Self-stabilizing Algorithms in the Context of Wireless Sensor Networks

Gerry Siegemund

Seminar on Algorithms at Saclay
November 28th, 2014

Institute of Telematics
Hamburg University of Technology
TUHH

About me

- PhD student at Hamburg Tec.
- Institute of Telematics : Communication & Computer Science
- $2\frac{1}{2}$ years in area of practical use of self-stabilization algorithms (SSAs) in wireless sensor networks (WSNs)
- PhD Supervisor Volker Turau (next talk, in $1\frac{1}{2}$ weeks)

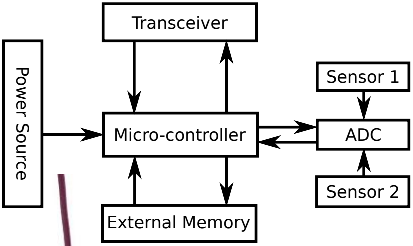
- Currently guest stay at École Polytechnique (Oct - Dec 2014)

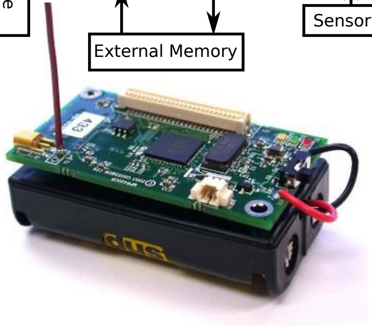
  Working with:
  Johanne Cohen, Laurence Pilard, and Khaled Maâmra

Table of Contents

Wireless Sensor

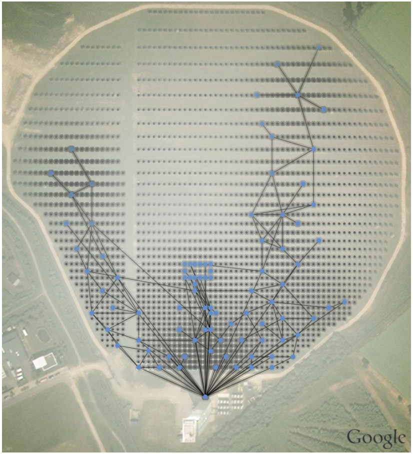
+ Small devices
+ Low energy consumption
+ Cheap
+ Sensors and communication
- Low computation power
- Small amount of memory
- Error prone (especially the communication)





Wireless Sensor Network

- Objective
  - Communication enables distributed computing
  - Parallel computation
  - Covering big, harsh areas due to wireless communication
  - Internet of Things (IoT)
- Challenges
  - Single point of failure has to be avoided
  - Transient faults need to be handled
  - ...



Wireless Sensor Network

- Solar power plant Jülich Germany
- 2000+ mirrors reflect sunlight to a tower
  $\rightarrow$ generates energy
- Each mirror has a sensor node to drive the mirror into safety position
- Cloud shift, sand storm
  - Biggest problem: Density too high $\rightarrow$ package collision

Stefan Unterschütz and Volker Turau

Reliable Signaling an Emergency Shutdown in Large-Scale, Wireless Controlled Industrial Plants, In Proceedings of the 10th ACM International Symposium on Mobility Management and Wireless Access (MOBIWAC '12), 2012.

Model

Modeling WSN

- WSN $\rightarrow$ directed graph G
- Sensor nodes $\rightarrow$ vertices V
- Possibility for node v to receive a message from node u $\rightarrow$ edge e

$\Rightarrow$ $G = (V, E)$

What is an edge again?

$p(u, v)$: Possibility of node v to receive message from node u
$e_{u,v}$ exists if $p(u, v) > Q_{min}$ (e.g., $Q_{min} > 0.7$)

Model

Modeling WSN

- WSN $\rightarrow$ directed graph G
- Sensor nodes $\rightarrow$ vertices V
- Possibility for node v to receive a message from node u $\rightarrow$ edge e

$\Rightarrow$ $G = (V, E)$

What is an edge again?

