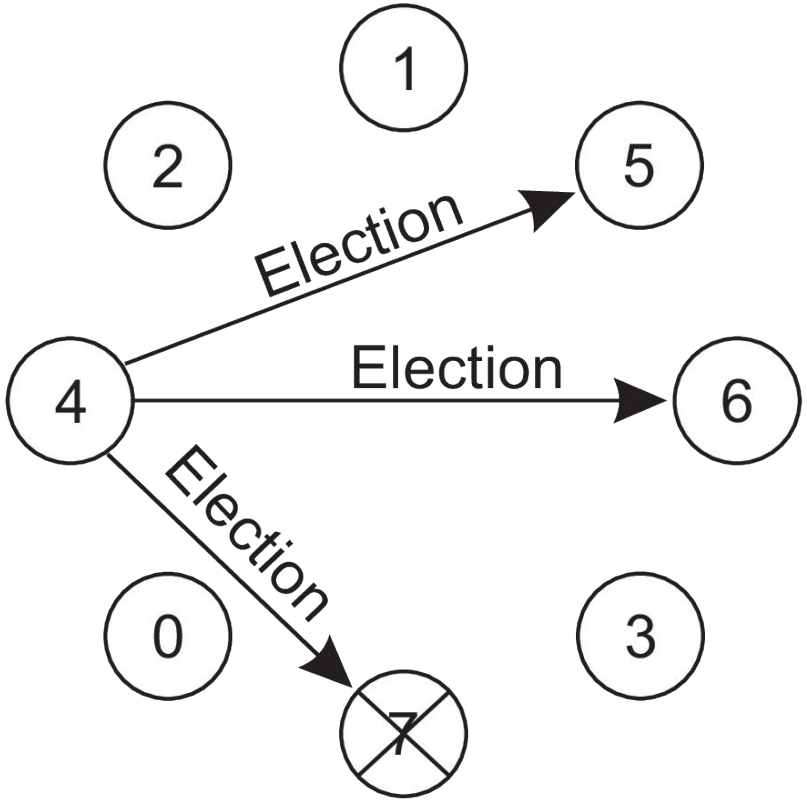
4. If x receives an answer from a node with a higher ID, x is defeated and waits for a notification message

5. If there is no notification message after a fixed period of time, x restarts the process from the beginning

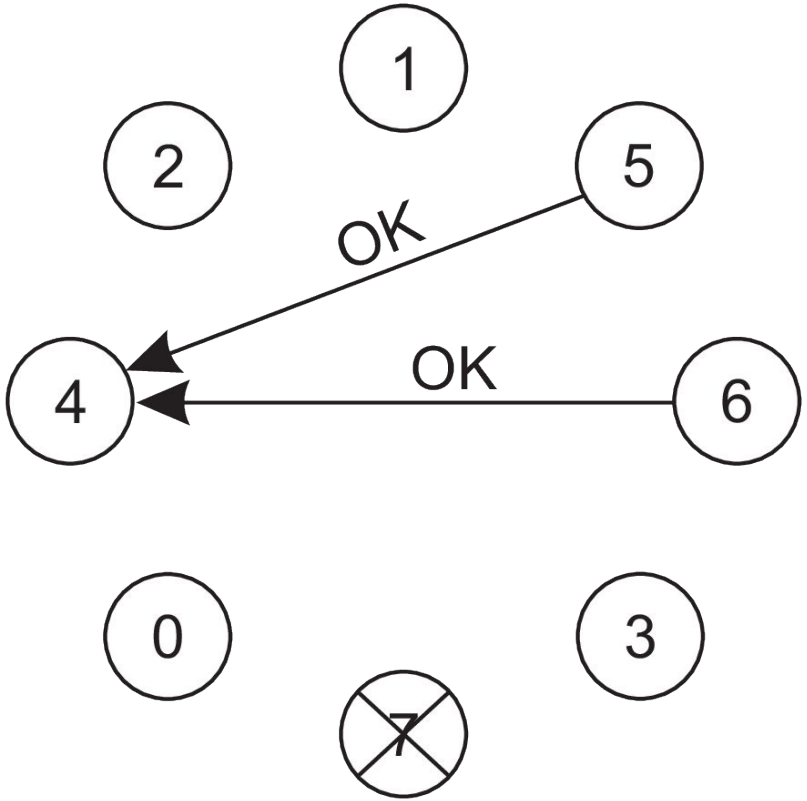
6. If x receives an election message from another entity with a lower ID, it sends an answer message and starts a new election process from scratch, by sending an Election message to higher-numbered entities

