Programming Wireless Sensor Networks in a Self-Stabilizing Style

C. Weyer\textsuperscript{1} V. Turau\textsuperscript{1} A. Lagemann\textsuperscript{2} J. Nolte\textsuperscript{2}

\textsuperscript{1}Institute of Telematics
Hamburg University of Technology

\textsuperscript{2}Distributed Systems/Operating Systems
Brandenburg University of Technology Cottbus

The Third International Conference on Sensor Technologies and Applications, 2009
Outline

Motivation

Short Introduction to Self-Stabilization

Self-Stabilization in WSN

Main Results
Self-Stabilization

Definition (Dijkstra 1974)

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.
Fault Model

- Transient faults
- Caused by environmental influences
  - Wireless channel characteristics
  - Cosmic rays
  - ...
- Lasting effect on state of the network
  - Message loss or corruption
  - Reset of nodes
  - Corruption of memory

Motivation
Motivation

Fault Model

- Transient faults
- Caused by environmental influences
  - Wireless channel characteristics
  - Cosmic rays
  - ...
- Lasting effect on state of the network
  - Message loss or corruption
  - Reset of nodes
  - Corruption of memory
- Other faults like:
  - Discharged nodes
  - Broken links
- Can be modeled as transient faults
Motivation

Benefits of Self-Stabilization

- Inherent non-masking fault tolerance
- Formally verifiable
- Proofs are based on simple model
- Transformation to realistic model possible
- While preserving self-stabilization property
Maximal Independent Set

Example (Maximal Independent Set)

```csharp
public bool in;

rule R1:
in = false and forall(Neighbors v : v.in = false)
    -> in := true;

rule R2:
in = true and exists(Neighbors v : v.in = true) ->
in := false;
```
Maximal Independent Set

Example (Maximal Independent Set)

```csharp
public bool in;

rule R1:
    in = false and forall(Neighbors v : v.in = false)
    -> in := true;

rule R2:
    in = true and exists(Neighbors v : v.in = true)
    -> in := false;
```
Example (Maximal Independent Set)

```java
public bool in;

rule R1:
    in = false and forall(Neighbors v : v.in = false) -> in := true;

rule R2:
    in = true and exists(Neighbors v : v.in = true) ->
    in := false;
```
Example (Dolev 2000)

```java
public map NodeID Platform.ID as ID;
public int dist;
public NodeID parent;

declare int minD := min(v.dist | Neighbors v);

rule R1:
  ID = 0 and !(parent = null and dist = 0) ->
      parent := null;
      dist := 0;

rule R2:
  ID != 0 and !(parent in (v.ID | Neighbors v : v.dist = minD)
      and dist = minD + 1) ->
      parent := choose(v.ID | Neighbors v : v.dist = minD);
      dist := minD + 1;
```
public map int Neighborhood.numOfNeigh as d;
public int c;

declare set int colors := (1:d);

declare bool
  B1 := c in (v.c|Neighbors v) or c>d+1;

declare bool
  B2 := colors = (v.c|Neighbors v);

rule R1:
  B1 and B2 ->
  c := d + 1;

rule R2:
  B1 and !B2 ->
  c := choose(colors \ (v.c|Neighbors v));
• Central entity (called daemon) assumed
• Algorithm execution is divided into rounds
• Daemon selects exactly one node
• Selection is fair
• Basically a serialization
Execution Model – Central Daemon Scheduler

- Central entity (called daemon) assumed
- Algorithm execution is divided into rounds
- Daemon selects exactly one node
- Selection is fair
- Basically a serialization
Central entity (called daemon) assumed
Algorithm execution is divided into rounds
Daemon selects exactly one node
Selection is fair
Basically a serialization
Central entity (called daemon) assumed
Algorithm execution is divided into rounds
Daemon selects exactly one node
Selection is fair
Basically a serialization
Communication between neighbors (Cluster-Tree Communication)

Above the cluster level, trees are used to communicate. Nodes on the same level communicate with the root of the tree. This allows for efficient and scalable communication in the network.

A. Lagemann  Self-Stabilizing WSN
Algorithm execution is divided into rounds

Every enabled node is automatically activated

Is not equivalent to central daemon model!
Execution Model – Synchronous

- Algorithm execution is divided into rounds
- Every **enabled** node is automatically **activated**
- Is **not** equivalent to central daemon model!
• Algorithm execution is divided into rounds
• Every *enabled* node is automatically *activated*
• Is *not* equivalent to central daemon model!
Execution Model – Synchronous

- Algorithm execution is divided into rounds
- Every **enabled** node is automatically **activated**
- Is **not** equivalent to central daemon model!
Algorithm execution is divided into rounds
• Every enabled node is automatically activated
• Is not equivalent to central daemon model!
• Algorithm execution is divided into rounds
• Every **enabled** node is automatically **activated**
• Is **not** equivalent to central daemon model!
Algorithm execution is divided into rounds
- Every enabled node is automatically activated
- Is not equivalent to central daemon model!
Transformations for WSN

- Communication Model
  - Cached Sensornet Transformation (Herman 2003)
- Execution Model
  - Strict transformations
    - Deterministic conflict manager (Gradinariu, Tixeuil 2007)
    - BitToss (Goddard, Hedetniemi, Jacobs, Srimani 2008)
  - Weak transformations
    - Randomized conflict manager (Gradinariu, Tixeuil 2007)
    - Randomized transformation (Turau, Weyer 2006)
Transformations for WSN