$p(u, v)$: Possibility of node v to receive message from node u
$e_{u,v}$ exists if $p(u, v) > Q_{min}$ (e.g., $Q_{min} > 0.7$)

Errors in WSN

How many erros can a happen in a WSN?

x !

Algorithms on WSNs

- Distributed system
- Algorithm design not trivial
- x errors per node
  - Lost/corrupted messages
  - Memory faults
  - Changes in physical system (node/link additions/losses)

$\Rightarrow$ Transient faults (very often)

$\Rightarrow$ Permanent faults (few)

$\Rightarrow$ Byzantine faults (few)

Errors in WSN

How many erros can a happen in a WSN?

x !

Algorithms on WSNs

- Distributed system
- Algorithm design not trivial
- x errors per node
  - Lost/corrupted messages
  - Memory faults
  - Changes in physical system (node/link additions/losses)

$\Rightarrow$ Transient faults (very often)

$\Rightarrow$ Permanent faults (few)

$\Rightarrow$ Byzantine faults (few)

Overcoming Errors in WSN

Making Algorithms in WSNs Work

- Solution A: “Thinking of all possible errors and programming the solutions into the source code.”
- Solution B: “Designing the algorithm to be inherently fault tolerant.”

Self-stabilizing Algorithms

A system is self-stabilizing if and only if:

1. Starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).
2. Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no error occurs (closure).

- Stated as rules:

  guard $\rightarrow$ statement

  if (guard == true**) do** statement (*enabled*)
- Local view
- Each node executes algorithm

Self-stabilizing Algorithms (Model)

- Scheduler (selects entities)
  - Distributed (no restriction)
  - Synchronous (all enabled instances, deterministic)
  - Central (only one instance)
  - $\Rightarrow$ in each *step* of the execution a number of entities make a *move* (i.e., enabled rules are executed)
  - Fairness
    - Unfair
    - Weakly fair
    - Fair
- Communication models
  - Message passing
  - Communication registers (read/write atomicity)
  - Locally shared memory (composite atomicity)

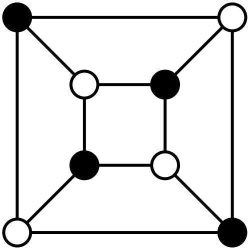
In WSN:

Distributed, unfair scheduler with message passing model

Self-stabilizing Algorithm Example

Maximal Independent Set $S_{MIS} \subset G$

Every node has a neighbor in S_{MIS} and every edge in G has an endpoint not in S.



rule : guard → statement

v is the current node, $N(v)$ is its neighborhood

Variables: $v.s \in \{Out, In\}$

Predicate: $inNeig(v) \equiv \exists u \in N(v) : u.s = In$

$v.s = Out \wedge \neg inNeig(v) \rightarrow v.s := In$
$v.s = In \wedge inNeig(v) \rightarrow v.s := Out$

Fault Tolerance

v is the current node, $N(v)$ is its neighborhood
Variables: $v.s \in \{Out, In\}$ *
Predicate: $inNeig(v) \equiv \exists u \in N(v) : u.s = In$
$v.s = Out \wedge \neg inNeig(v) \rightarrow v.s := In$
$v.s = In \wedge inNeig(v) \rightarrow v.s := Out$

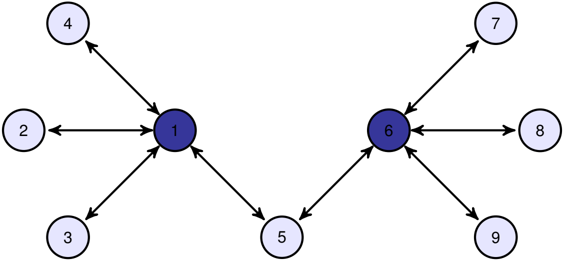
- Lost/corrupted messages *
- Memory faults (program code can hardly be recovered) *
- Changes in physical system (node/link additions/losses) *

$\Rightarrow$ Transient faults *
$\Rightarrow$ Permanent faults *
$\Rightarrow$ Byzantine faults

During fault repair: System in faulty state, detectable only in special cases (local view).