7. If x receives a notification message, it treats the sender as the coordinator

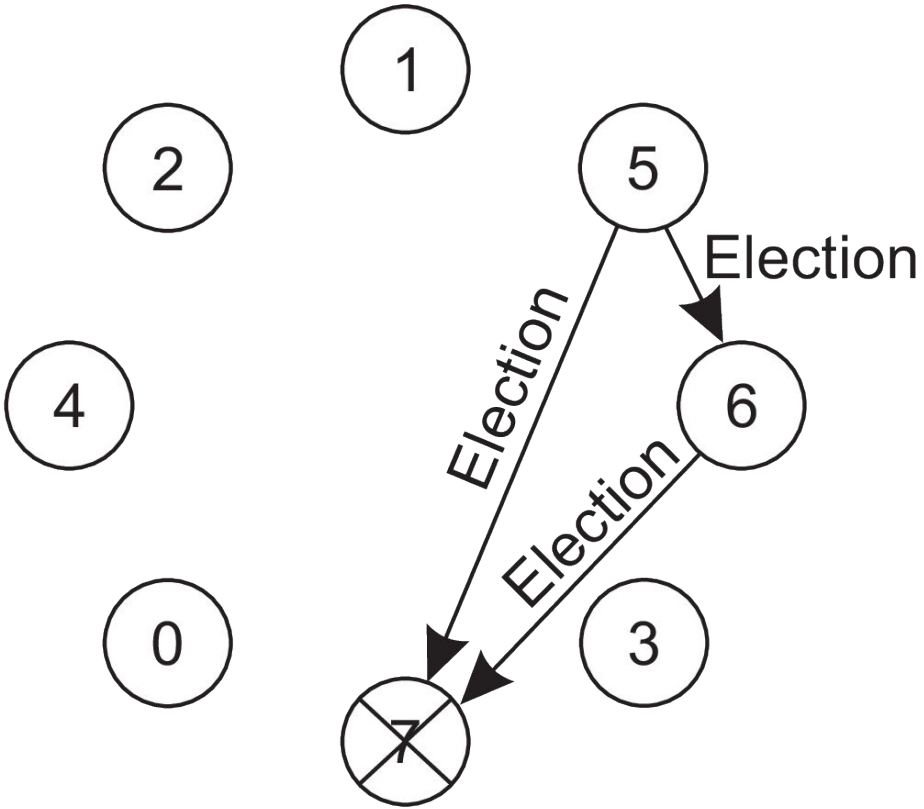
Example



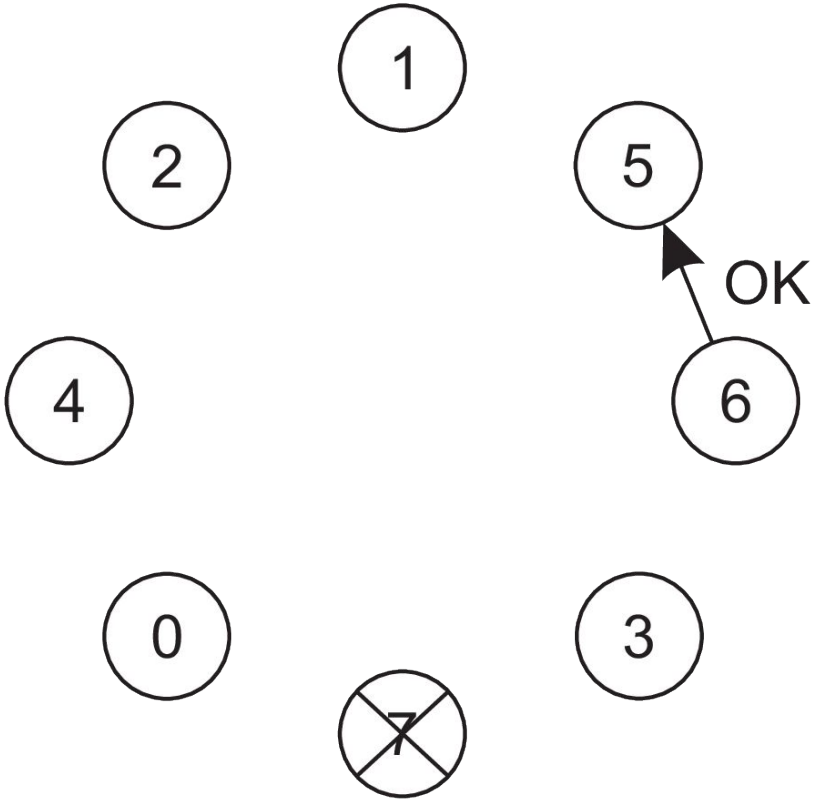
Example



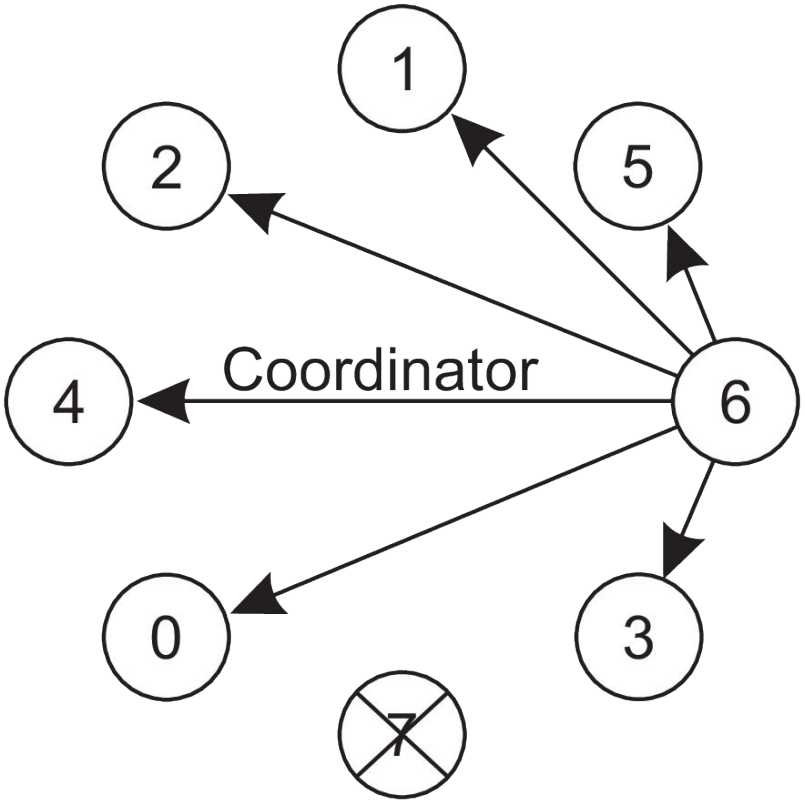
Example



Example



Example



Message Complexity (Worst Case)

The entity with the lowest id starts the election

Id 1 -> n - 1 mgs

Id 2 -> n - 2 mgs

....

Id N-1 -> 1 mgs

$$\sum_{i=1}^{n} i \approx \Theta(n^2)$$

There are also the n notification messages

Election in Wireless Environments

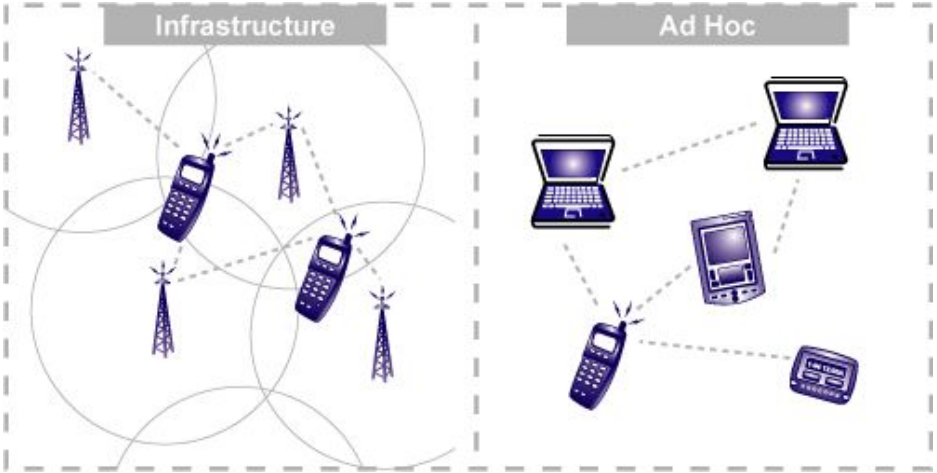
In wireless network

- Message-passing is not reliable
- The topology of the network changes
- Entities knows only their neighbors (that may change)

The algorithms for leader election we described do not work with a changing topology

Ad-hoc Mobile Wireless Network

- There is no pre-existing infrastructure
- Each node participates in routing by forwarding data for other nodes
- The routes of data are dynamically determined on the basis of network connectivity and the routing algorithm



Features of Mobile Wireless Network

- Highly performing network
- No expensive infrastructure must be installed
- Quick distribution of information around sender
- No single point of failure
- Very dynamic topology since all entities may move

Leader Election

Problem statement

Given a network of entities with a value, after a **finite number of topological changes**, every **connected component** will eventually select a unique leader

The leader => the node with the highest value

Assumptions

- Each entity carries a value => performance related feature, e.g., battery power, computational capabilities
- Unique & ordered id
- Bidirectional and FIFO links
- Nodes may move, fail and restart => arbitrary topology changes such as network partitioning and merge
- Multiple initiators

How Does The Process Start?

- The leader of a connected component periodically send heartbeat messages to other nodes
- If a node does not receive a heartbeat message from the leader for a given period of time, it triggers the election process
- Multiple nodes can detect the leader departure and starts the process

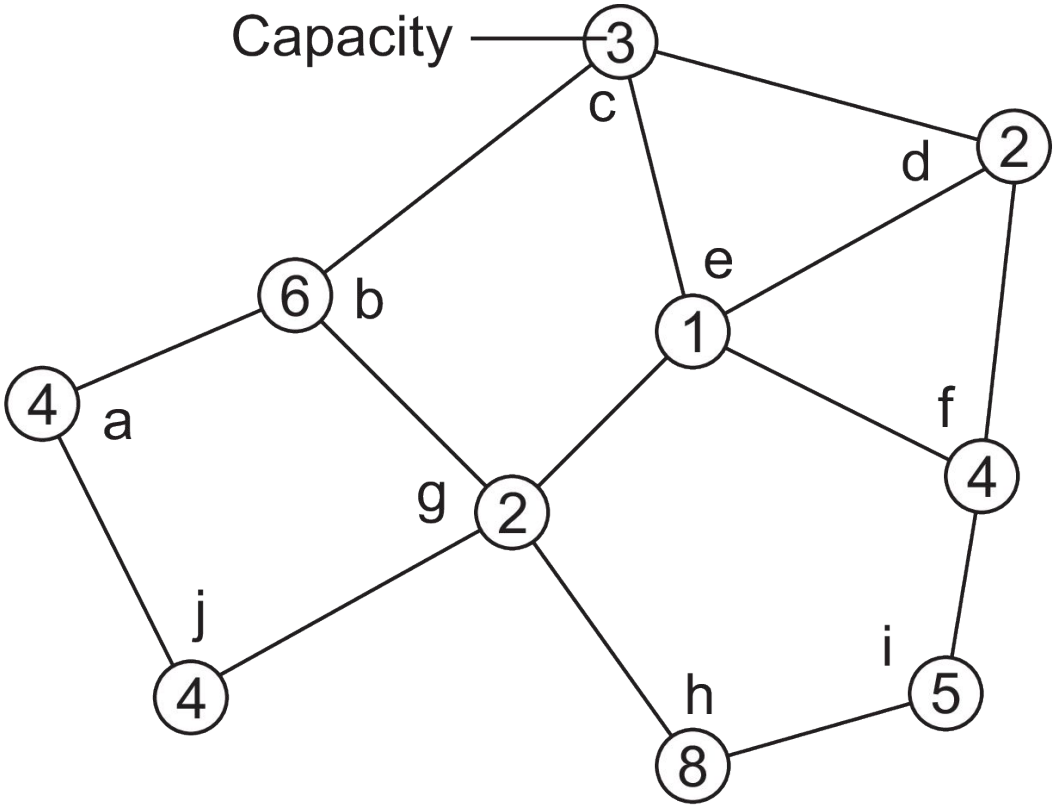
Basic Schema (Single Initiator + No Failures)

