Self-stabilization and Sensor Networks

Sébastien Tixeuil

LRI - CNRS 8623 & INRIA Grand Large
Université Paris Sud, France
tixeuil@lri.fr

December 4, 2008

Outline

Sensor Networks

- processor + sensors + radio
- 2 AA batteries, on/off switch
- 3 LEDs for debugging

Sensor Networks

While (batteries supply power)

- Collect, aggregate and reduce data
- log into memory

In spite of numerous fault modes

- Permanent sensor failures, node failures
- restarts, radio failures
- transient faults, reconfigurations

Distributed Systems

Definition (Classical System, *a.k.a.* Non stabilizing)

Starting from a particular initial configuration, the system immediately exhibits correct behavior.

Definition (Self-stabilizing System)

Starting from any initial configuration, the system eventually reaches a configuration from with its behavior is correct.

Distributed Systems

Definition (Classical System, *a.k.a.* Non stabilizing)

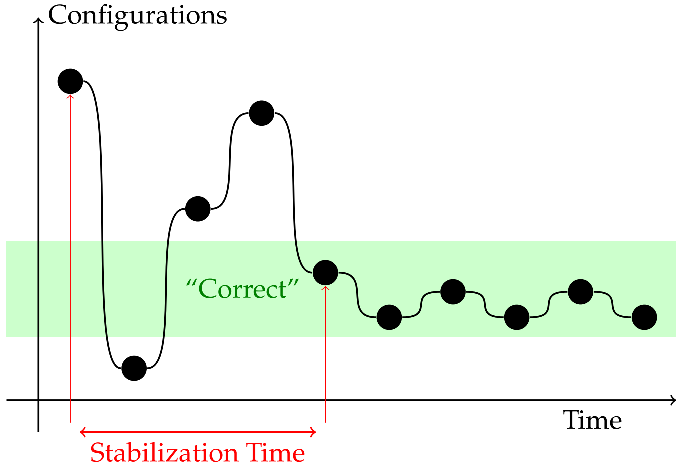
Starting from a particular initial configuration, the system immediately exhibits correct behavior.

Definition (Self-stabilizing System)

Starting from any initial configuration, the system eventually reaches a configuration from with its behavior is correct.

- Self-stabilization permits to recover from transient failures

Self-stabilization



Complexity Criteria

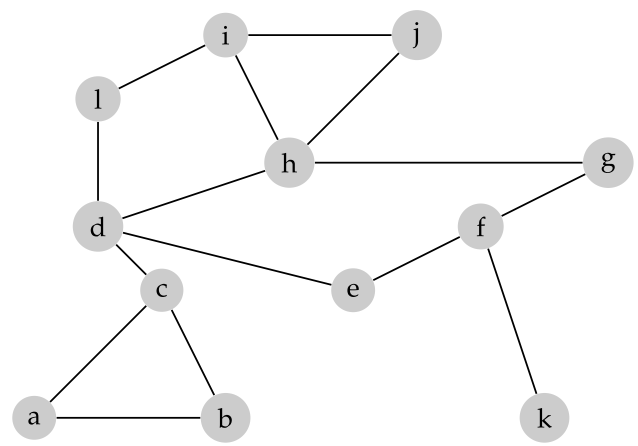
Maximize useful lifetime of system

- "maximise useful": correct quickly from illegitimate state
  - Self-stabilization, scalability
- "maximise lifetime": use minimal energy to preserve batteries
  - local vs. global preserving

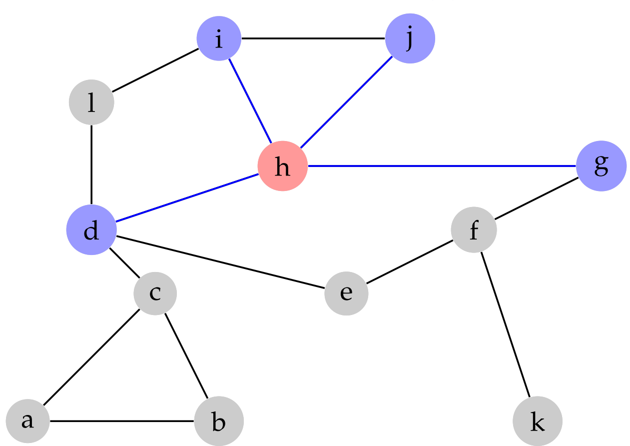
System Specifics

- only one radio frequency
- no collision detect
- access technique: CSMA/CA
- use CRC to detect collision
- no directional send/receive
- msg. are small (30 bytes)
- radio range about 1 meter
- number of neighbors < 10
- could be large number of nodes (perhaps > 100000)
- unique node IDs (probably)
- cost a few $ (someday)
- slow processor (4 MHz)
- limited memory (4 KB RAM)
- item nodes have real-time clocks ≡ drift between 1 msec and 100 msec per second
- several power modes available

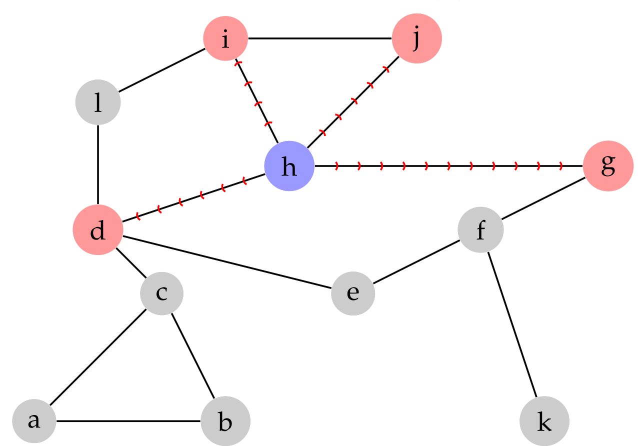
The Model(s)



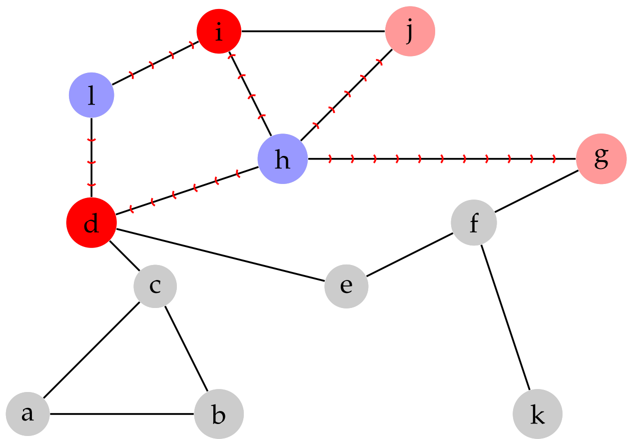
The Model(s)



The Model(s)



The Model(s)



The Model(s)

Self-stabilizing model

- Read neighborhood state,
- compute and update local state

Sensor Network model

- Read local state,
- compute and broadcast to neighborhood
- Collisions may appear

Self-stabilization in Sensor Networks

Transform (i.e. Simulate) the self-stabilizing model into the sensor networks model

- Pros: reuse existing SS algorithms
- Cons: potentially inefficient, overhead

Design self-stabilizing algorithms for the sensor networks model

- Pros: potentially efficient
- Cons: ignore previous SS work

Self-stabilization in Sensor Networks

Transform (i.e. Simulate) the self-stabilizing model into the sensor networks model

- Pros: reuse existing SS algorithms
- Cons: potentially inefficient, overhead
- [Herman 03] Cached Sensornet Transform

Design self-stabilizing algorithms for the sensor networks model

- Pros: potentially efficient
- Cons: ignore previous SS work

Self-stabilization in Sensor Networks

Transform (i.e. Simulate) the self-stabilizing model into the sensor networks model

- Pros: reuse existing SS algorithms
- Cons: potentially inefficient, overhead
- [Herman 03] Cached Sensornet Transform

Design self-stabilizing algorithms for the sensor networks model

- Pros: potentially efficient
- Cons: ignore previous SS work
- [Herman 03] Unison with collisions