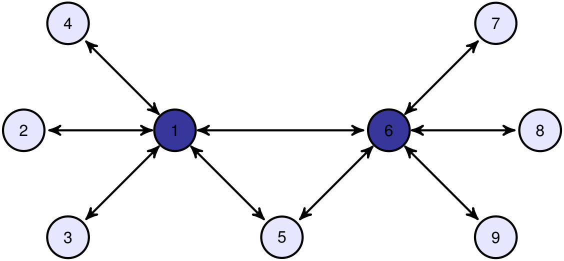
Edge Changes are ”Faults“

Example MIS



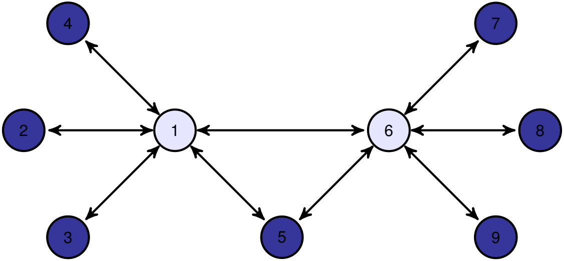
Edge Changes are ”Faults“

Example MIS



Edge Changes are ”Faults“

Example MIS



One like change $\triangleq$ 9 moves.

2

Topology Control
Neighborhood Management

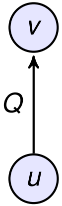
Neighborhood Management

- Problems
  - SSAs require a sufficiently stable neighborhood relation
  - Links emerge and break unpredictably
  - Challenge in WSNs: Limited resources, only subset of neighbors maintainable (scaling)
- Goals
  - Connected network
  - As few link changes as possible / stable neighborhood
  - Fast response if neighbor is lost, or joins / agile reactions

Link Quality Estimator (LQE)

Determining the Quality of a Link

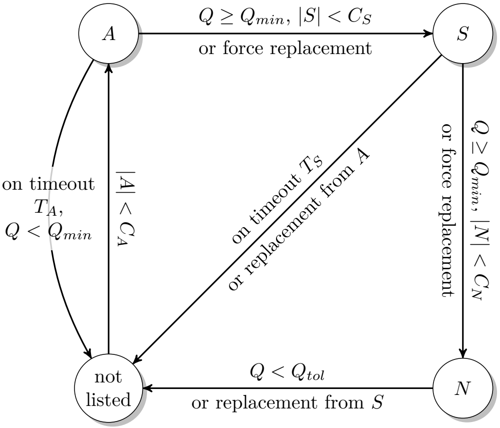
- Link Quality Estimator $\leftrightarrow$ Packet Reception Rate
  - Send sequence number (SN) at interval T_{est}
  - No message received in interval $T_{delay} > T_{est}$:
  - $\rightarrow$ mark down quality
  - Message received but SN different to SN+1:
  - $\rightarrow$ adjust quality to number of missed messages
- Examples:
  - Fuzzy-LQE
  - Kalman Filter
  - EWMA, WMEWMA
  - HoPS (Holistic Packet Statistic)



Christian Renner, Sebastian Ernst, Christoph Weyer, and Volker Turau
Prediction Accuracy of Link-Quality Estimators, Proceedings of the 8th European Conference on Wireless Sensor Networks (EWSN'11), 2011.