- The initiator starts building a SPT
- Leaves start a convergecast computing the best candidate to be the leader
- The root of SPT notify the leader to all entities

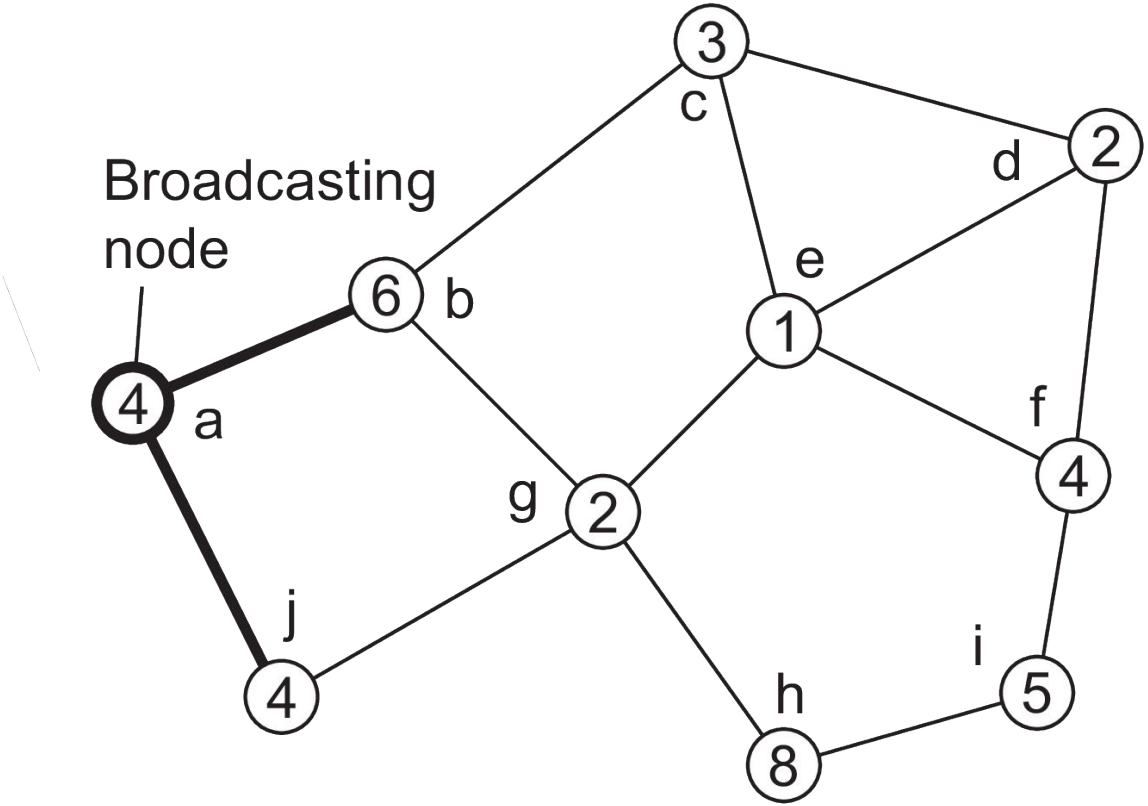
Three kinds of messages

1. **Election** => start the process
2. **Ack** => join the spanning tree & carry info about candidate
3. **Leader** => leader notification