Cached Sensornet Transform

Basic Algorithm

- Each node p has a variable v_p
- Each neighbor q of p has a variable $c_q v_p$
  - $c_q v_p$ is the cached value of v_p at q
- Whenever p assigns v_p, p also broadcasts the new value to the neighborhood
- Whenever a neighbor q of p receives v_p, q updates $c_q v_p$ accordingly

Cached Sensornet Transform

Definition (Cache coherence)

For all neighbors p and q, $c_q v_p = v_p$

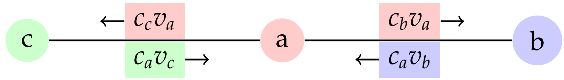
Lemma (Closure)

If started from a cache coherent state, and without collisions, the self-stabilizing model is simulated by replacing all occurrences of $c_q v_p$ by v_p

Example

Lemma (Closure)

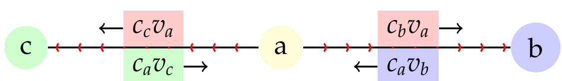
If started from a cache coherent state, and without collisions, the self-stabilizing model is simulated by replacing all occurrences of $c_q v_p$ by v_p



Example

Lemma (Closure)

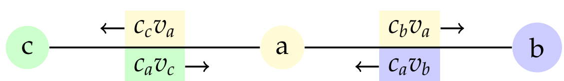
If started from a cache coherent state, and without collisions, the self-stabilizing model is simulated by replacing all occurrences of $c_q v_p$ by v_p



Example

Lemma (Closure)

If started from a cache coherent state, and without collisions, the self-stabilizing model is simulated by replacing all occurrences of $c_q v_p$ by v_p



Cached Sensornet Transform

Periodic retransmit

- Each node p periodically broadcasts v_p to its neighborhood

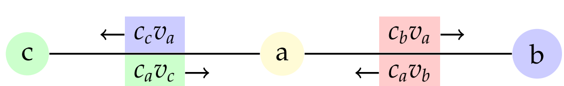
Lemma (Convergence)

If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached

Example

Lemma (Convergence)

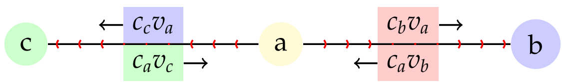
If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached



Example

Lemma (Convergence)

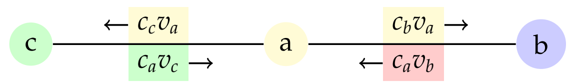
If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached



Example

Lemma (Convergence)

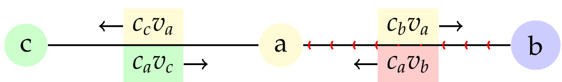
If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached



Example

Lemma (Convergence)

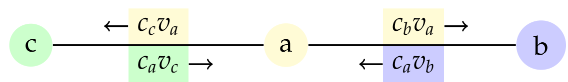
If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached



Example

Lemma (Convergence)

If started from an arbitrary state, and without collisions, a cache coherent state is eventually reached



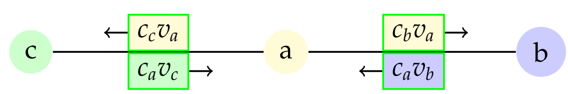
Cached Sensornet Transform

Message Corruption

- Each neighbor q of p has a Boolean variable $b_q v_p$
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \rightarrow A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \rightarrow A$; for all neighbors q of p, $b_p v_q$ becomes false

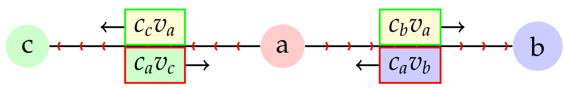
Example

- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \rightarrow A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \rightarrow A$; for all neighbors q of p, $b_p v_q$ becomes false



Example

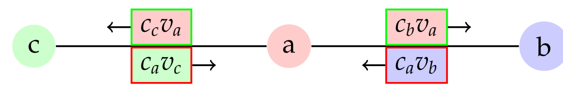
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Sensor Networks and Self-stabilization TDMA Clustering Conclusion

Example

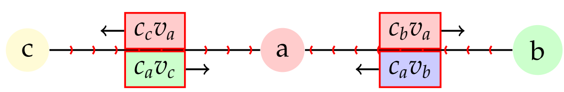
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Sensor Networks and Self-stabilization TDMA Clustering Conclusion

Example

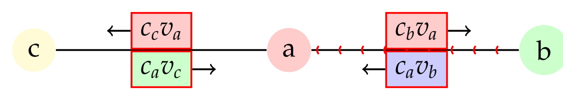
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Sensor Networks and Self-stabilization TDMA Clustering Conclusion

Example

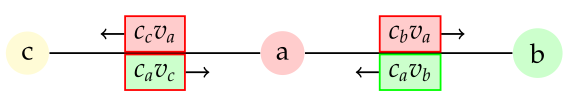
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Sensor Networks and Self-stabilization TDMA Clustering Conclusion

Example

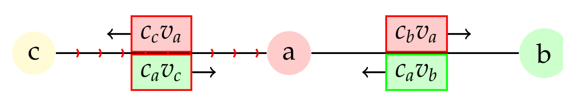
- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Sensor Networks and Self-stabilization TDMA Clustering Conclusion

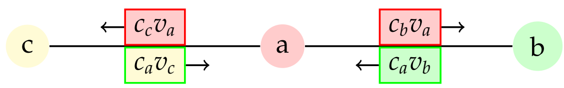
Example

- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Example

- If q receives v_p correctly, $b_q v_p$ becomes true
- $G \to A$ becomes
  for all neighbors q of p, $b_p v_q$ and $G \to A$; for all neighbors q of p, $b_p v_q$ becomes false



Cached Sensornet Transform

Periodic Retransmit

Message Corruption

Lemma (Self-stabilization)

If started from an arbitrary state, the self-stabilizing model is eventually simulated

Self-stabilizing Unison

Specification

- Each node p has a clock variable v_p
- For every neighbors p and q, $|v_p - v_q| \leqslant 1$

Self-stabilizing Unison

Specification

- Each node p has a clock variable v_p
- For every neighbors p and q, $|v_p - v_q| \leqslant 1$

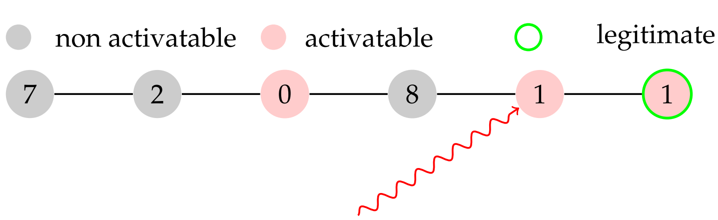
Self-stabilizing Unison

- for every neighbor q, $v_q \geqslant v_p \to v_p := v_p + 1$

Example

Self-stabilizing Unison

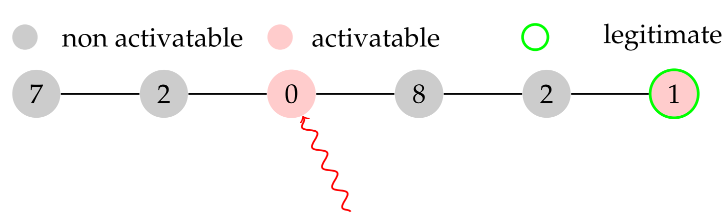
- for every neighbor q, $v_q \geqslant v_p \to v_p := v_p + 1$



Example

Self-stabilizing Unison

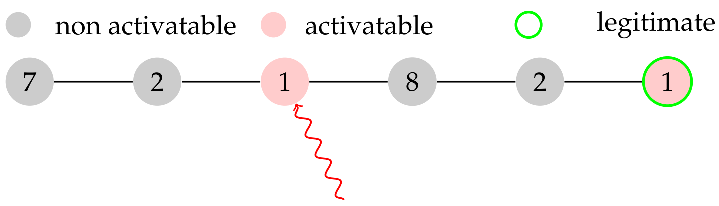
- for every neighbor q, $v_q \geqslant v_p \to v_p := v_p + 1$