Neighborhood Management Algorithm

Local View of a Node v



- Restricted size Assessment List (A) / Standby List (S) / Neighborhood List (N)
- Message <ID | SN | N> sent constantly
- A is filled as messages arrive until full
- Quality is assets on all lists

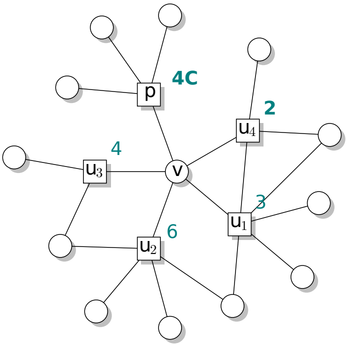
Neighborhood Management

- Promotion from S to N based on category:
  1. Symmetry
  2. Current cluster id (SS leader election)
  3. Weight (dependency of neighbors deduced by 2-hop neighborhood)
  4. LQE value
- Not promoted nodes will be evicted from S after time out
- Nodes in N are considered the neighborhood of a nodes *v*

Gerry Siegemund, Volker Turau, Christoph Weyer, Stefan Lobs, and Jörg Nolte
Brief Announcement: Agile and Stable Neighborhood Protocol for WSNs., In Proceedings of the 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (**SSS**), 2013.

Neighborhood Management Algorithm

Local View of a Node v (|N| < 5)

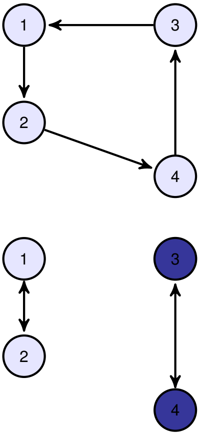


1. $weight(u) = dep(u) + 1$ if $N(u) \cap N(v) \neq \emptyset$, i.e., u is adjacent to another neighbor
2. $weight(u) = 2(dep(u) + 1)$ if $N(u) \cap N(v) = \emptyset$ and not all 2-hop neighbors of u are dependent
3. $weight(u) = C(dep(u) + 1)$ if $N(u) \cap N(v) = \emptyset$ and all 2-hop neighbors that are neighbors of u are dependent (C is a constant larger than $|\mathrm{N}|$).

Neighborhood Management Algorithm

2-hop knowledge not sufficient to decide if resulting topology is connected

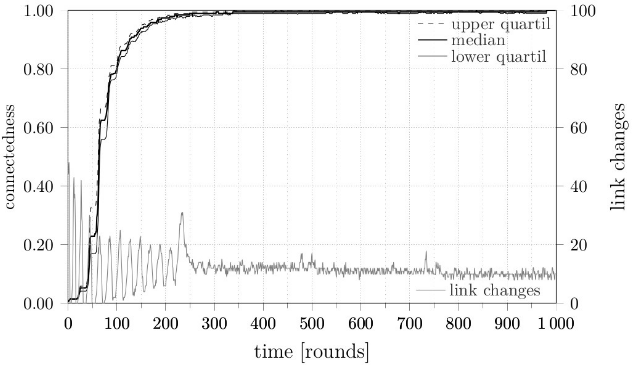
- Some global knowledge necessary
- See Category 2 : Current cluster id (leader election (SSL))
- None collateral composition (output of NMA feeds into SSL and vice versa)
- SSL slower than NMA



- Example right: restriction |N|<2

Neighborhood Management Algorithm

Simulation



- 100 simulations
- 200 nodes, N=7
- OMNeT++ with MiXiM framework (physical behavior of wireless channel)

3

Self-stabilizing Algorithms in Wireless Sensor Networks Example: PubSub-System

Publish/ Subscribe System

Topic Based ⇐

- Publisher : Sends messages
- Subscriber : Gets messages from all publishers
- Channel : Defines a certain topic

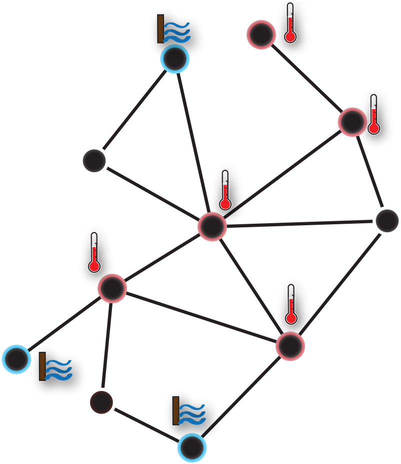
Content Based

- Publisher : Sends messages
- Subscriber : Gets messages from all publishers
- Filter : Filters messages by content to be rooted to appropriate subscriber

