Randomized Self-Stabilizing Algorithms for Wireless Sensor Networks

Volker Turau and Christoph Weyer

Institute of Telematics
Hamburg University of Technology

International Workshop on Self-Organizing Systems 2006

TUHH
Technische Universität Hamburg-Harburg

Outline

Wireless Sensor Networks

Definition (WSN)

Wireless sensor networks are networks of many small, battery-powered, resource-constrained devices equipped with a CPU, sensors and transceivers embedded in a physical environment where they operate unattendedly

Challenges:

- Resource limitations
- High failure rates
- Ad hoc deployment
- Unreliable communication links

Faul Tolerance in Wireless Sensor Networks

- WSNs experience node/link failures, changing environmental conditions, nodes lose synchrony, programs reach arbitrary states
- Traditional approaches
  - masking where effects of faults are shielded
  - shutdown and globally reset of complete network

  are not feasible

Problem

What fault tolerance mechanisms are suitable for WSNs?

Faul Tolerance in Wireless Sensor Networks

- WSNs experience node/link failures, changing environmental conditions, nodes lose synchrony, programs reach arbitrary states
- Traditional approaches
  - masking where effects of faults are shielded
  - shutdown and globally reset of complete network

  are not feasible

Problem

What fault tolerance mechanisms are suitable for WSNs?

Non-Masking Fault Tolerance

Definition (Dijkstra)

We call the system **self-stabilizing** if and only if regardless of the initial state [...], the system is guaranteed to find itself in a legitimate state after a finite number of moves

- Objective: recovery from transient faults in bounded time without any external intervention

Key Principle

Instead of modeling individual errors the error free state is modeled

- Error free state is defined by a **predicate** $\mathcal{P}$ defined **locally**, i. e., based on local state of each node and states of neighboring nodes

Definitions

Definition

- The **topology** of a network N consisting of n nodes is repr. by undirected graph $G = (N, E)$, E set of bidirectional communication links
- The **state** s_i of node i is described by its local variables
- Tuple of local states $(s_1, s_2, \ldots, s_n)$ is called **configuration** of N
- Σ denotes the set of all configurations
- A configuration $\sigma \in \Sigma$ is called **legitimate** if it satisfies $\mathcal{P}$ (free of faults)

Definitions

Definition

- A **system** is a pair $(\Sigma, \rightarrow)$, where $\rightarrow: \Sigma \times \Sigma$ is a transition relation
- A **transition** is caused by the execution of a **program** on a node
- An **execution** is a maximal sequence $c_0, c_1, c_2, \ldots$ of configurations such that $c_0 \in \Sigma$ and $c_i \rightarrow c_{i+1}$ for each $i \geq 0$
- A configuration $\sigma \in \Sigma$ is **reachable** from a configuration $c \in \Sigma$, if there exists an execution starting in c and passing through σ

Main Definition

Definition (Self-Stabilization)

- Let $\mathcal{L} \subseteq \Sigma$ be the set of all legitimate configurations relative to $\mathcal{P}$. A system $(\Sigma, \rightarrow)$ is **self-stabilizing** with respect to $\mathcal{P}$ if the following properties hold:
  1. If $c \in \mathcal{L}$ and $c \rightarrow c'$ then $c' \in \mathcal{L}$ (**closure property**)
  2. Starting from any configuration $c \in \Sigma$ every execution reaches $\mathcal{L}$ within a finite number of transitions (**convergence property**)

Programs

Definition

- A program P consist of r **rules** of the following kind:

$$guard_i \longrightarrow statement_i$$

- **Guards**: Boolean expressions based on local view of node: state of node and states of neighbors only
- **Statements**: Only change the local state (based on local view)
- **Move**: Execution of a rule by a node
- A node is called **enabled** if guard of at least one of its rules is satisfied

Schedulers

Definition

- **Scheduler**: Controls interleaved execution of enabled nodes
- **Schedule**: Sequence $S_1, S_2, \ldots$ of subsets of enabled nodes, **rounds**
- **Central daemon scheduler**: $|S_i| = 1$ for all i
- **Distributed daemon scheduler**: S_i is any subset of enabled nodes all i
- **Fully distributed daemon scheduler**: $|S_i|$ includes all enabled nodes

Example: Maximal independent sets

Example

- *Independent set I*: subset of nodes s.t. no two nodes in I are neighbors
- State of node described by boolean variable *in* (**true** $\hookrightarrow$ node in MIS)
- Rules:

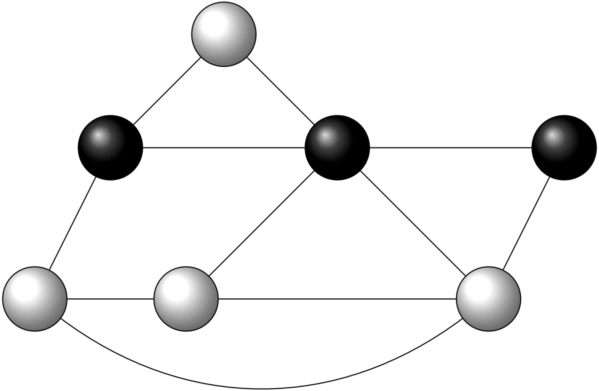
$$\textbf{if } (in = \textbf{false} \wedge \forall \textit{ neighbors } v : (v.in = \textbf{false})) \longrightarrow in := \textbf{true}$$
$$\textbf{if } (in = \textbf{true} \wedge \exists \textit{ neighbor } v : (v.in = \textbf{true})) \longrightarrow in := \textbf{false}$$

Theorem (Hedetniemi et al.)

Algorithm finds MIS in at most 2n moves using central daemon scheduler.