Example

Self-stabilizing Unison

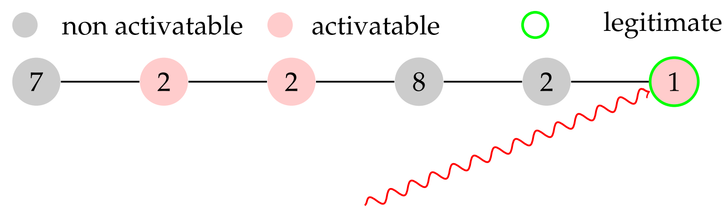
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

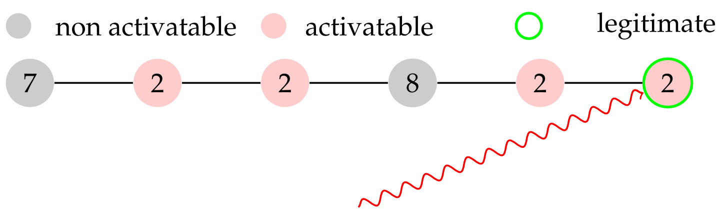
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

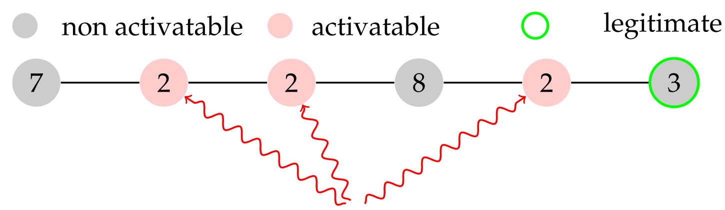
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

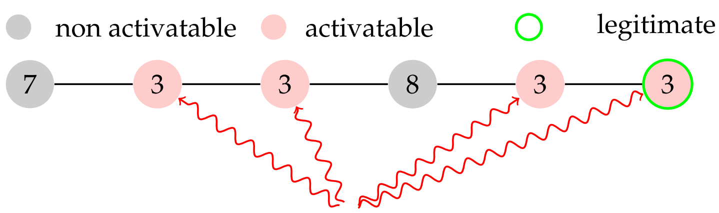
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

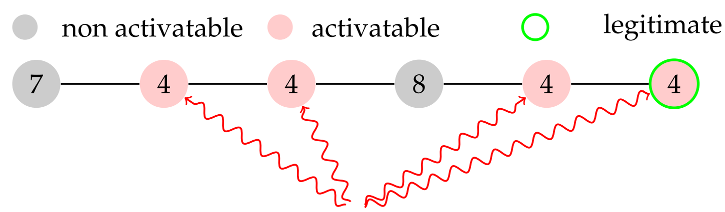
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

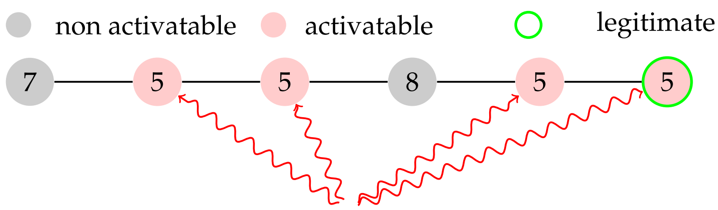
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

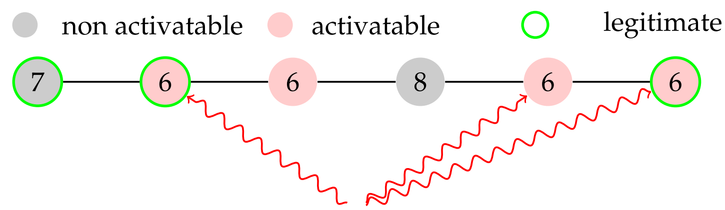
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

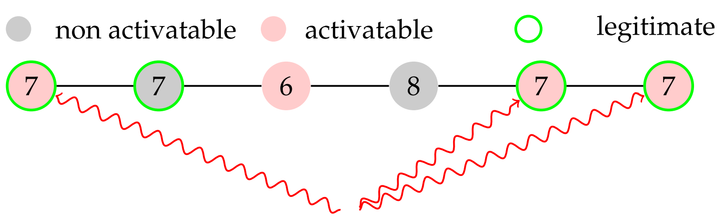
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

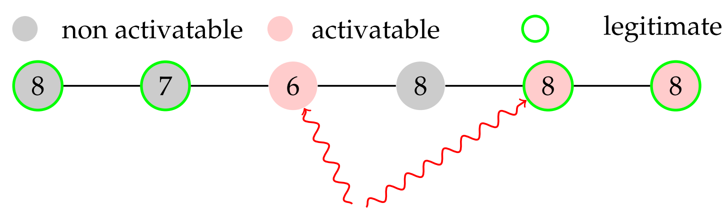
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

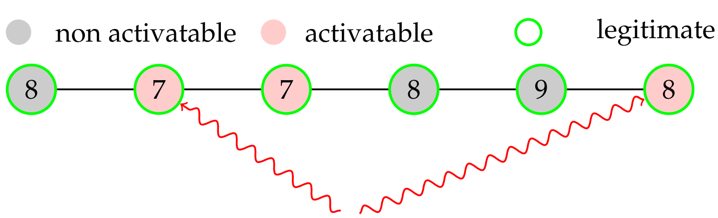
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

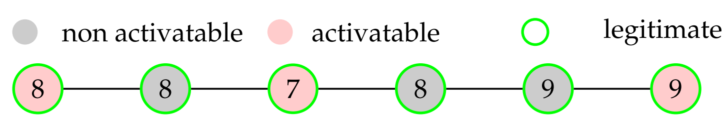
- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Example

Self-stabilizing Unison

- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$



Unison with Collisions

Specification

- Each node p has a clock variable v_p
- For every neighbors p and q, $|v_p - v_q| \leqslant 1$

Self-stabilizing Unison

- for every neighbor q, $v_q \geqslant v_p \rightarrow v_p := v_p + 1$

Unison with Collisions

Specification

- Each node p has a clock variable v_p
- For every neighbors p and q, $|v_p - v_q| \leqslant 1$

Self-stabilizing Unison with Collisions

- for every neighbor q, $c_p v_q \geqslant v_p \rightarrow v_p := v_p + 1$

Unison with Collisions

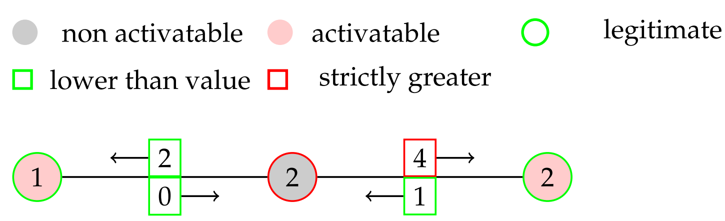
Specification

- Each node p has a clock variable v_p
- For every neighbors p and q, $|v_p - v_q| \leqslant 1$

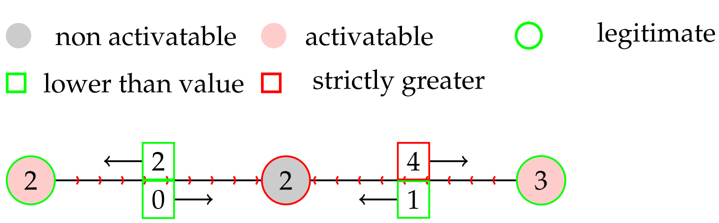
Self-stabilizing Unison with Collisions

- for every neighbor q, $c_p v_q \geqslant v_p \rightarrow v_p := v_p + 1$
- Only correctly received messages update cached variables