Topic Based Pub/Sub Systems in WSN

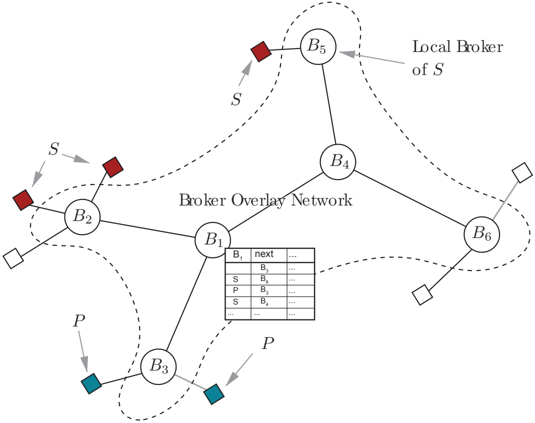
Usage in WSN

- Publisher (P)
  - Acquire sensor data
  - Aggregate values
  - Publication (Pub-Msg), Advertisement (Adv-Msg)
- Subscriber (S)
  - Actuator
  - Sink
  - Subscription (Sub-Msg)
- Channels (C)
  - Control channel
  - Collection channel



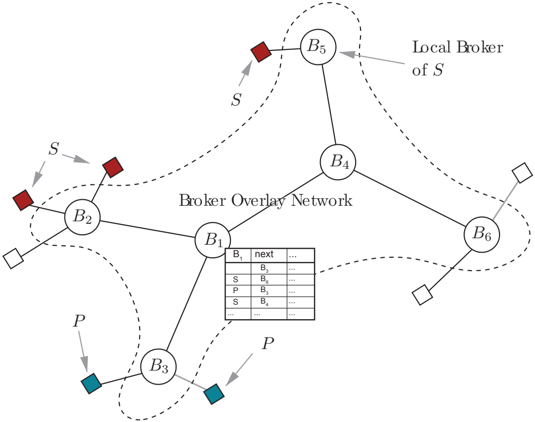
Common Approach

- P advertises generation of data for C(s)
- S subscribe to C(s)
- Rooting tables are built to “connect” P & S
  - Broker nodes (Fig.)



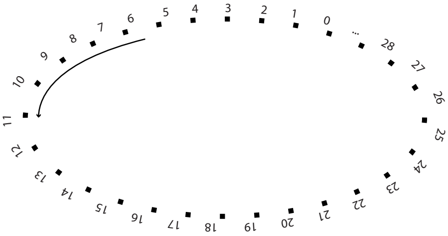
Self-Stabilizing Attempt

- Constant resending of advertisement and subscription
- Fixed time-out for renewal
- On given broker overlay (Jaeger)
  - Overlay needs to be constructed (self-stabilizing)
  - Long delay vs dead node



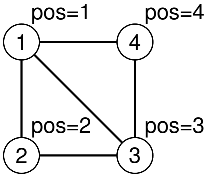
Mühl, Gero, Jaeger, Michael, Herrmann, Klaus, Weis, Torben, Ulbrich, Andreas, and Fiege, Ludger
Self-stabilizing publish/subscribe systems: Algorithms and evaluation, In Proceedings of the Parallel Processing (Euro-Par), 2005.