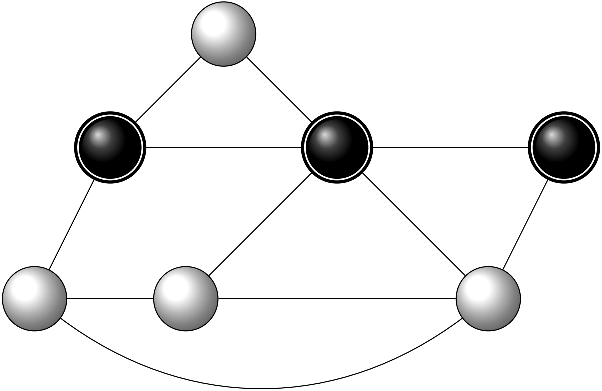
Example: MIS using **central** resp. **fully distributed** daemon scheduler

Maximal Independent Sets (Central daemon)



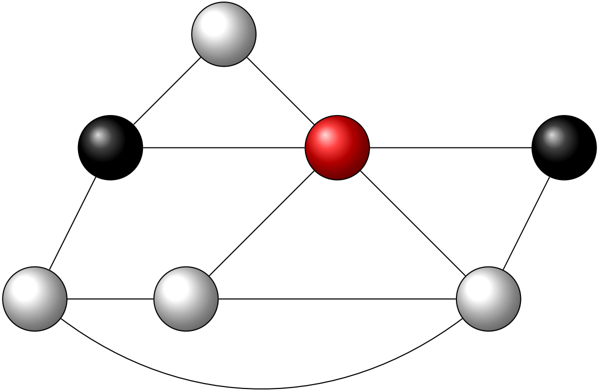
Random initialization

Maximal Independent Sets (Central daemon)



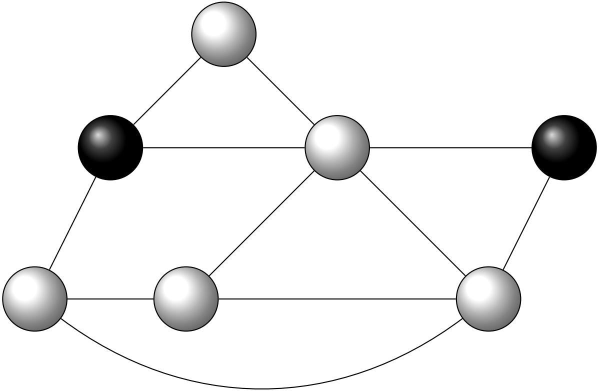
3 nodes are enabled

Maximal Independent Sets (Central daemon)



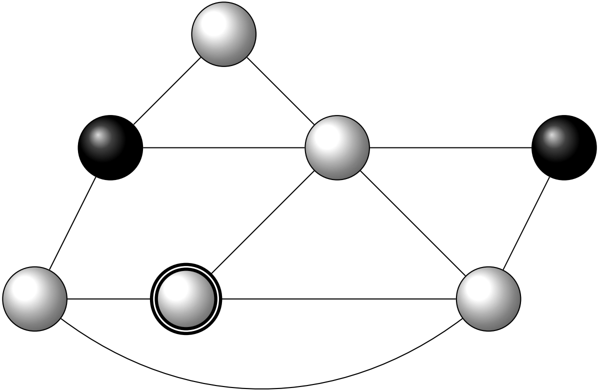
Central daemon selects a node

Maximal Independent Sets (Central daemon)



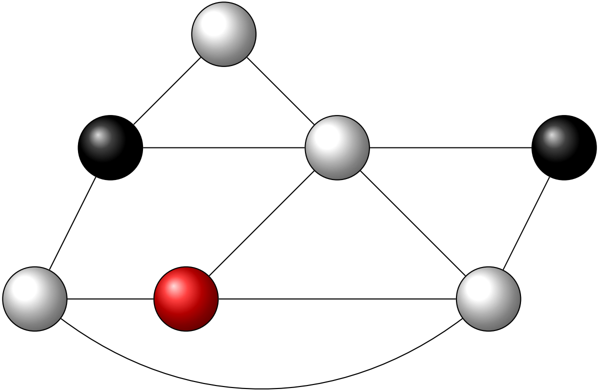
Node executes

Maximal Independent Sets (Central daemon)



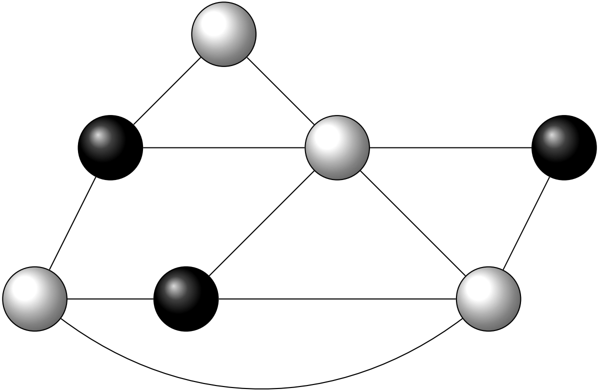
1 node is enabled

Maximal Independent Sets (Central daemon)



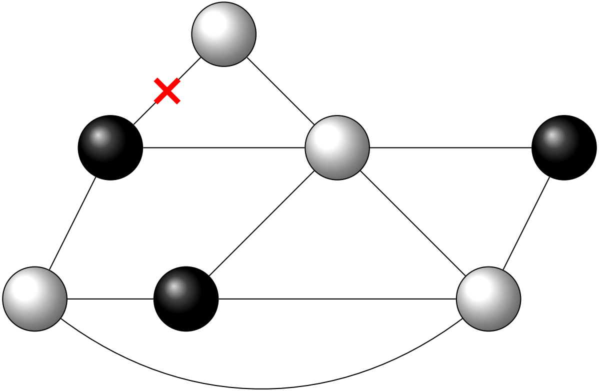
Central daemon selects a node

Maximal Independent Sets (Central daemon)



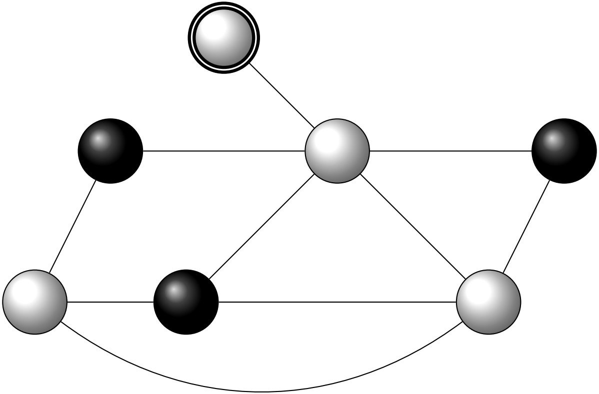
Node executes, stabilization

Maximal Independent Sets (Central daemon)