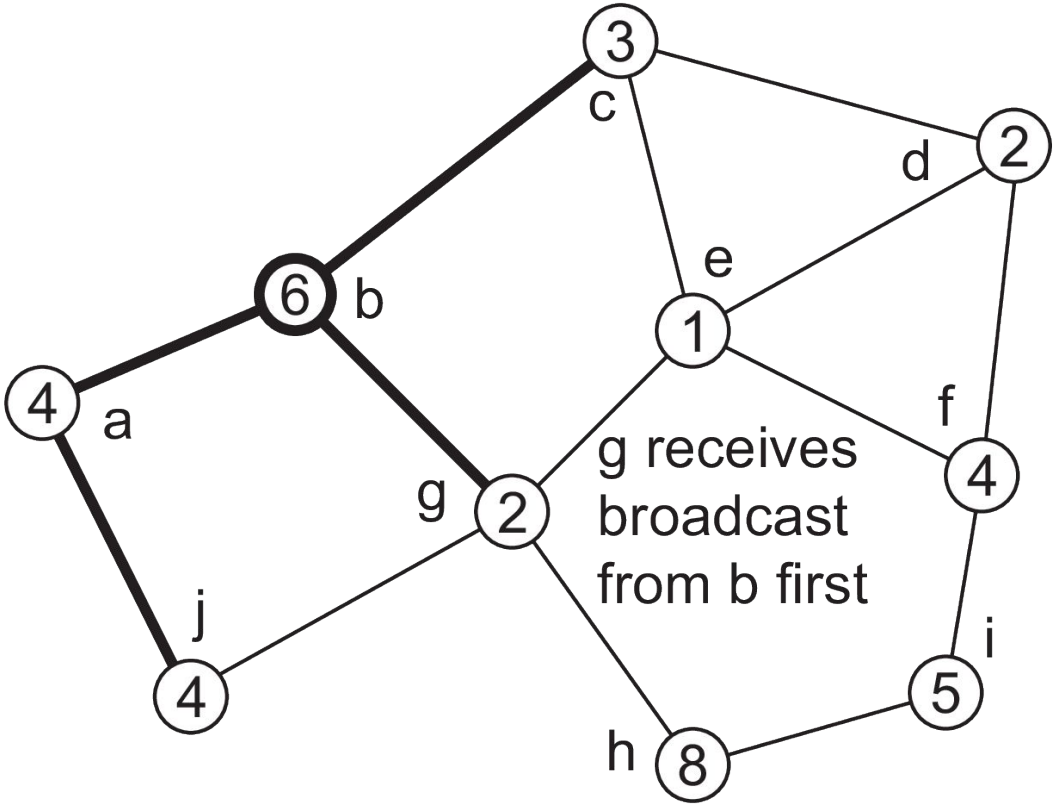
Example



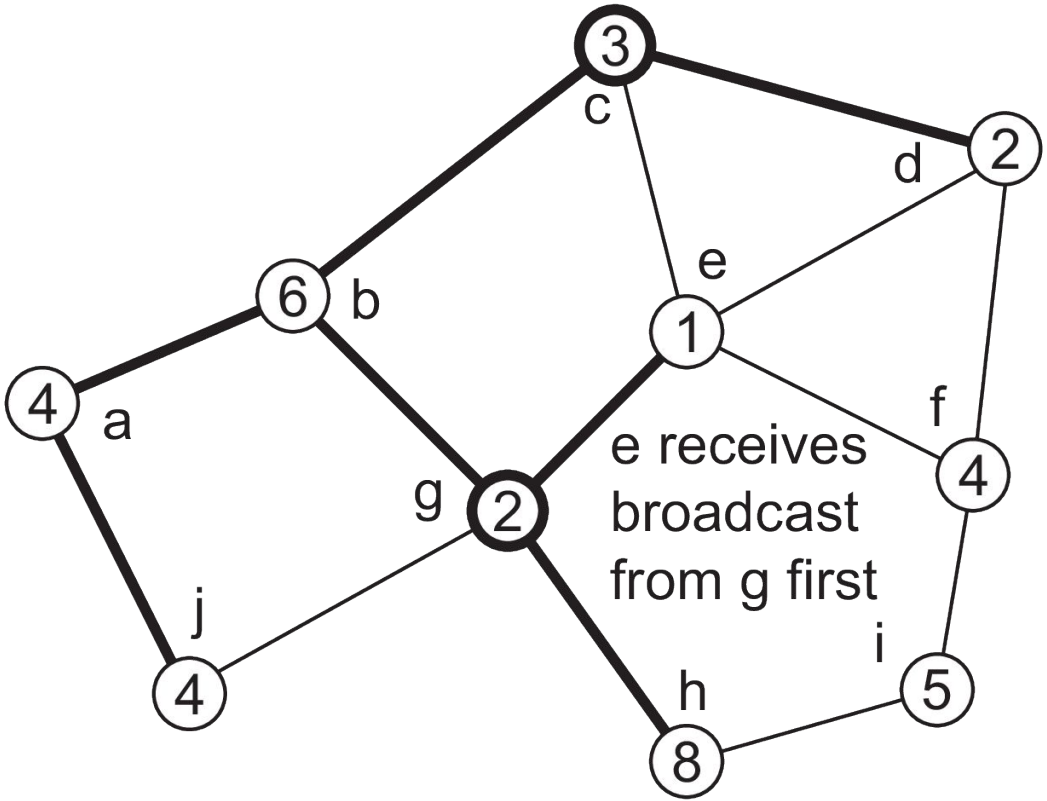
Example



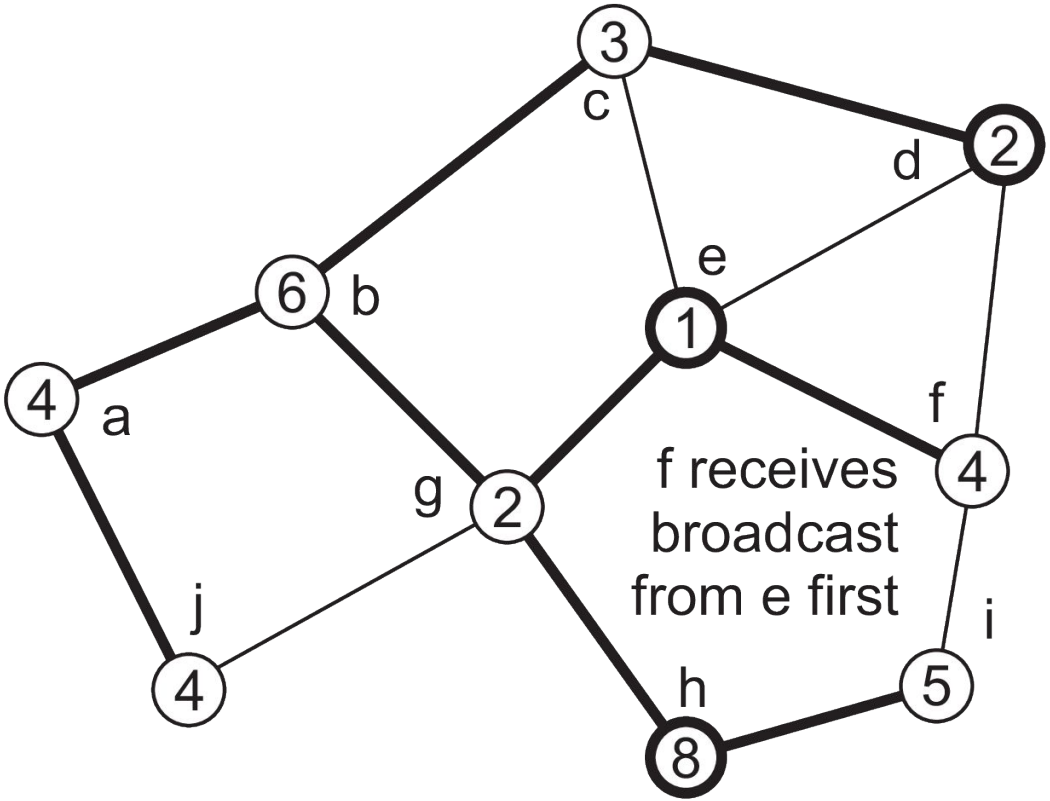
Example



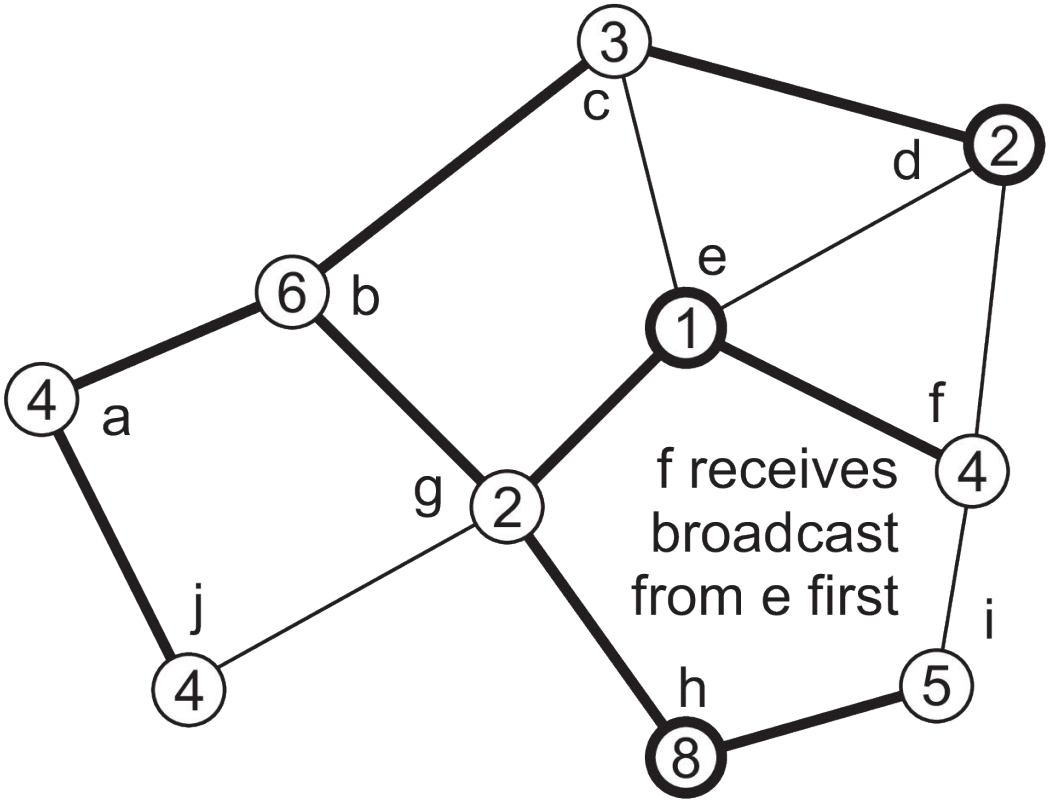
Example



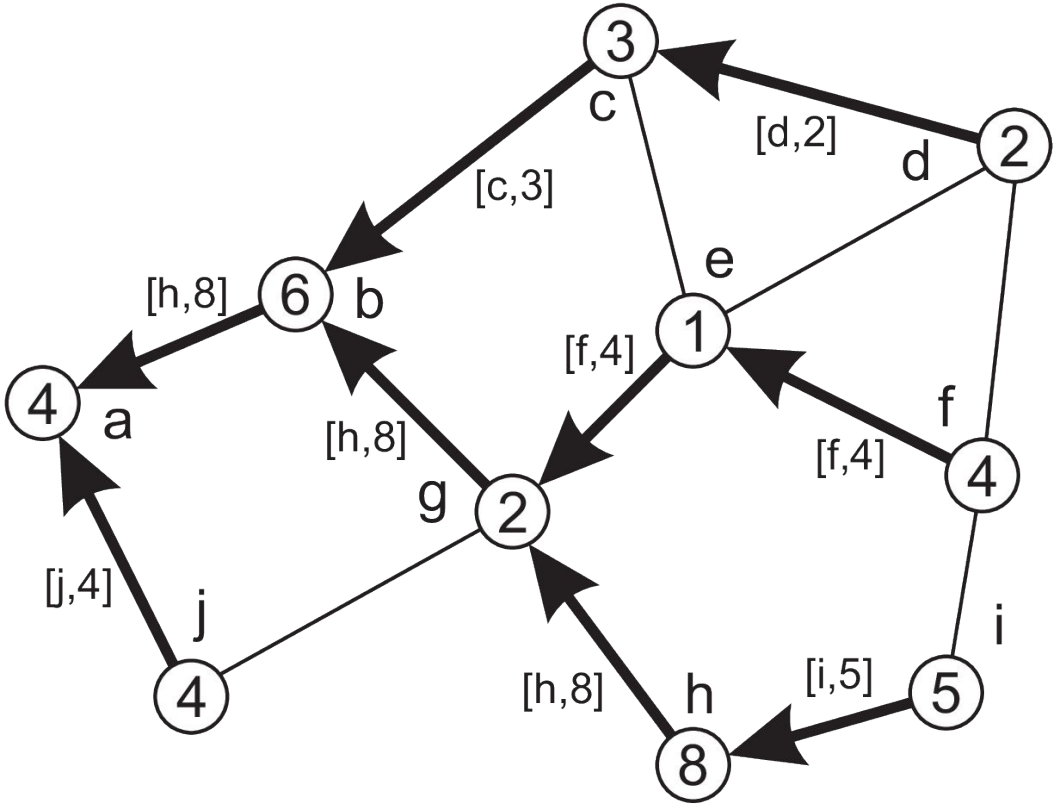
Example



Example



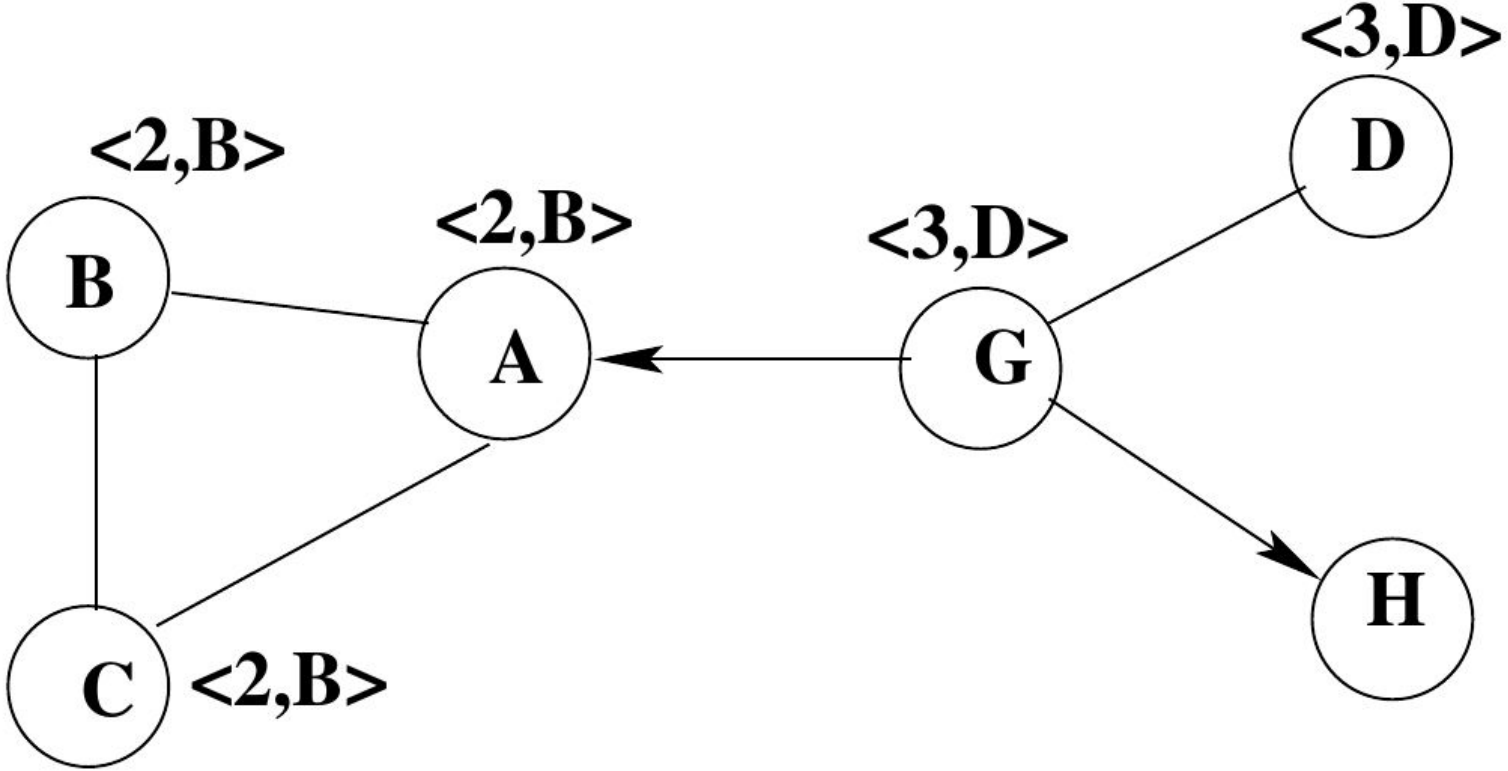
Example



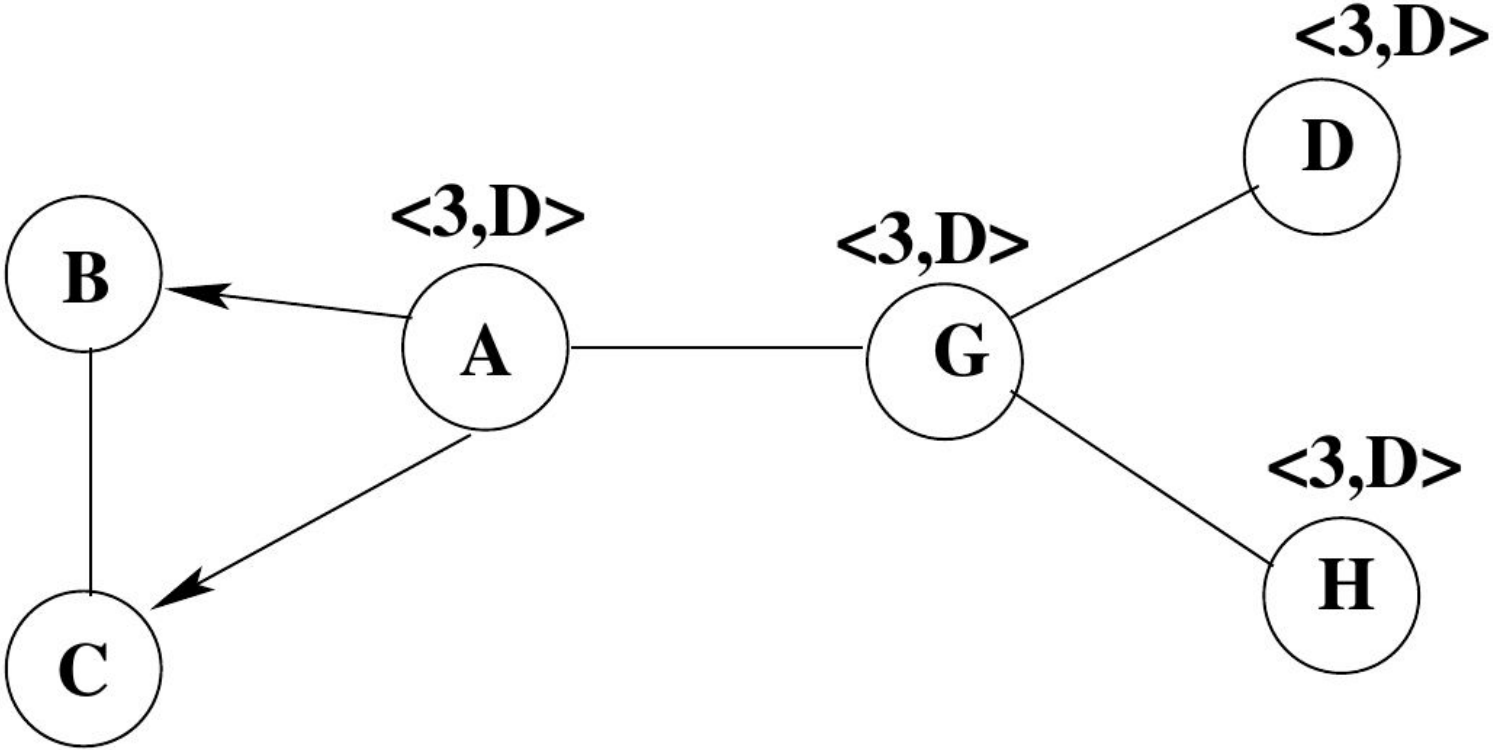
Handling Concurrent Computation

- A node can take part only in one election process (computation)
- Computation-index identifies computations and are totally ordered