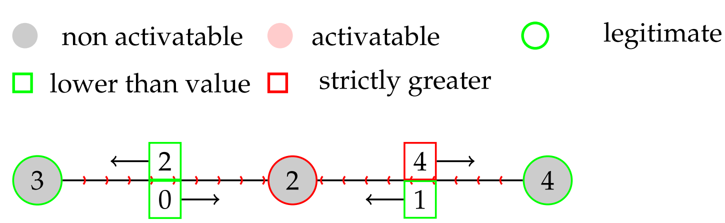
Example



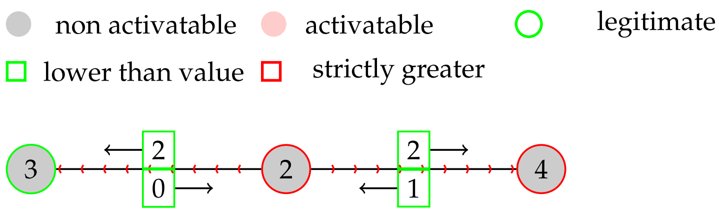
Example



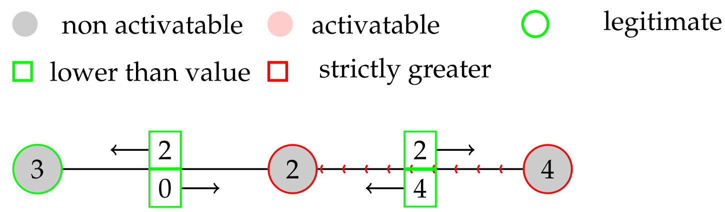
Example



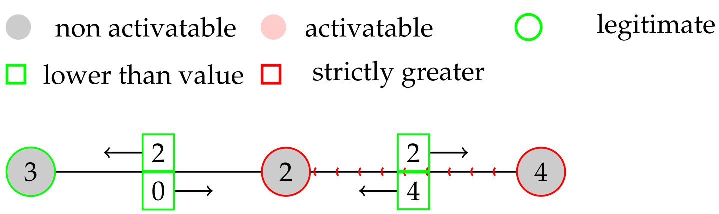
Example



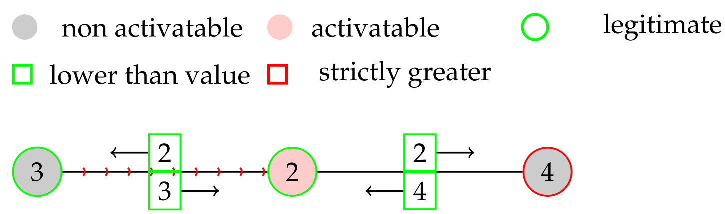
Example



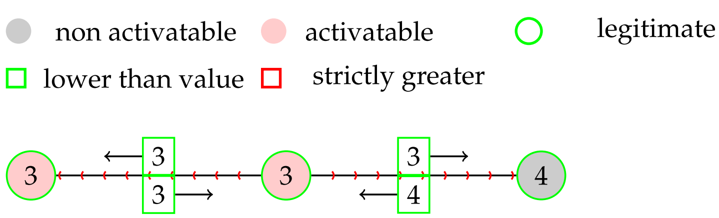
Example



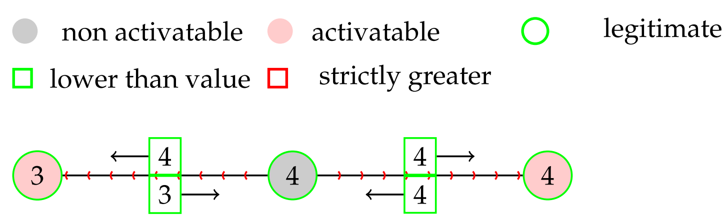
Example



Example



Example



Unison with Collisions

Cache coherence weakening

- For every neighbors p and q, $c_p v_q \leqslant v_q$

Self-stabilizing Unison with collisions

- Unison and Weak cache coherence are preserved by program executions
- Unison and Weak cache coherence eventually hold
- Some extra work is expected to get bounded clock values

Self-stabilization in Sensor Networks

Transform (i.e. Simulate) the self-stabilizing model into the sensor networks model

- [Herman 03] Cached Sensornet Transform
- Overhead is not upper bounded

Design self-stabilizing algorithms for the sensor networks model

- [Herman 03] Unison with collisions
- Proof in the model is specific to the problem

Outline

Towards an Intermediate Model

An atomic step at a node

- Compute new state, write new state at all neighbors (no collision)

Hypothesis

- Global clock, unique IDs

Solution

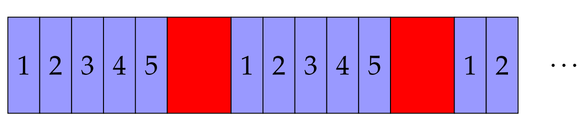
- TDMA to avoid collisions

Towards an Intermediate Model

Solution

- TDMA to avoid collisions
- assume synchronised, real-time clocks (to enable TDMA slotted time)
- but TDMA implemented using CSMA/CA as basic, underlying model

TDMA Scheduling

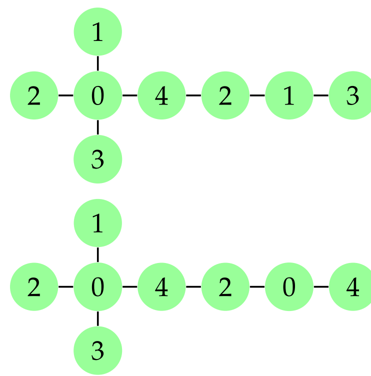


- Algorithm messages are transmitted during the "overhead" periods
- TDMA slot assignment is the output of our algorithm

Self-stabilizing TDMA for Sensors

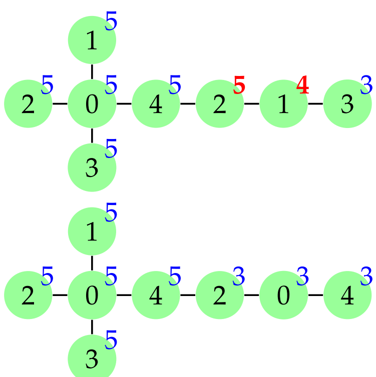
- [Kulkarni, Arumugam 03] 2-D Grids
  - nodes are aware of their positions
  - Not suitable for dynamic/faulty networks
- [Herman,Tixeuil 04] General graphs of bounded degree
  - Randomized algorithm, self-stabilizing in expected $O(1)$ time, to assign TDMA slots
  - Solution is a protocol stack based on variable propagation, minimal coloring of N^2, MIS construction, and mapping colors $\leftrightarrow$ TDMA slots

Example



- both are minimal,
- but second solution is better for time-slot assignment

Example



- both are minimal,
- but second solution is better for time-slot assignment

Overview

N^2 minimal coloring $\rightarrow$ TDMA schedule

↑

N^3 coloring + MIS $\rightarrow$ N^2 minimal coloring

↑

N^3 coloring $\rightarrow$ MIS

↑

Variables propagation $\rightarrow$ N^3 coloring

↑

CSMA/CA $\rightarrow$ Variables propagation

CSMA/CA $\rightarrow$ Variables propagation

- Wait fixed delay
  - to process received messages, and update local variables
- Wait random delay
  - to allow Aloha-style analysis for probability of collisions among neighbors
- "Age" information to remove invalid data

Shared variables $\rightarrow$ N^3 coloring

Previous L(1,0)& L(1,1) Coloring

- [Ghosh, Karaata 93] Planar graphs L(1,0)
- [Sur, Srimani 93] Bipartite graphs L(1,0)
- [Gradinariu,Tixeuil 00] General graphs L(1,0)
- [Gradinariu, Johnen 01] Colors of size n^2 L(1,1)

Our algorithm

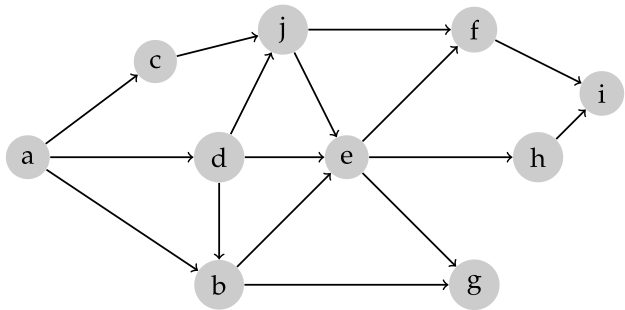
$\exists j \in N_i^3, \mathrm{color}_j = \mathrm{color}_i \rightarrow \mathrm{color}_i := \mathrm{random}(\Delta \setminus \{\mathrm{color}_j | j \in N_i^3\})$