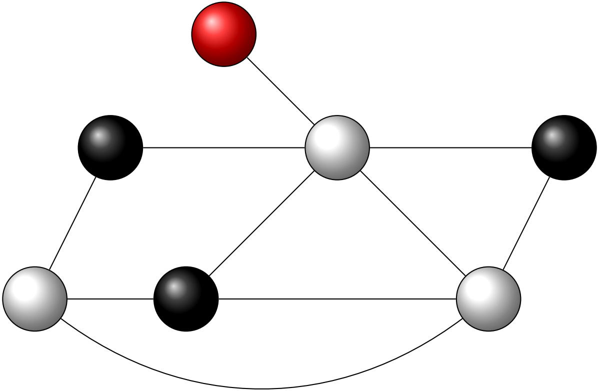
Link fails

Maximal Independent Sets (Central daemon)



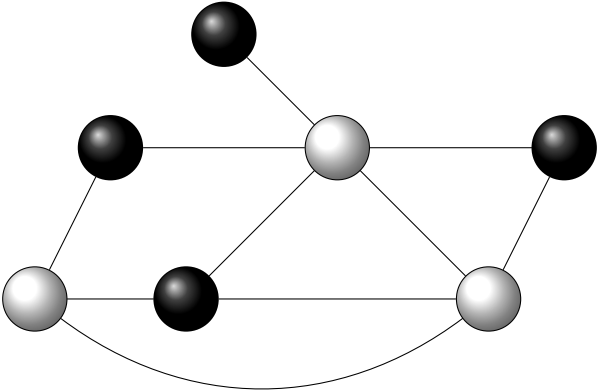
Node gets enabled

Maximal Independent Sets (Central daemon)



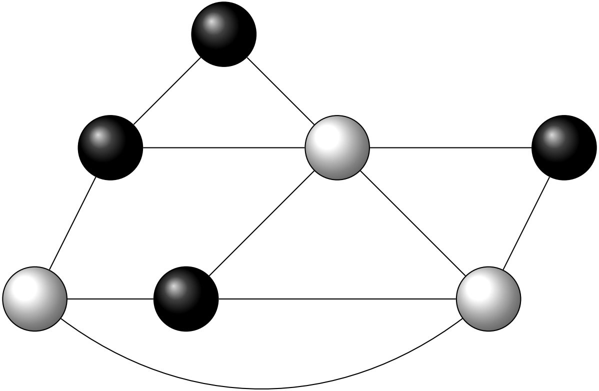
Central daemon selects a node

Maximal Independent Sets (Central daemon)



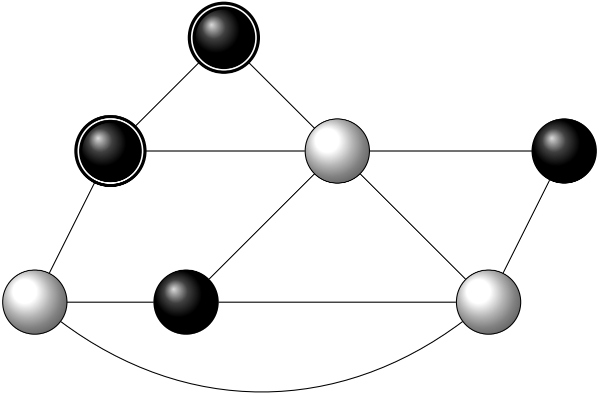
Node executes, stabilization

Maximal Independent Sets (Central daemon)



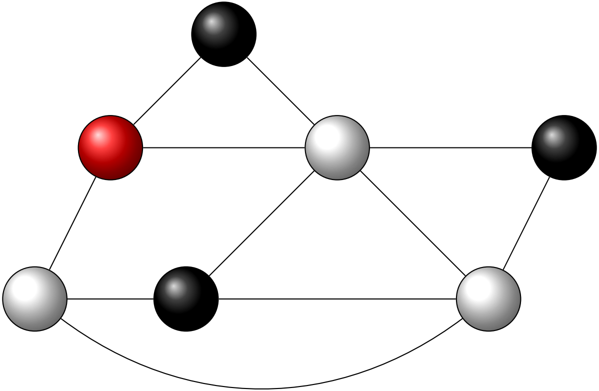
Link becomes available again

Maximal Independent Sets (Central daemon)



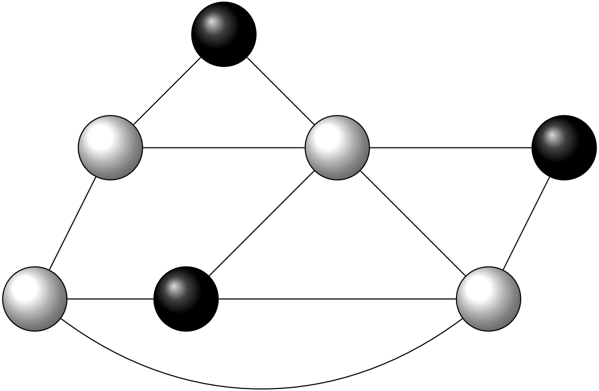
2 nodes gets enabled

Maximal Independent Sets (Central daemon)



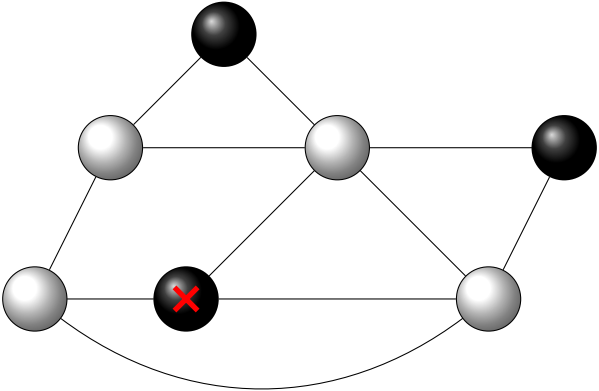
Central daemon selects a node

Maximal Independent Sets (Central daemon)