- When a node x receives a message belonging to a computation with a higher index, x stops its current computation and joins the one with higher index

Example



Example



Handling Node Partition

An entity x waits for acks messages from its children before sending an ack to its parent

What happens when a child of x, call it y, is disconnected?

If x detects that y is disconnected, x stops waiting for the ack and ignores y

If y can no longer report its ack, it terminates the process by acting as the root and notifying the leader

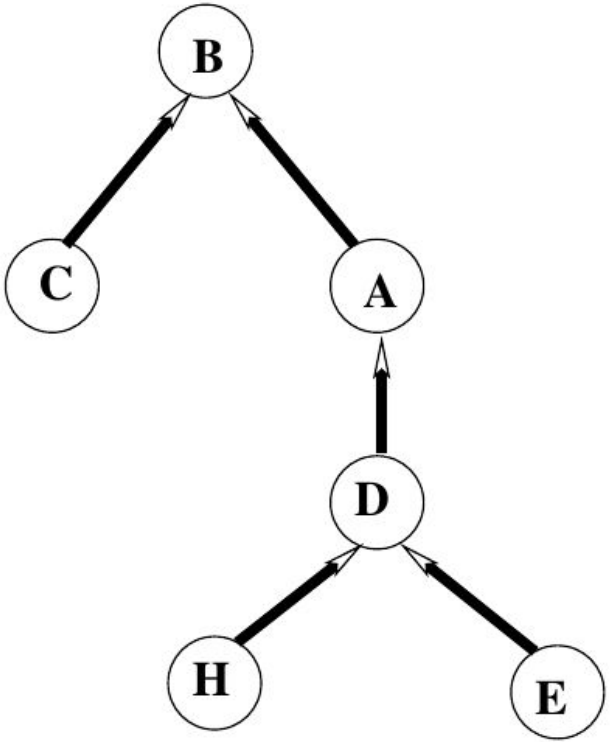
How to Detect Node Disconnection?

- Each node in the spanning tree sends periodically **Probe** msgs to all nodes that is waiting for an ack
- A node receiving a **Probe** replies with a **Reply** msg
- If no **Reply** arrives the child is considered disconnected

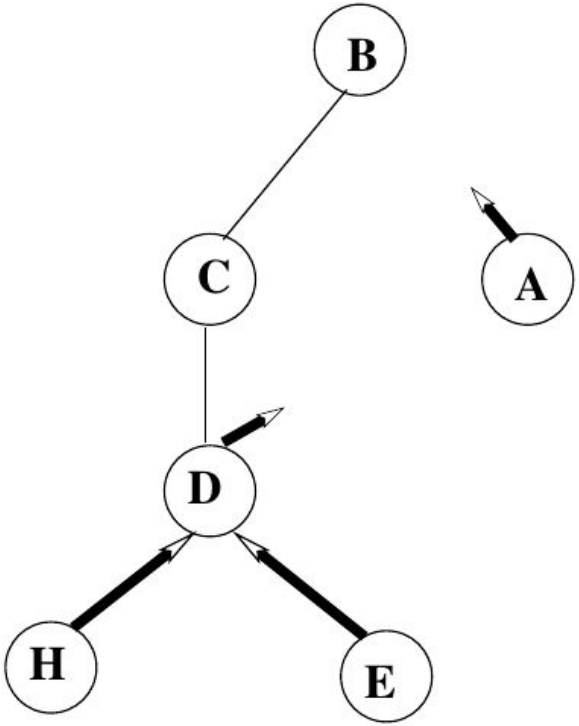
Further kinds of messages

1. **Probe** => ask if a node is still connected
2. **Reply** => confirm the presence of a node