- Stabilizes in expected $O(1)$
- Output an ID-based DAG of constant height

N^3 coloring → MIS

[Ikeda, Kamei, Kakugawa 02]

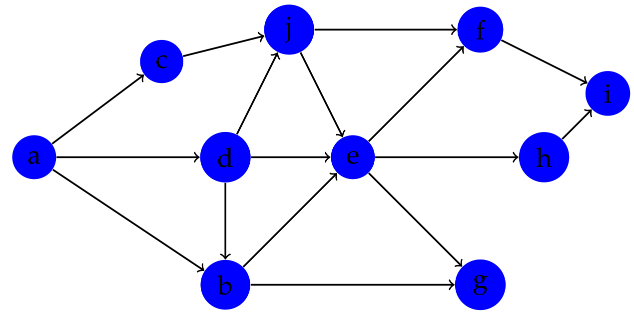
No parent in MIS → join MIS



N^3 coloring → MIS

[Ikeda, Kamei, Kakugawa 02]

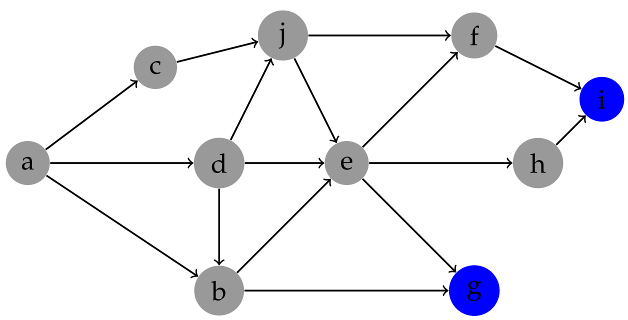
No parent in MIS → join MIS



N^3 coloring → MIS

[Ikeda, Kamei, Kakugawa 02]

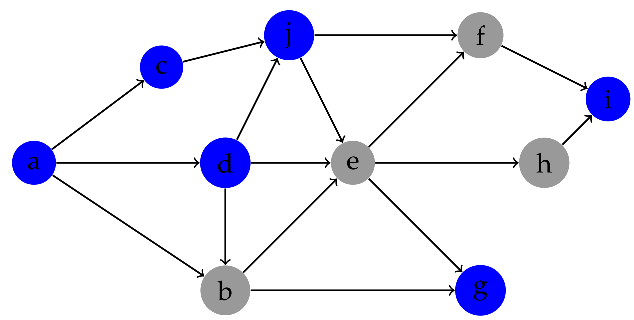
No parent in MIS → join MIS



N^3 coloring → MIS

[Ikeda, Kamei, Kakugawa 02]

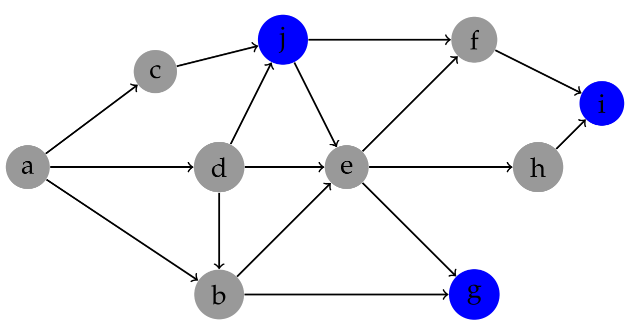
No parent in MIS → join MIS



N^3 coloring → MIS

[Ikeda, Kamei, Kakugawa 02]

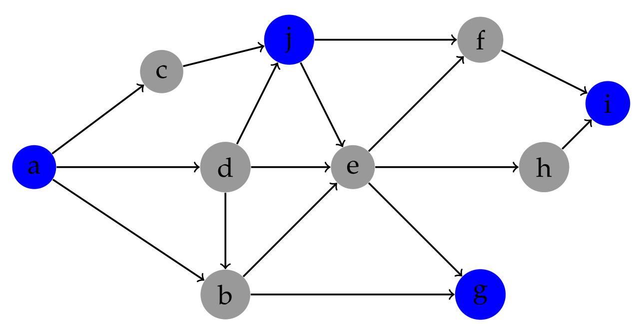
No parent in MIS → join MIS



N^3 coloring → MIS

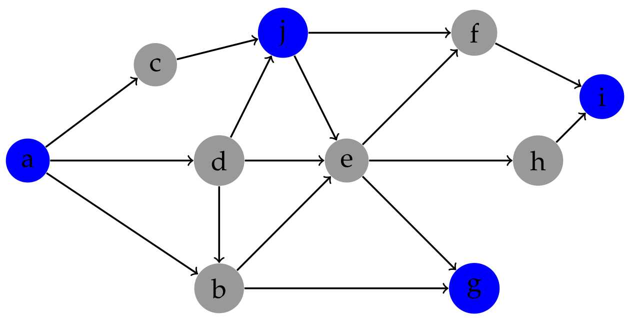
[Ikeda, Kamei, Kakugawa 02]

No parent in MIS → join MIS



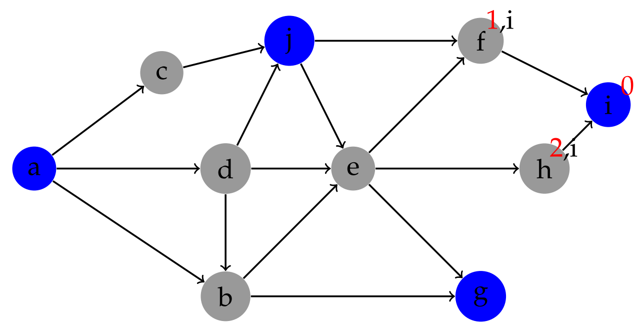
N^3 coloring + MIS $\rightarrow$ Minimal N^2 coloring

MIS $\rightarrow$ send colors to dominated nodes



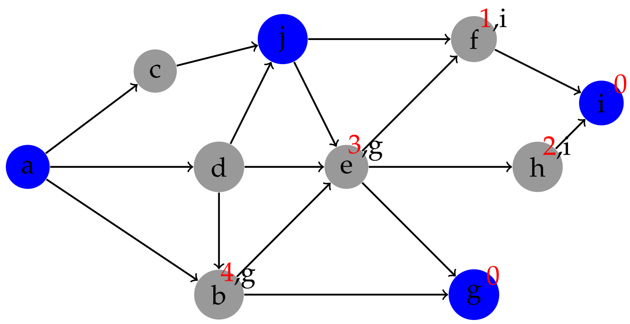
N^3 coloring + MIS $\rightarrow$ Minimal N^2 coloring

MIS $\rightarrow$ send colors to dominated nodes



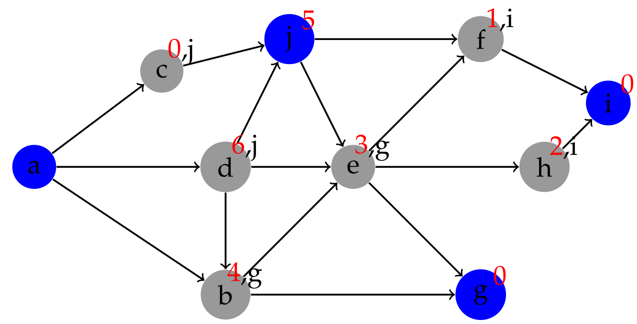
N^3 coloring + MIS $\rightarrow$ Minimal N^2 coloring

MIS $\rightarrow$ send colors to dominated nodes



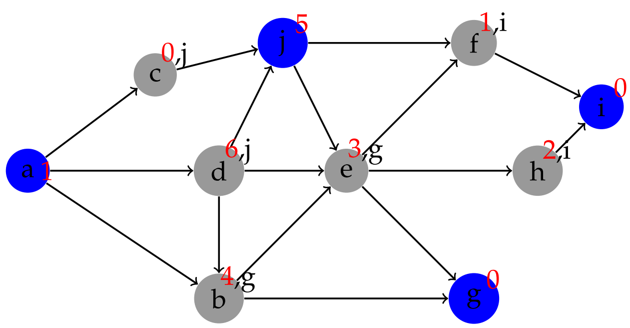
N^3 coloring + MIS $\rightarrow$ Minimal N^2 coloring