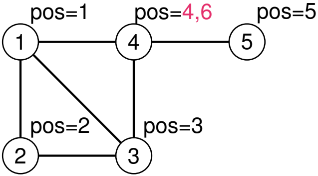
A Virtual Ring



- Ordered structure of node positions
- Each node has one prede-/suc-cessor
- Routing over all positions is straight forward
- Well known structure (P2P Overlays, *Chord*)
- each **physical** node may have multiple positions on **virtual** ring

A Virtual Ring

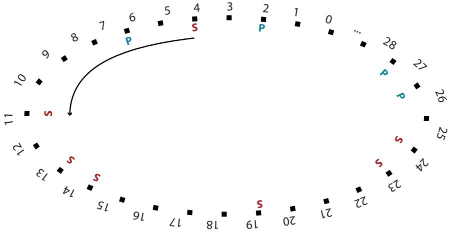




- each **physical** node may have multiple positions on **virtual** ring
- physical structure might not contain straight forward ring

- Example left: Node 4 has two positions on virtual ring

A Virtual Ring in a Pub/Sub Setup



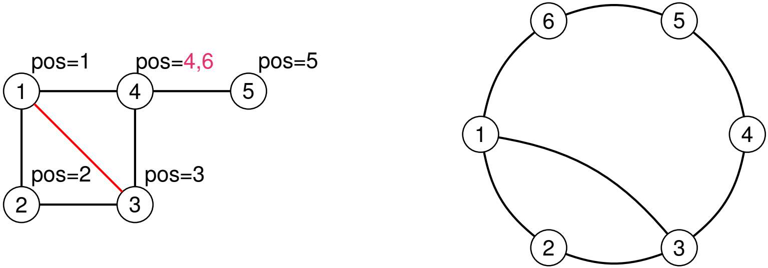
- Sub-Msg travel around the ring
  → received by every node
  → could as well be omitted
- Adv-Msg not necessary
  → no "direct" routing between P and S
- Pub-Msg has to travel to every node → unnecessary
  → Not close to shortest path

- Subscription message: notification from subscriber
- Publication (/data) message, send by Publisher
- Advertisement message used to build paths between publisher and other nodes

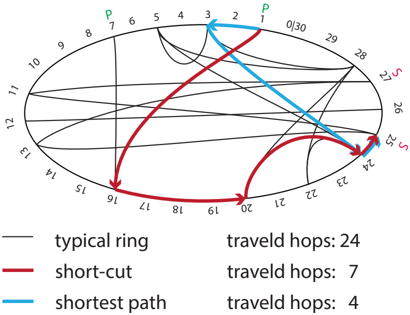
A Virtual Ring with Short-Cuts

Short-Cut

Actual connection between nodes (in underlying topology) which is not already part of the ring topology



A Short-Cut Virt. Ring in a Pub/Sub Setup



- Sub-Msg travel around the ring
  → received by every node
- Adv-Msg still not necessary

→ Pub-Msg are routed from S to next S

- Sub-Msg includes all positions (Pos) of S
- Each node stores, for all its own pos, the next S
- When a short-cut exists it is taken

→ much shorter path compared to original ring approach

A Short-Cut Virt. Ring in a Pub/Sub Setup

Algorithm 1 Handling of PUB and SUB messages

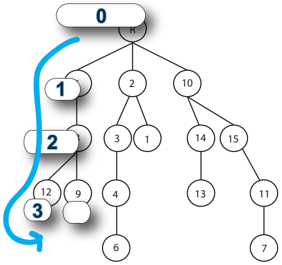
F	a table per channel, with each position pos of node v, forwarding position $fPos$ for each pos, and a TTL
$homePos$	smallest of all positions of node v (i.e., $F[0][0]$)

```
function fwdPos(pos, c)
    /*returns forwarding position for a pub. for channel c from position pos*/
    /*if no subscriber found return nextPosOnRing(pos) */
    p := indexOf(pos)][1];
    if p = ⊥ then
        return nextPosOnRing(pos);
    else
        return F[indexOf(pos)][1];
    end if
end function
```

```
Reception of: PUB⟨h, p, ch, mess⟩
if ch ∈ C_S and mess not yet delivered to node then
    deliver(mess);
end if
f_p := fwdPos(p, ch);
if ¬isBetween(h,p,f_p) then
    sendOnRing(f_p, PUB⟨h,f_p,ch,mess⟩)
end if
```

```
Reception of: SUB⟨h, p, ch, s_list⟩
/*s_list = {s_0, ..., s_k}: list of positions of (re)subscriptions from node in*/
for i=0, ..., sizeOf(F) do
    find first s_j after F[i][0] counter clockwise from s_list;
    P_new :=getPosClosestTo(F[i][0], s_j)
    if F[i][1] = ⊥ || isBetween(P_new, F[i][0], F[i][1]) then
        F[i][1] := P_new;
        renew TTL for p_i with 2δ_S;
    end if
    f_p := nextPosOnRing(p);
    if f_p ≠ h then
        sendOnRing(f_p, SUB⟨h,f_p,ch,s_list⟩)
    end if
end for
```