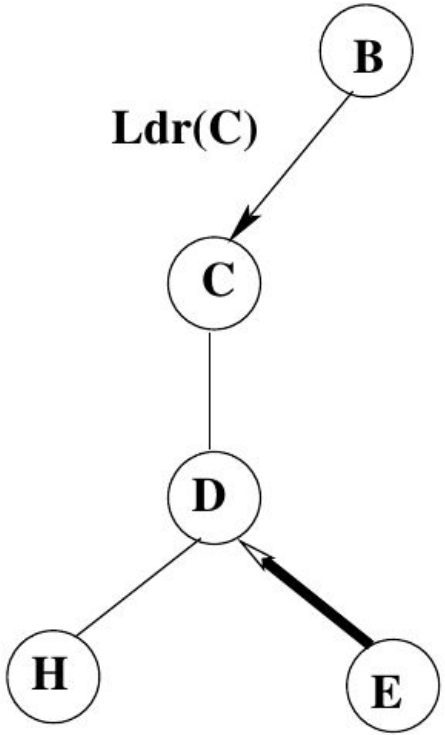
Example



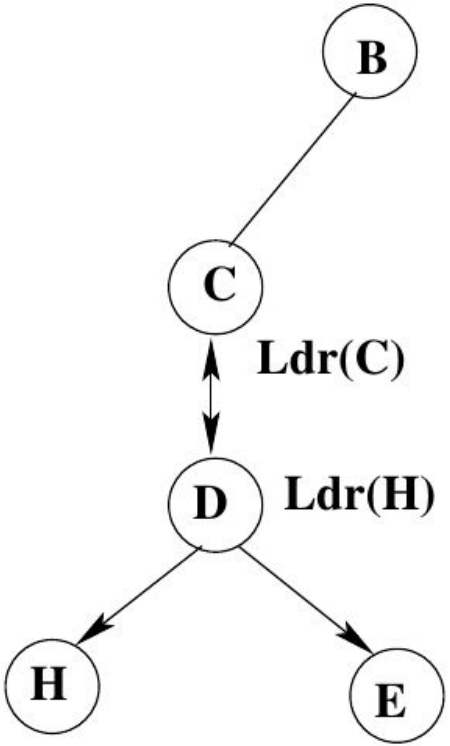
Example



Example



Example

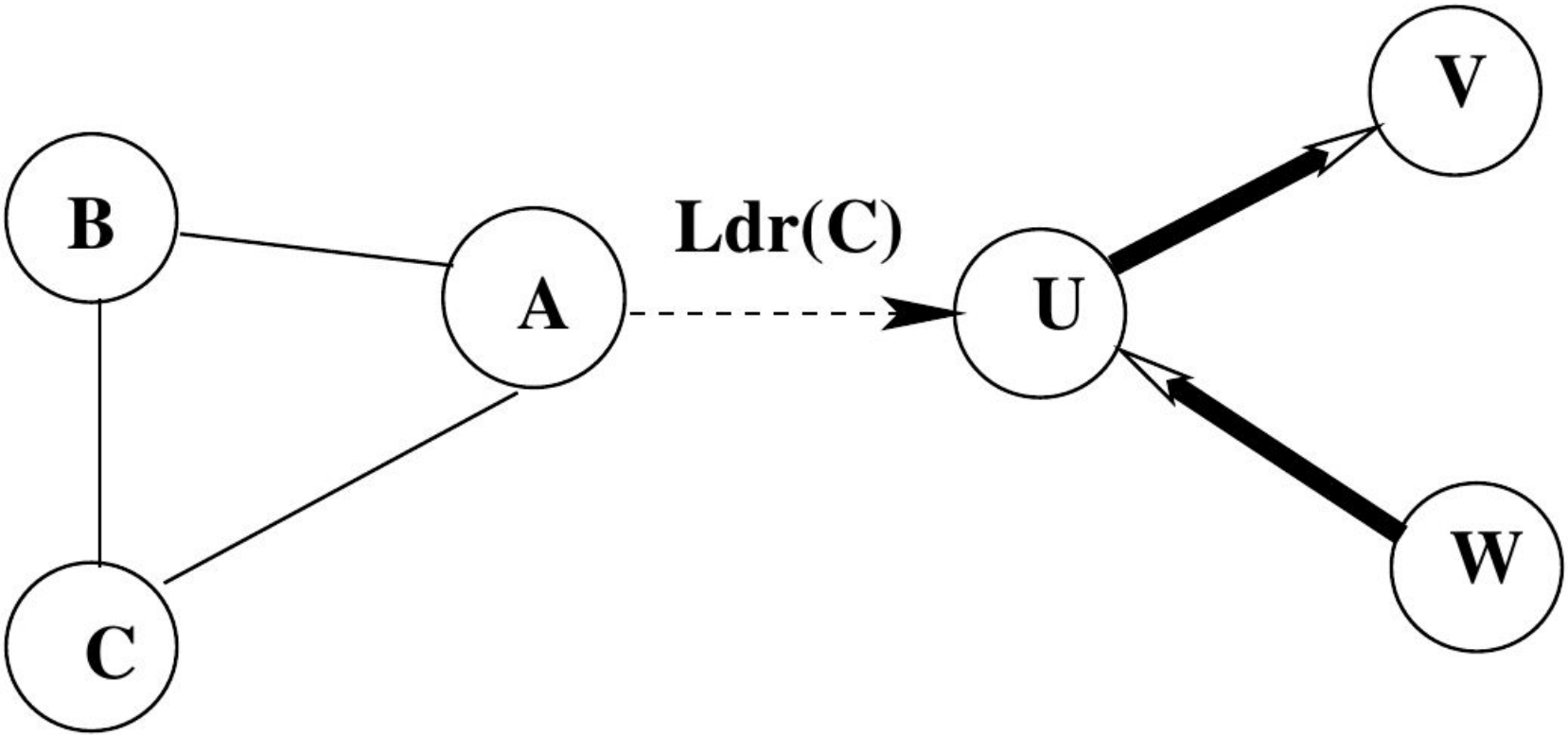


Merge Partitions

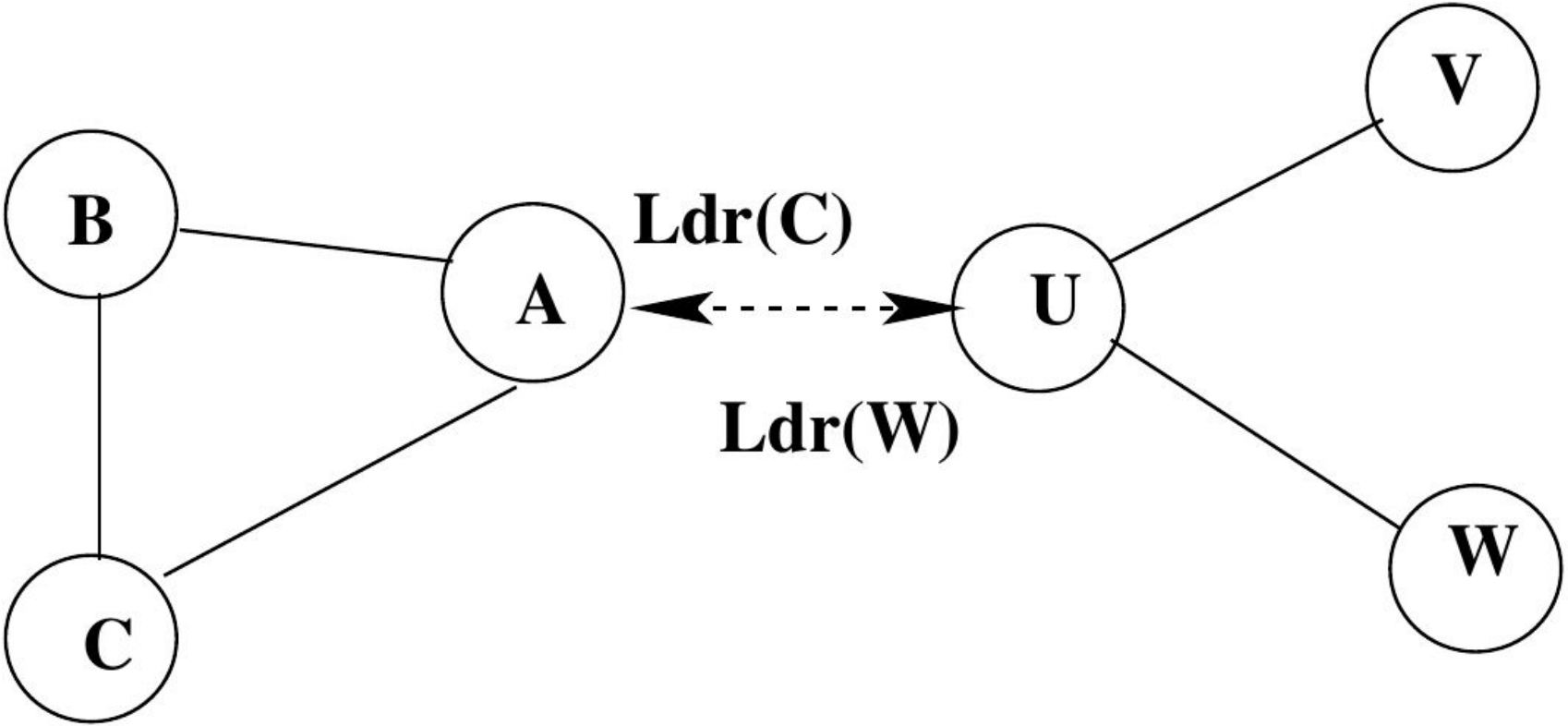
Node mobility can cause partition to merge together