MIS $\rightarrow$ send colors to dominated nodes

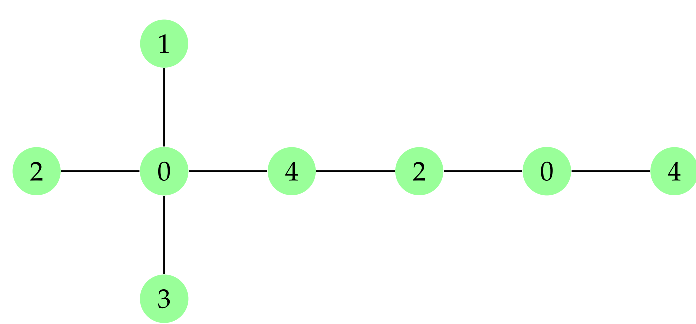


N^3 coloring + MIS $\rightarrow$ Minimal N^2 coloring

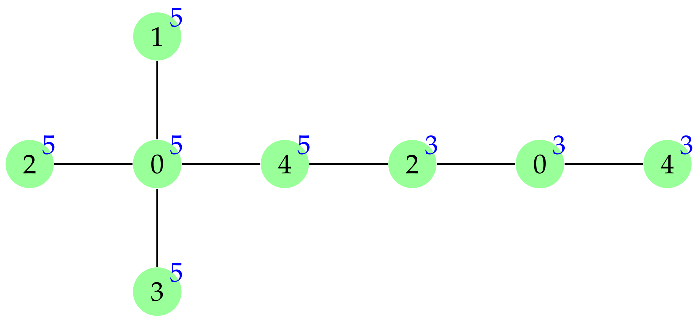
MIS $\rightarrow$ send colors to dominated nodes



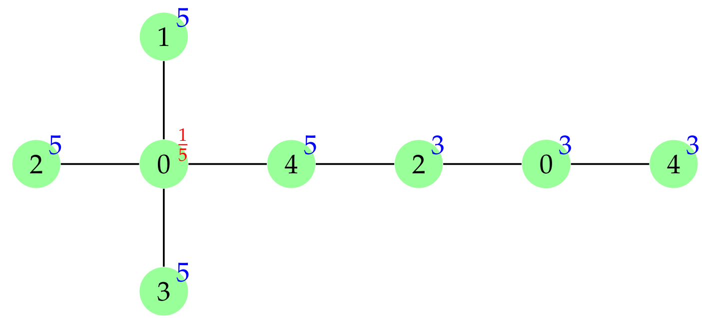
Minimal N^2 coloring $\rightarrow$ TDMA Schedule