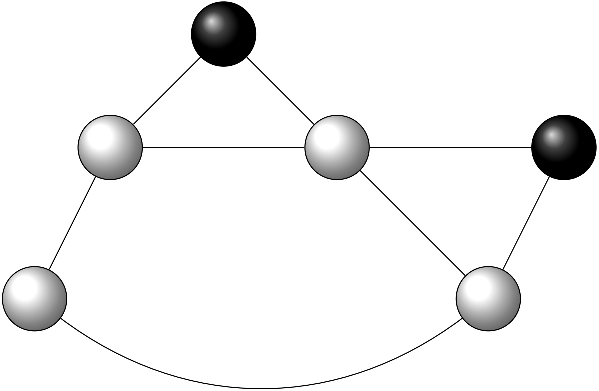
Node executes, stabilization

Maximal Independent Sets (Central daemon)



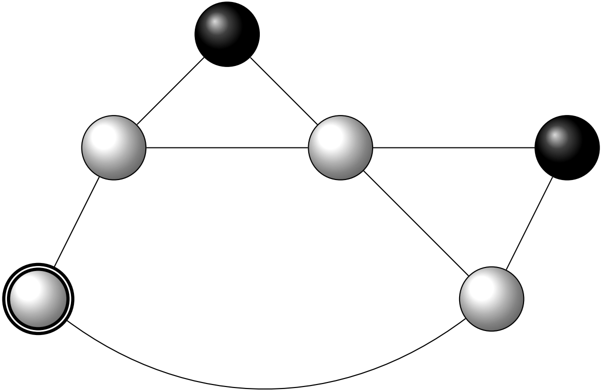
Node fails

Maximal Independent Sets (Central daemon)



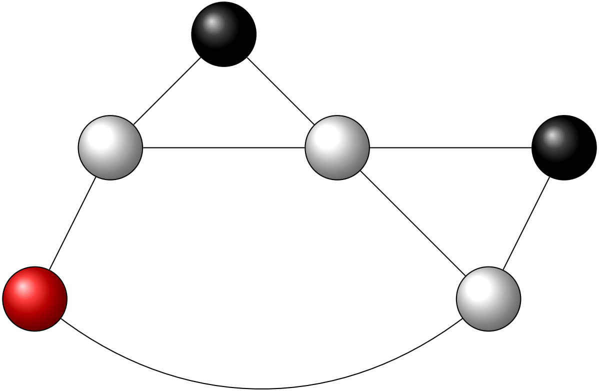
Links disappear

Maximal Independent Sets (Central daemon)



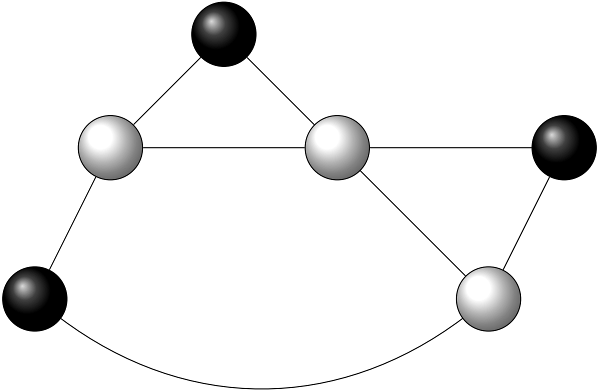
Node gets enabled

Maximal Independent Sets (Central daemon)



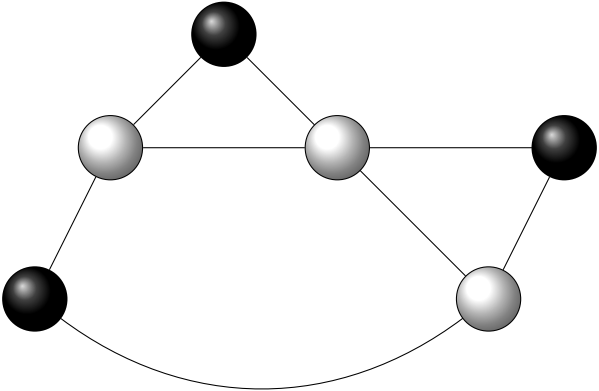
Central daemon selects a node

Maximal Independent Sets (Central daemon)



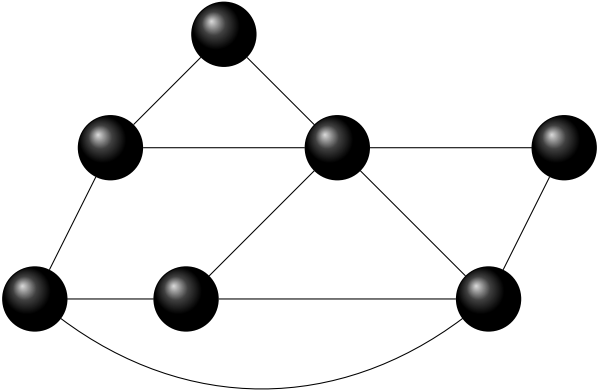
Node executes, stabilization

Maximal Independent Sets (Central daemon)



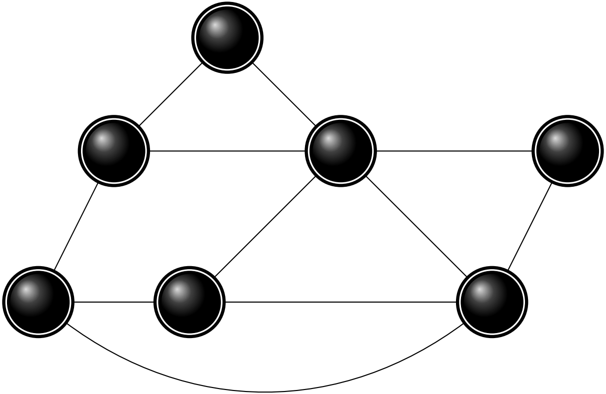
MIS with fully distributed daemon scheduler