How to Build Virtual Ring, Self-Stabilized

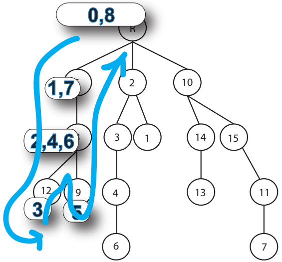


For short-cuts to be used position (order of nodes) necessary

- A tree is used as to build virtual ring, e.g., Bosilca 2009 proposed

Bosilca, George, Coti, Camille, Herault, Thomas, Lemarinier, Pierre, and Dongarra, Jack
International Conference on Parallel Computing(PARCO), 2009.

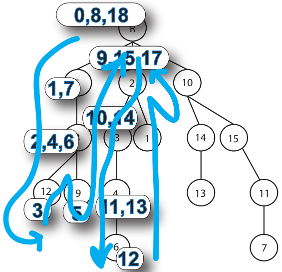
How to Build Virtual Ring, Self-Stabilized



For short-cuts to be used position (order of nodes) necessary

- A tree is used as to build virtual ring, e.g., Bosilca 2009 proposed

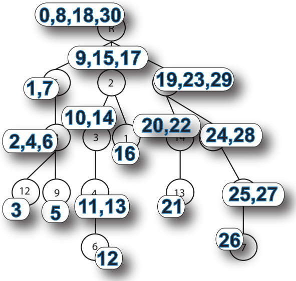
How to Build Virtual Ring, Self-Stabilized



For short-cuts to be used position (order of nodes) necessary

- A tree is used as to build virtual ring, e.g., Bosilca 2009 proposed

How to Build Virtual Ring, Self-Stabilized



For short-cuts to be used position (order of nodes) necessary

- A tree is used as to build virtual ring, e.g., Bosilca 2009 proposed

More Parallel Approach

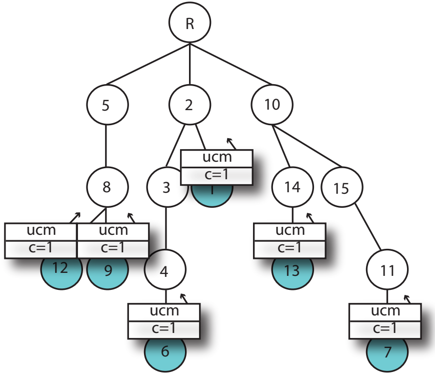
- First tree is build (SS, multiple algorithms well known)
- Each leaf node sends an *up-cast* message (ucm)
- Parent node aggregate the number of sub-tree children and send it *up*
- Root collects data of all children and computes Pos

$\rightarrow$ Root: $Pos_{first} := 0$

$$Pos_{next} := Pos_{prev} + 2 * Children_{sub-tree}$$

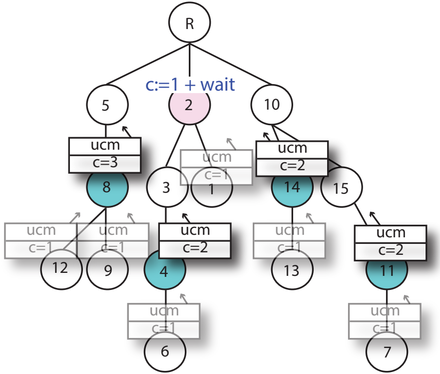
- Root sends *down-cast* message (dcm) with corresponding position to appropriate child
- Each child computes its Pos & sends dcm