1. Two connected component with their leader merge together => the leader with highest value wins
2. A connected component has a leader, the other is running the election process => let the election process terminate and then selects the best leader

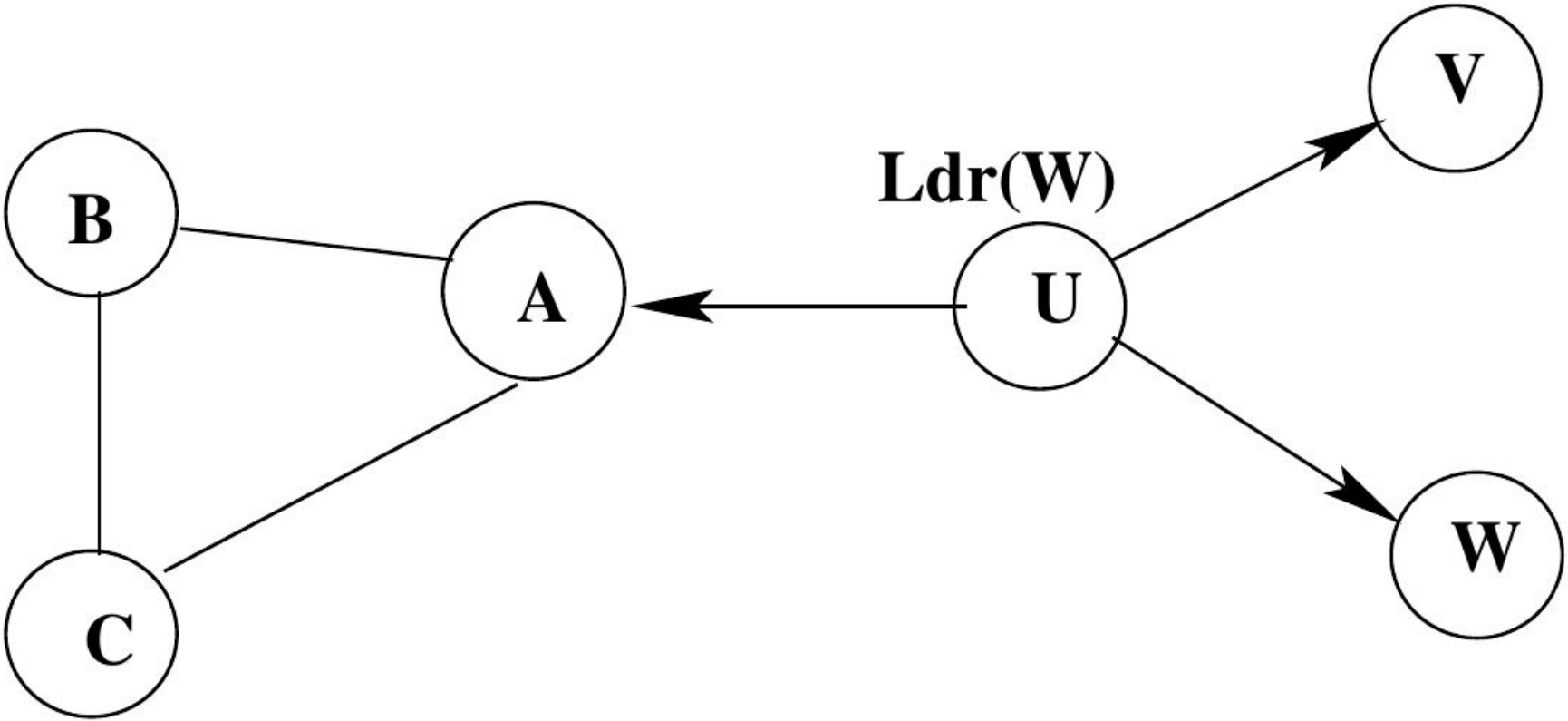
Example



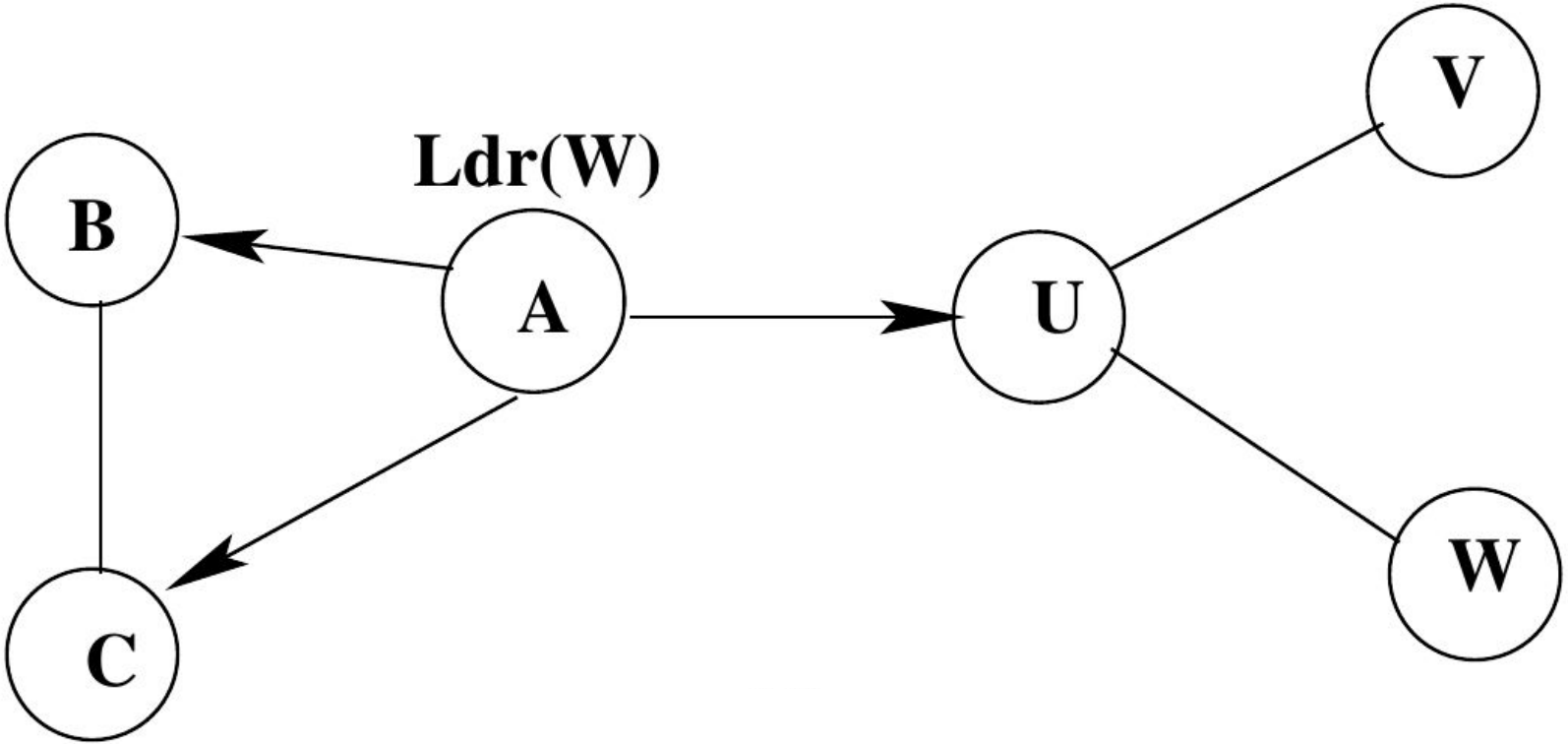
Example



Example



Example



Conclusions

- The bully algorithm
- Election in an ad hoc wireless network

References

Chapter 6.4 of “Distributed Systems by M. van Steen and A. S. Tanenbaum”