Maximal Independent Sets (Fully distributed daemon)



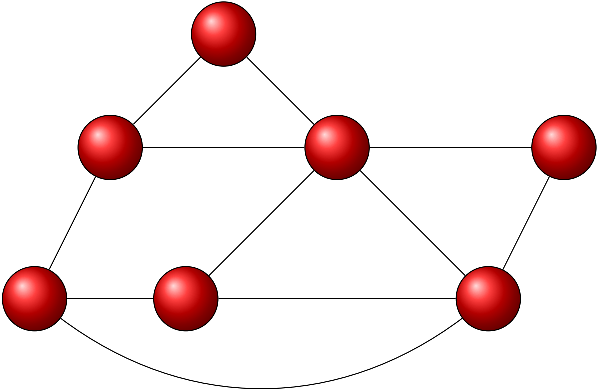
Random initialization

Maximal Independent Sets (Fully distributed daemon)



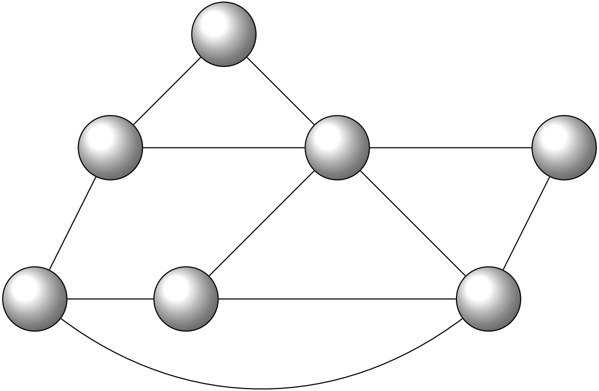
All nodes are enabled

Maximal Independent Sets (Fully distributed daemon)



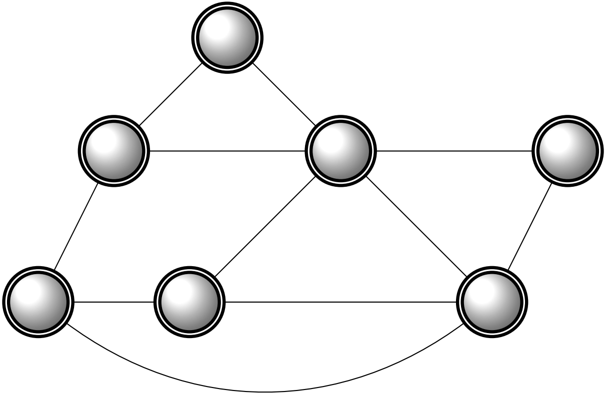
Daemon selects all nodes

Maximal Independent Sets (Fully distributed daemon)



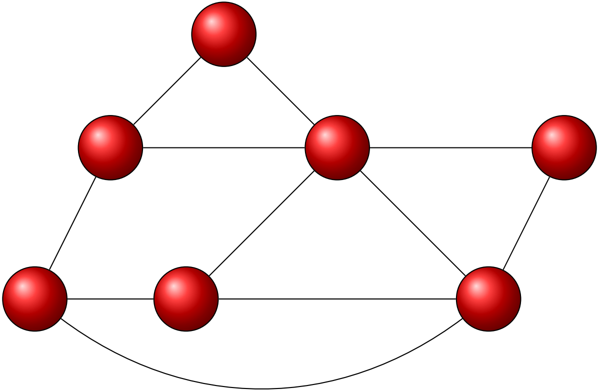
All nodes execute

Maximal Independent Sets (Fully distributed daemon)



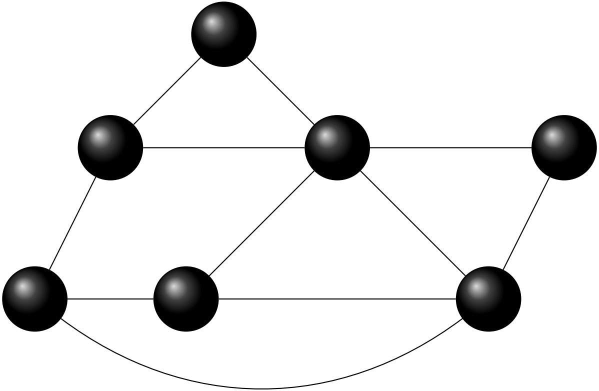
All nodes are enabled

Maximal Independent Sets (Fully distributed daemon)



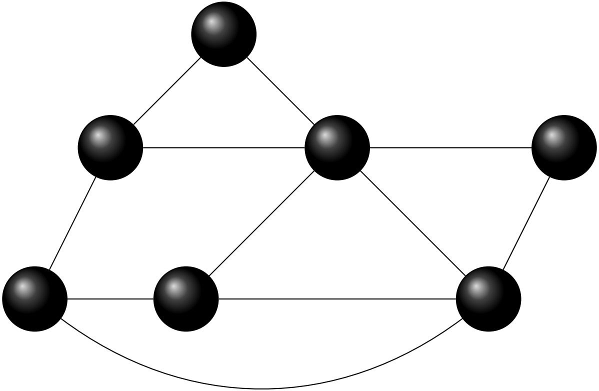
Daemon selects all nodes

Maximal Independent Sets (Fully distributed daemon)



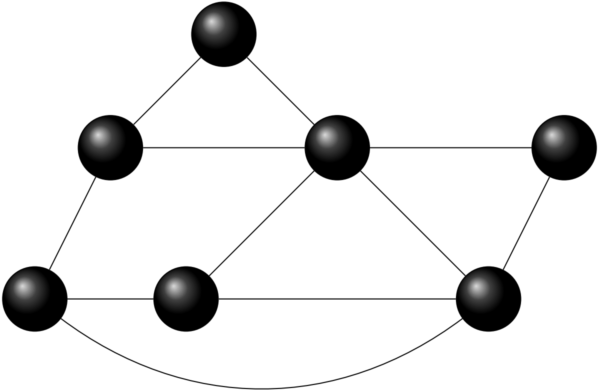
All nodes execute

Maximal Independent Sets (Fully distributed daemon)



No stabilization!