- Communication Model
  - Cached Sensornet Transformation (Herman 2003)

- Execution Model
  - **Strict transformations**
    - Deterministic conflict manager (Gradinariu, Tixeuil 2007)
    - BitToss (Goddard, Hedetniemi, Jacobs, Srimani 2008)
  - **Weak transformations**
    - Randomized conflict manager (Gradinariu, Tixeuil 2007)
    - Randomized transformation (Turau, Weyer 2006)
Transformations for WSN

- Communication Model
  - Cached Sensornet Transformation (Herman 2003)

- Execution Model
  - **Strict transformations**
    - Deterministic conflict manager (Gradinariu, Tixeuil 2007)
    - BitToss (Goddard, Hedetniemi, Jacobs, Srimani 2008)
  - **Weak transformations**
    - Randomized conflict manager (Gradinariu, Tixeuil 2007)
    - Randomized transformation (Turau, Weyer 2006)
Transformations for WSN

- Communication Model
  - Cached Sensornet Transformation (Herman 2003)

- Execution Model
  - Strict transformations
    - Deterministic conflict manager (Gradinariu, Tixeuil 2007)
    - BitToss (Goddard, Hedetniemi, Jacobs, Srimani 2008)
  - Weak transformations
    - Randomized conflict manager (Gradinariu, Tixeuil 2007)
    - Randomized transformation (Turau, Weyer 2006)
Transformations for WSN

- Communication Model
  - Cached Sensornet Transformation (Herman 2003)

- Execution Model
  - Strict transformations
    - Deterministic conflict manager (Gradinariu, Tixeuil 2007)
    - BitToss (Goddard, Hedetniemi, Jacobs, Srimani 2008)
  - Weak transformations
    - Randomized conflict manager (Gradinariu, Tixeuil 2007)
    - Randomized transformation (Turau, Weyer 2006)
Properties of Transformations

- **Strict transformations**
  - **Advantage**: equivalent to central daemon
  - **Drawback**: limited concurrent activity

- **Weak transformations**
  - **Advantage**: allow for more concurrency
  - **Drawback**: only probabilistic convergence
Major Concern: Convergence Time

- Represents responsiveness of algorithms
- High convergence time leads to low availability
- Major Question: Influence of transformations on convergence time?
- Do weak transformations reduce convergence time more than strict ones?
Upper Bounds vs. Average

- Determining convergence time analytically yields upper bounds
- Analytical determination of average is prohibitive $\Rightarrow$ large value space
- Only practical method: simulation
- **Contribution**: analysis of convergence time of three algorithms central to WSN applications
Maximal Independent Set (Density 9)

- BitToss
- CMR
- CMD
- Random

Number of nodes vs. convergence time (rounds)

- 100
- 200
- 300
- 400
- 500
- 600
- 700
- 800
- 900
- 1000
Vertex Coloring (Density 9)
Spanning Tree (Density 9)
Main Results Simulations with 400 Nodes

Maximal Independent Set (400 Nodes)

![Graph showing convergence time vs. density for different protocols: BitToss, CMR, CMD, and Random.]

- **BitToss**
- **CMR**
- **CMD**
- **Random**

**x-axis:** Density
**y-axis:** Convergence time (rounds)
Vertex Coloring (400 Nodes)

The graph shows the convergence time in rounds for different node densities and several algorithms:
- BitToss
- CMR
- CMD
- Random

The x-axis represents node density, while the y-axis represents the convergence time in rounds.
Main Results  Simulations with 400 Nodes

Spanning Tree (400 Nodes)

convergence time (rounds)
node density
Random
CMD
CMR
BitToss
Synchronous

A. Lagemann  Self-Stabilizing WSN
Main Results  
Dependency of Node Density

Vertex Coloring (400 Nodes, Synchronous Version)

- Random (p=0.20)
- Random (p=0.35)
- Random (p=0.50)
- Random (p=0.65)
- Random (p=0.80)
- Synchronous (2d)
- Synchronous ($d^2$)

convergence time (rounds)
node density
Summary

- average convergence time much better than upper bounds from literature
- randomized transformation very good performance
- with randomized transformation convergence time only depends on convergence time of original algorithm

Outlook

- Use SelfWISE on real sensor hardware (e.g. TMoteSky or SunSpot).
- Determine the duration of a round under real conditions.
Thank You!

Questions ?
Appendix For Discussion

Basic Definitions

- The state of node is described by its variables
- Configuration \( c \) of network is tuple of node states
- Each node has strict local view upon network
  - Node can read/write own state
  - Node can read state of neighbors
- Absence of faults is defined by a predicate \( P \)
- A configuration is legitimate if it satisfies \( P \)
- A transition \( c \to c' \) is caused by executing an algorithm
- An algorithm consists of rules of the following kind
  
  \[ \text{guard}_1 \longrightarrow \text{statement}_1 \]
  
  \[ \text{guard}_2 \longrightarrow \text{statement}_2 \]
  
  \[ \ldots \]
Main Definition

Definition (Self-Stabilization)

Let $\mathcal{L}$ be the set of all legitimate configurations relative to a predicate $\mathcal{P}$. A system is self-stabilizing with respect to $\mathcal{P}$ if:

1. If $c \in \mathcal{L}$ and $c \rightarrow c'$ then $c' \in \mathcal{L}$ (closure property)
2. Starting from any configuration every execution reaches $\mathcal{L}$ within a finite number of transitions (convergence property)
SelfWISE

A. Lagemann  Self-Stabilizing WSN  26