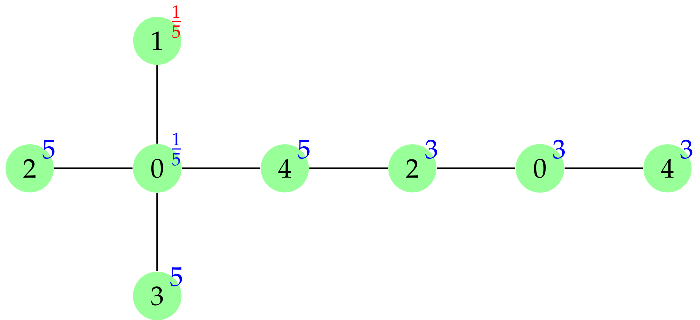
Minimal N^2 coloring → TDMA Schedule



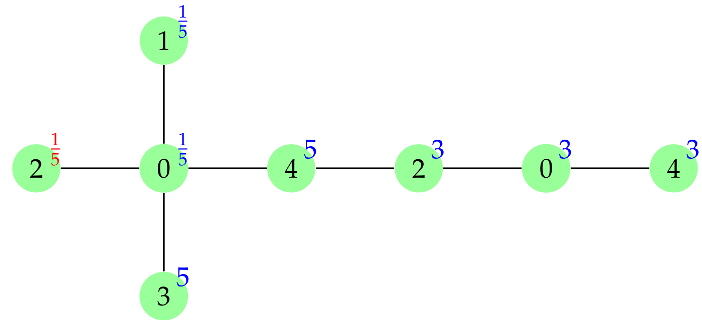
Minimal N^2 coloring → TDMA Schedule



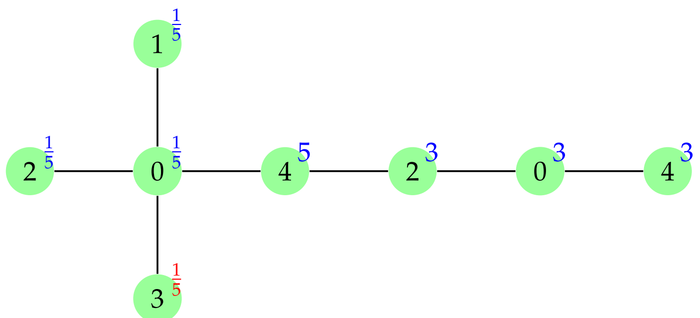
Minimal N^2 coloring → TDMA Schedule



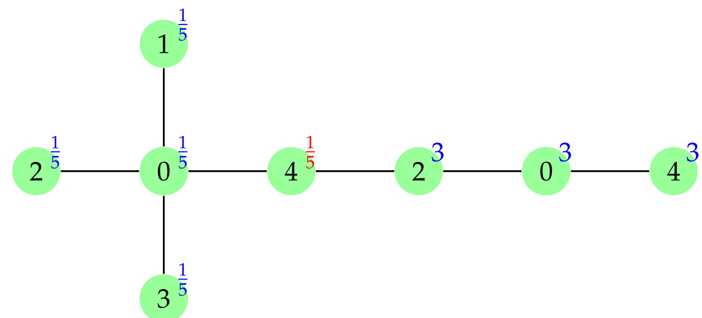
Minimal N^2 coloring → TDMA Schedule



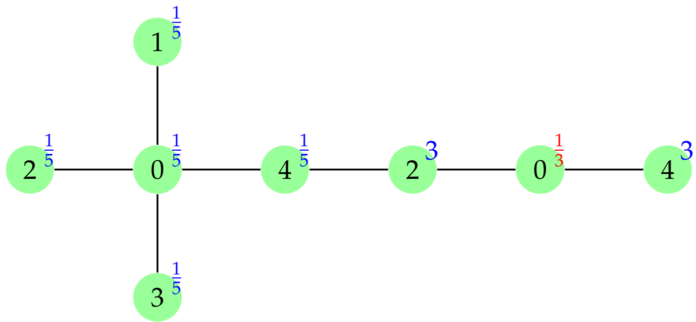
Minimal N^2 coloring → TDMA Schedule



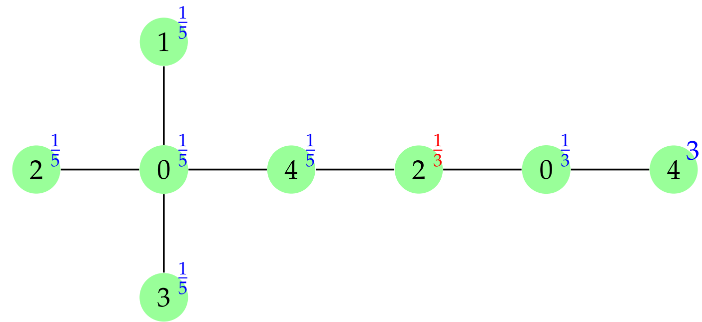
Minimal N^2 coloring → TDMA Schedule



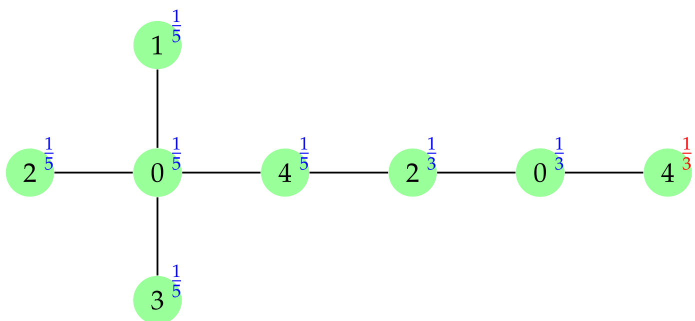
Minimal N^2 coloring → TDMA Schedule



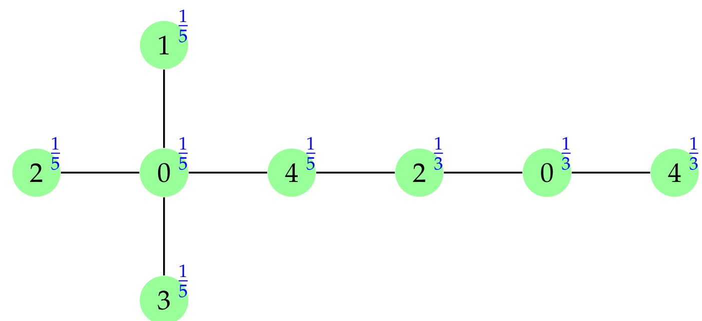
Minimal N^2 coloring → TDMA Schedule



Minimal N^2 coloring → TDMA Schedule



Minimal N^2 coloring → TDMA Schedule



Outline

Motivation

Clusters for routing

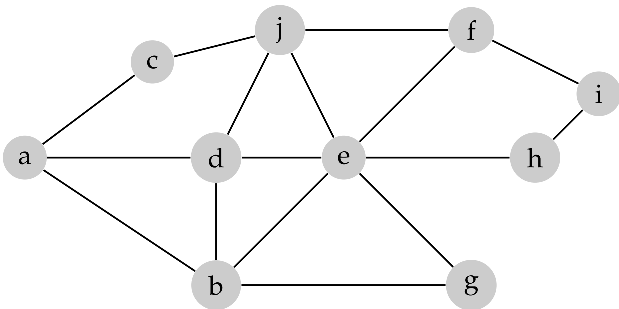
MANET routing protocols are flat, thus not scalable
Cluster-heads have extra responsibility for the routing of message

Cluster-heads should be stable

Handle departures and removals Handle node mobility

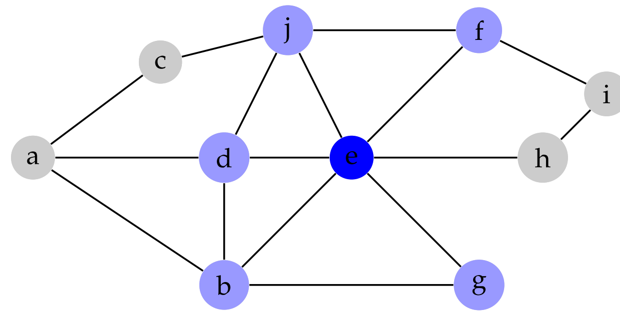
Density

$$\rho(u) = \frac{|\{e=(v,w)\in E \mid w\in\{u\}\cup N_u \text{ and } v\in N_u\}|}{|N_u|}$$



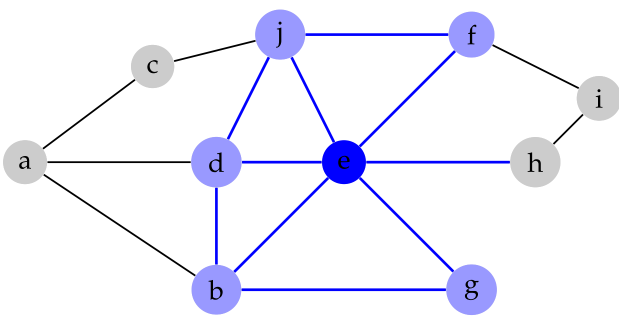
Density

$$\rho(u) = \frac{|\{e=(v,w)\in E \mid w\in\{u\}\cup N_u \text{ and } v\in N_u\}|}{|N_u|}$$

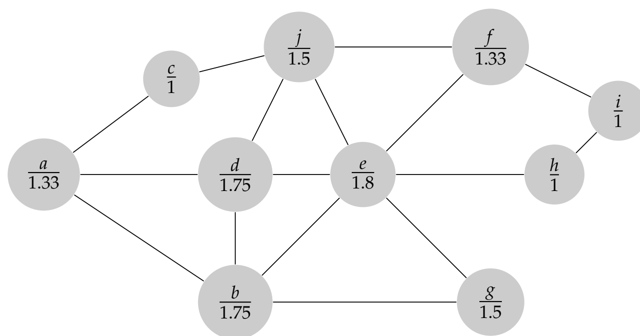


Density

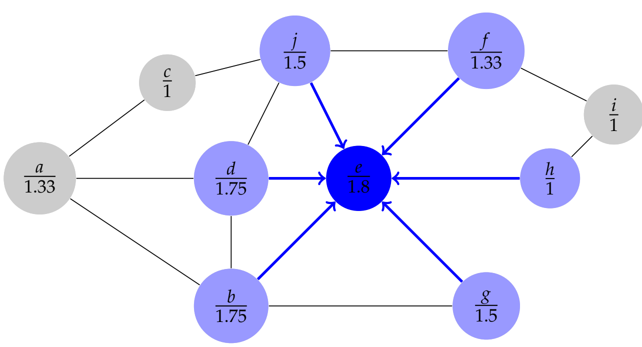
$$\rho(u) = \frac{|\{e=(v,w)\in E \mid w\in\{u\}\cup N_u \text{ and } v\in N_u\}|}{|N_u|}$$



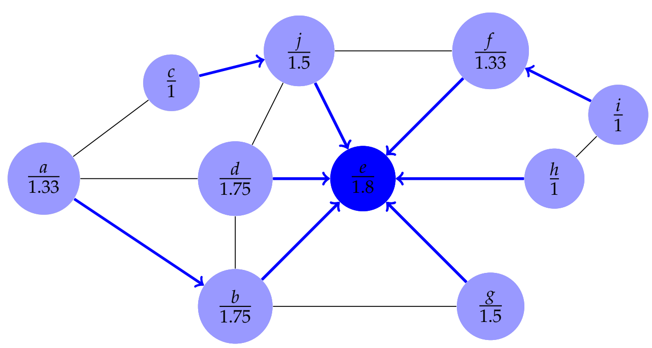
Cluster Head Heuristics



Cluster Head Heuristics



Cluster Head Heuristics

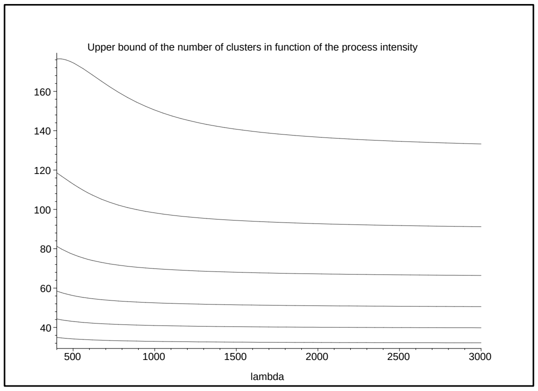


Number of Cluster Heads

- Using stochastic geometry, it is possible to calculate the mean density, and then upper bound the number of cluster-heads

$$\mathbb{P}^o_\Phi\left(\rho(0) > \max_{k=1,..,\Phi(B_0)} \rho(Y_k)\right) \leqslant \left(1+\sum_{n=1}^{+\infty}\frac{1}{n}\frac{\left(\lambda\pi R^2\right)^n}{n!}\right)\exp\{-\lambda\pi R^2\}$$

Number of Cluster Heads



Self-stabilizing Clustering

Basic Idea

- Identify N and N^2
- Compute and broadcast density
- Attach to neighbor with higher density
- use identifiers to break ties

Self-stabilizing Clustering

Basic Idea

- Identify N and N^2
- Compute and broadcast density
- Attach to neighbor with higher density
- use identifiers to break ties
- Can be $O(\text{Diameter})$ if graph is regular