Maximal Independent Sets (Fully distributed daemon)



Back to main text

Expected Advantages for WSNs

- No need to initialize nodes in consistent manner
- In-network update of software
- Eventual consistent node recovery after failure (temporary power outage, memory crash)
- Handling errors in transmissions, e.g. data corruption

Introducing Self-Stabilization for WSNs

- Majority of work on self-stabilization is based on models not suitable for the constraints of WSNs:
  - central daemon scheduler
  - atomicity
  - shared memory model
  - unique processor identifiers
  - xed topology

Problem

Which models allow to use self-stabilization for WSNs?

Introducing Self-Stabilization for WSNs

- Majority of work on self-stabilization is based on models not suitable for the constraints of WSNs:
  - central daemon scheduler
  - atomicity
  - shared memory model
  - unique processor identifiers
  - xed topology

Problem

Which models allow to use self-stabilization for WSNs?

Central Daemon Scheduler

- Problem: Implementation in a WSN
- Distributed implementation of central daemon
  - self-stabilizing mutual exclusion or token passing algorithm
  - requires globally unique ids or semi-uniform network
  - disadvantage: not silent, high communication load

$\hookrightarrow$ Fully distributed scheduler

- Need for symmetry breaking mechanism
  - globally unique ids
  - randomization
- CSMA/CA: random back offs (sufficient?)

Communication Style

- Common models: shared memory and message passing not suitable
  - broadcast is main communication primitive
  - sending and receiving are mutual exclusive

$\hookrightarrow$ Cached Sensornet Transformation - CST (Herman)

- Each node maintains cache with the states of all neighbors
- Atomically, when node changes its state it also broadcasts new state
- Neighbors update their cache upon receiving message
- Cache coherence: entries in cache are fresh

Topology

- Topology emerges after deployment
- Topology has to be dynamically determined by neighborhood protocol
- Neighborhood protocols: balance between agility and stability

$\hookrightarrow$ Options:

- Either topology changes are tolerated by algorithm or
- Time between changes must be sufficiently long to reach a legitimate state

Problem

Problem

How can we transform algorithms that stabilze under central daemon into algorithms that stabilize under fully distributed scheduler?

Probabilistically Self-Stabilization

Definition (Probabilistically Self-Stabilization)

A system $(\Sigma, \rightarrow)$ is **probabilistically self-stabilizing** with respect to $\mathcal{P}$ if

1. The closure property as defined above holds
2. There exists function $f : \mathbb{N} \rightarrow [0, 1]$ with $\lim_{k \to \infty} f(k) = 0$, s.t. the probability of reaching a legitimate configuration, starting from any configuration within k transitions, is $1 - f(k)$ (**probabilistic convergence property**)

Transformation of self-stabilizing algorithms

- Fully distributed daemon scheduler: Nodes have common understanding of time and execution times of statements are bounded

Definition (Algorithm $\mathcal{A}^{\mathcal{CR}}$)

Let $\mathcal{A}$ be a self-stabilizing algorithm

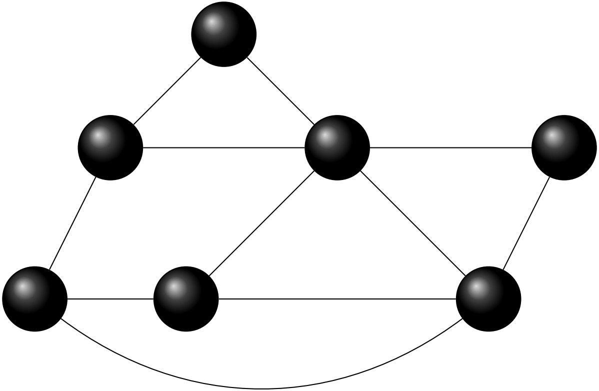
- Apply cached sensornet transformation
- Transform each rule *guard* $\longrightarrow$ *statement* into

  guard $\longrightarrow$ **if** $(\text{rand}() < p)$ **then** *statement*

  with fixed $p \in (0, 1)$

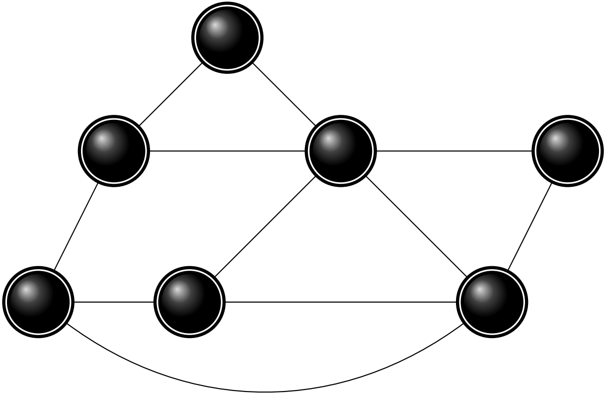
- Demo: (**MIS based on $\mathcal{A}^{\mathcal{CR}}$**)