Faster Self-stabilizing Clustering

Basic Idea

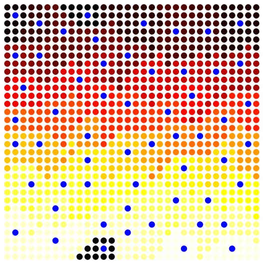
- Identify N and N^2
- Compute and broadcast density
- Random $L(1,1)$ coloring with δ^2 colors
  - This can be done in expected $O(1)$ time
- Attach to neighbor with higher density
- use colors to break ties

Faster Self-stabilizing Clustering

Basic Idea

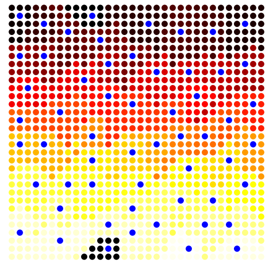
- Identify N and N^2
- Compute and broadcast density
- Random $L(1,1)$ coloring with δ^2 colors
  - This can be done in expected $O(1)$ time
- Attach to neighbor with higher density
- use colors to break ties
- Expected constant stabilization time
- Use lexicographic order $(\text{density}, \text{color})$

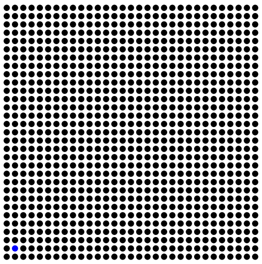
Simulation Results



	R = 0.05		R = 0.08		R = 0.1	
	With DAG	No DAG	With DAG	No DAG	With DAG	No DAG
# clusters	61.0	61.4	19.2	19.5	11.7	11.7
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	2.6	2.6	3.1	3.1	3.2	3.2
average tree length	2.7	2.7	3.3	3.3	3.5	3.5

Simulation Results



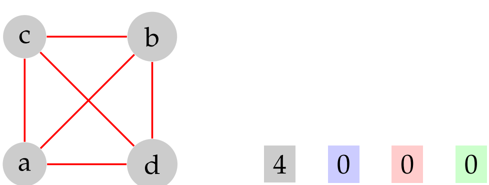


	R = 0.05		R = 0.08		R = 0.1	
	With DAG	No DAG	With DAG	No DAG	With DAG	No DAG
# clusters	52.8	1.0	29.3	1.0	18.5	1.0
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	3.4	29.1	4.1	19.1	3.6	6.5
average tree length	3.7	83.4	4.7	100.5	4.5	32.1

How fast is the coloring ?

Model

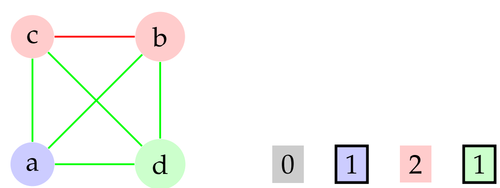
Urns and balls



How fast is the coloring ?

Model

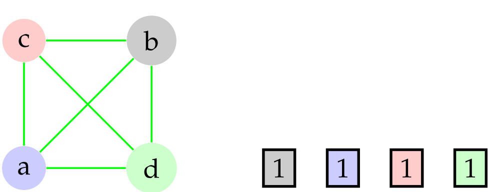
Urns and balls



How fast is the coloring ?

Model

Urns and balls



How fast is the coloring ?

Model

Urns and balls

Expected stabilization time

$$\mathbb{E}[N] = V_0$$

$$V_i = \frac{1}{1 - p_{i,i}} \left(1 + \sum_{j=i+1}^{L-1} p_{i,j} V_j \right) \text{ for } i = L-2, \ldots, 0$$

with $V_{L-1,L-1} = 1/(1 - p_{L-1,L-1})$.

How fast is the coloring ?

Model
Urns and balls

What is missed by the model ?

How fast is the coloring ?

Model
Urns and balls

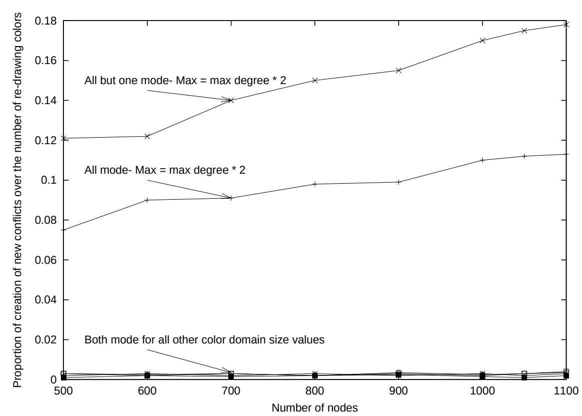
What is missed by the model ?

How fast is the coloring ?

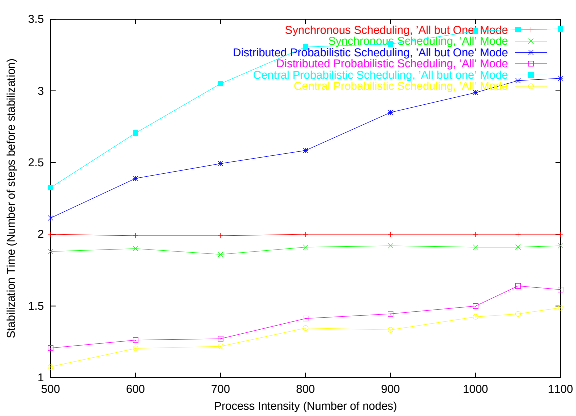
Model
Urns and balls

What is missed by the model ?

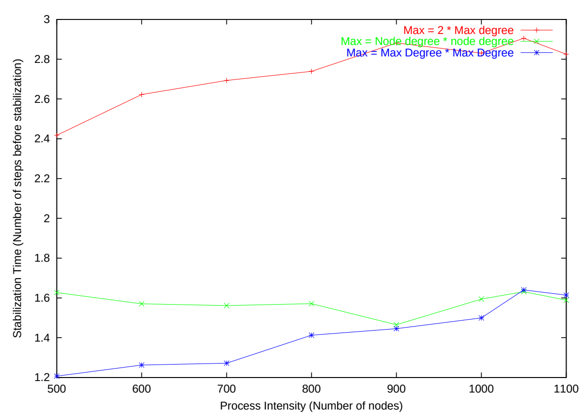
How fast is the coloring ?



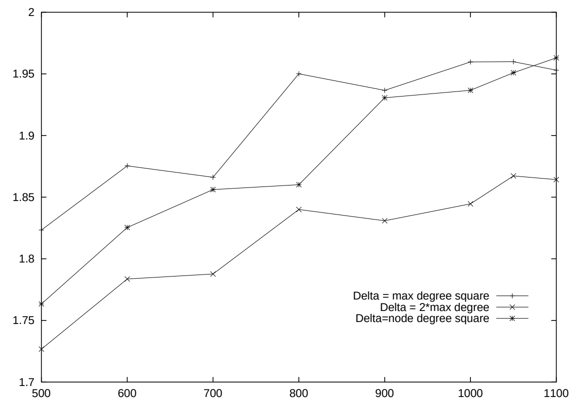
How fast is the coloring ?



Influence of the color domain



Influence of the color domain



Improving Stability

- When two nodes compete for being cluster-heads (same density), the former cluster-head wins
- Color is still used to break remaining ties
- Clusters merge if cluster-heads are 2 hops apart
- Still exp. Constant stabilization time

Improving Stability

Random moves at random speeds

- Observe every 15 seconds for 2 minutes

Pedestrians (0-1.6 m/s)

- Original algorithm: 78% re-election
- “Stable enhanced” algorithm: 82% re-election

Cars (0-10 m/s)

- Original algorithm: 25% re-election
- “Stable enhanced” algorithm: 31% re-election

Conclusion

- Self-stabilization is interesting for sensor networks
  - Known SS solutions should be implemented in sensor networks
- Sensor networks are interesting for self-stabilization
  - Simple devices
  - Small operating system

Conclusion

- Self-stabilization is interesting for sensor networks
  - Known SS solutions should be implemented in sensor networks
- Sensor networks are interesting for self-stabilization
  - Simple devices
  - Small operating system
- Energy constraints and collisions make things complicated

Sensors in Action

Launch Movie