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



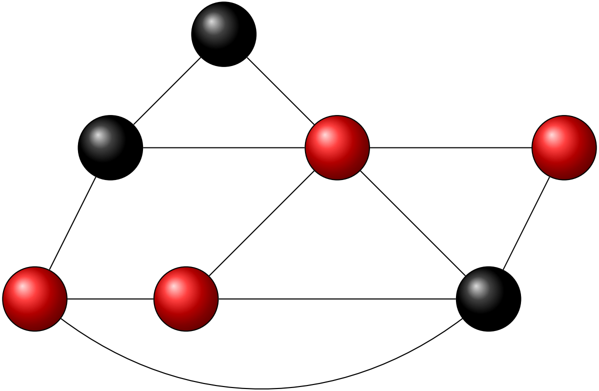
Random initialization

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



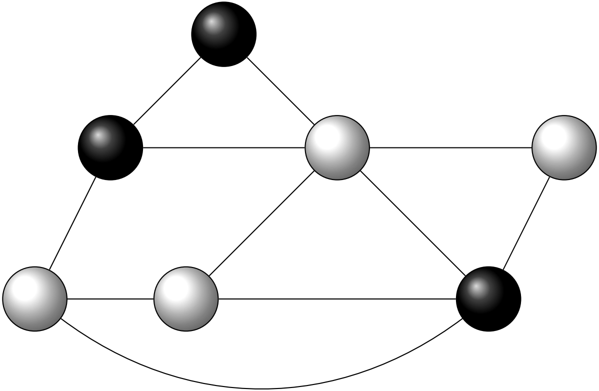
All nodes are enabled

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



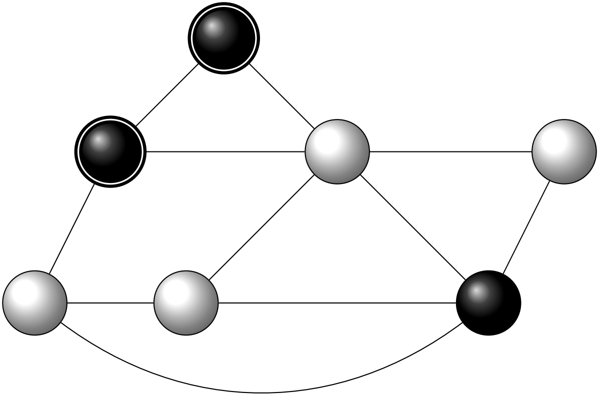
Daemon selects four nodes

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



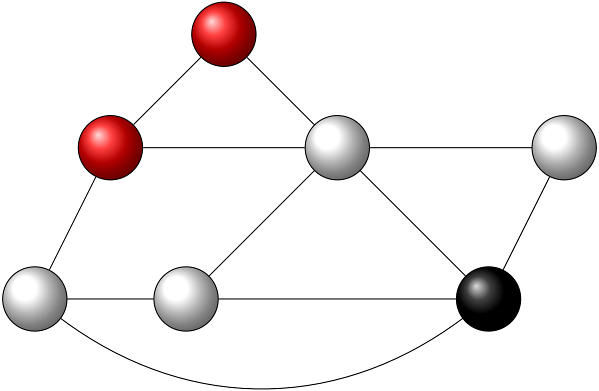
Four nodes execute

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



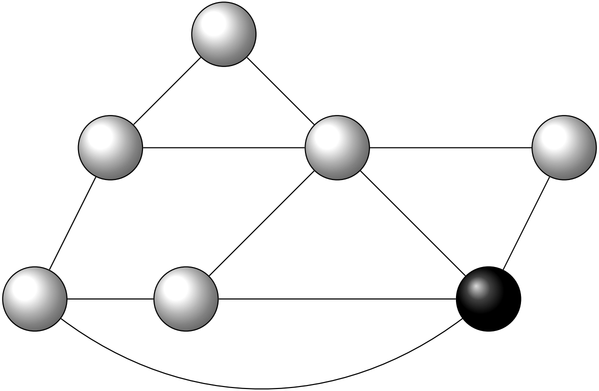
Two nodes are enabled

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



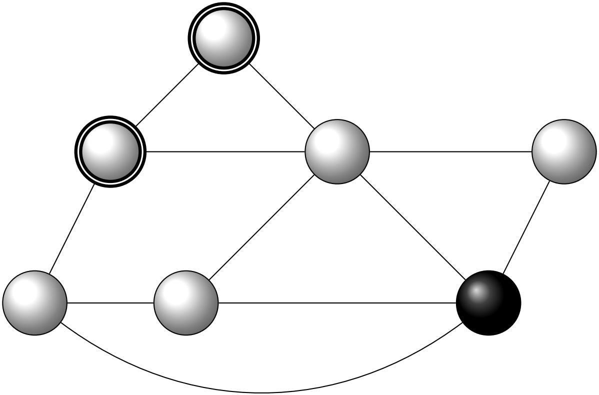
Daemon selects both nodes

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets

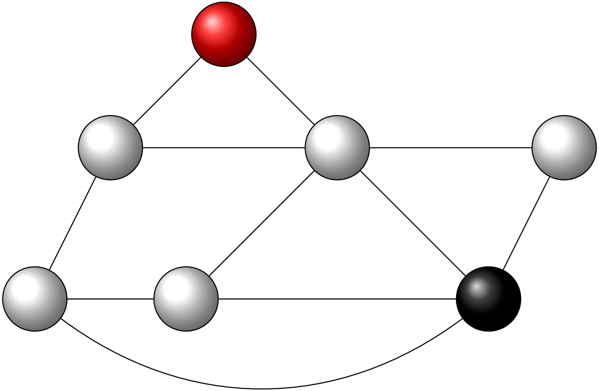


Two nodes execute

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets

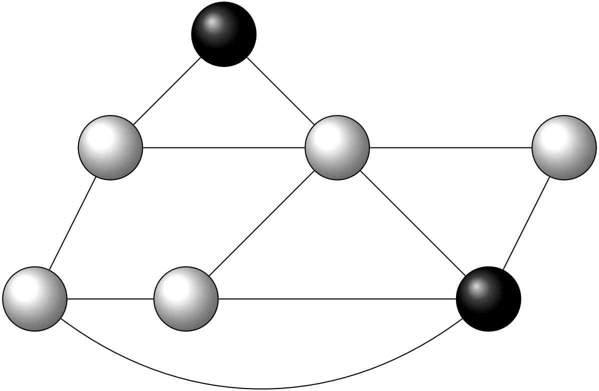


Two nodes are enabled



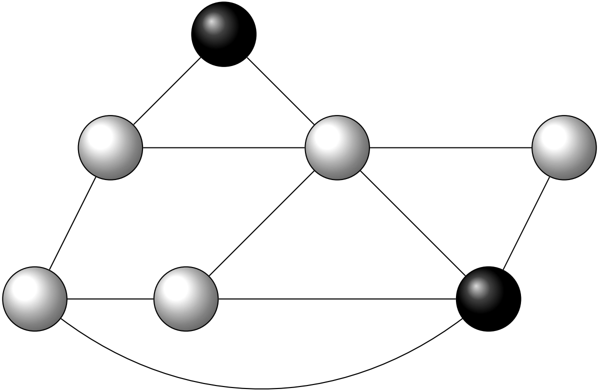
Daemon selects one node

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



Stabilization

Algorithm $\mathcal{A}^{\mathcal{CR}}$ for Maximal Independent Sets



Back to main text

Algorithm $\mathcal{A}^{\mathcal{CR}}$

Theorem

Let $\mathcal{A}$ be a self-stabilizing algorithm that stabilizes under central daemon scheduler after finite number of moves with respect to predicate $\mathcal{P}$. If

1. *initial configuration is cache coherent, and*
2. *all broadcasts are reliable*

then algorithm $\mathcal{A}^{\mathcal{CR}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under distributed daemon scheduler.

- Both assumptions are necessary

Unreliable Communication

- Assumption: All transmissions are independent and succeed with fixed probability
- Stabilization not guaranteed: Algorithm may reach non-cache coherent configuration in which no node is enabled
- Solution: Nodes broadcast their states to neighbors periodically at beginning of every round
- Call this algorithm $\mathcal{A}^{\mathcal{CRP}}$

Algorithm $\mathcal{A}^{\mathcal{CRP}}$

Theorem

Let $\mathcal{A}$ be as before. Assume that the probability that a message is successfully transmitted is fixed and that these events are independent. Then algorithm $\mathcal{A}^{\mathcal{CRP}}$ is probabilistic self-stabilizing with respect to $\mathcal{P}$ under the distributed daemon scheduler.

- Initial configuration does not need to be cache coherent
- Loss of a message is not a transient fault
- Messages may be lost during final interval

Reducing Communication

- Broadcasting the state in every round causes two problems
  - Energy consumption is increased
  - Likelihood of collisions is increased (slower stabilization)
- Solutions:
  - Nodes broadcast state after random waiting period
  - Nodes do not broadcast state in each round, but randomly skip rounds
  - This algorithm is probabilistic self-stabilizing

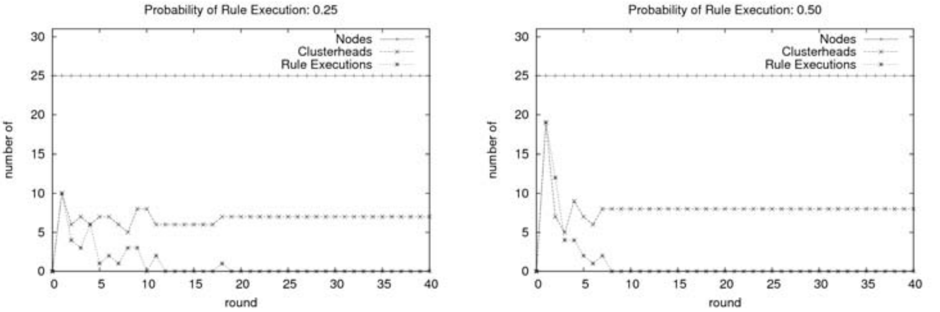
Periodic Broadcasting with implicit Acknowledgments

- Observation: Once neighbors know current state of node, node can suspend broadcasting until next change of state
- Idea: Nodes include in broadcasts latest received states of all neighbors
- If all neighbors know current state, node can stop broadcasting
- This algorithm is probabilistic self-stabilizing
  - Disadvantage: Increased packet size leading to more collisions
  - Advantage: After system reached legitimate state, no broadcast messages are needed until next transient fault

Algorithm $\mathcal{A}^{\mathcal{CRP}}$ in Real Experiments

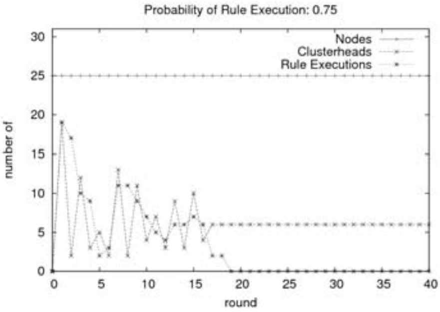
- Experiment with a real WSN: 25 nodes of type ESB
- Lowest layer of implementation is a synchronization protocol to force nodes to operate in rounds
- Nodes randomly select instant during each round to broadcast state

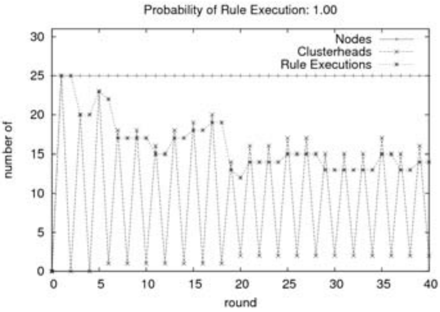
Algorithm $\mathcal{A}^{\mathcal{CRP}}$ in Real Experiments



Execution of algorithm $\mathcal{A}^{\mathcal{CRP}}$ during first 40 rounds

Algorithm $\mathcal{A}^{\mathcal{CRP}}$ in Real Experiments





Execution of algorithm $\mathcal{A}^{\mathcal{CRP}}$ during first 40 rounds

Limitations of Self-Stabilization

- Nodes do not know when system is stable
- Duration of unavailability is unknown
- System experiences effect of transient faults and must be prepared to tolerate these situations

Conclusions

- WSNs need fault-tolerance mechanisms
- Model for Self-stabilizations in WSNs
- Transformation of SS-algorithms under central daemon into WSNs
- Real implementation

Randomized Self-Stabilizing Algorithms for Wireless Sensor Networks

Volker Turau and Christoph Weyer

Institute of Telematics
Hamburg University of Technology

International Workshop on Self-Organizing Systems 2006

TUHH
Technische Universität Hamburg-Harburg