Building a Virtual Ring from a Tree



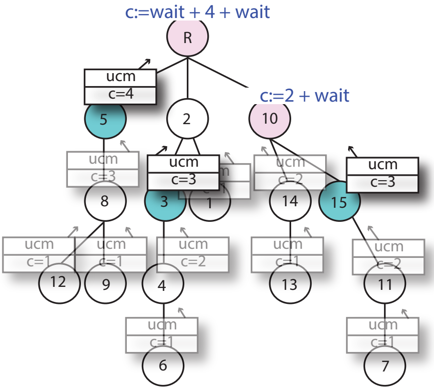
Each leaf node sends an *up-cast* message (ucm) with number of children $c = 1$

Building a Virtual Ring from a Tree



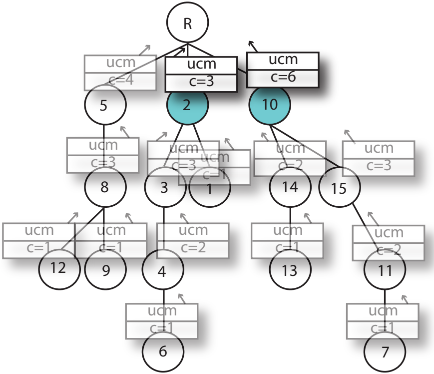
- Parent waits until all children sent dcm
- Aggregates number of sub-tree children and sends dcm with $c = \sum c_s + 1$ up to their parent

Building a Virtual Ring from a Tree



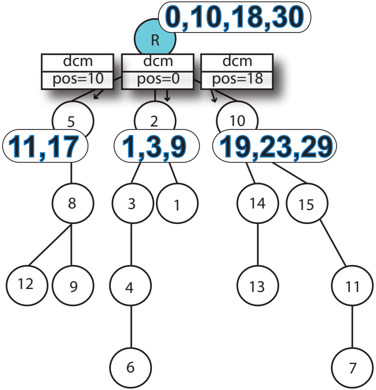
- Parent waits until all children sent dcm
- Aggregates number of sub-tree children and sends dcm with $c = \sum c_s + 1$ up to their parent

Building a Virtual Ring from a Tree



Root collects data of all children

Building a Virtual Ring from a Tree

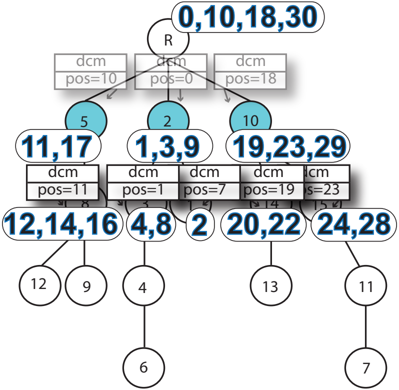


- Calculate root Pos

$\rightarrow$ $Pos_{first} := 0$
$Pos_{next} := Pos_{prev} + 2 * Children_{sub-tree}$

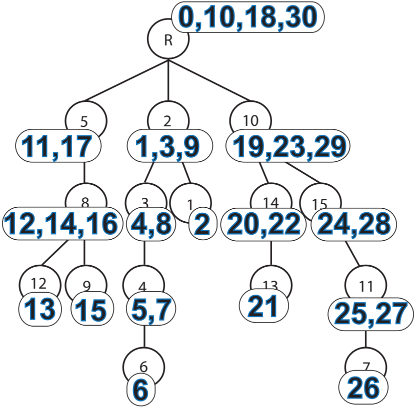
- Root sends dcm with corresponding position to appropriate child
- Each child computes its Pos

Building a Virtual Ring from a Tree



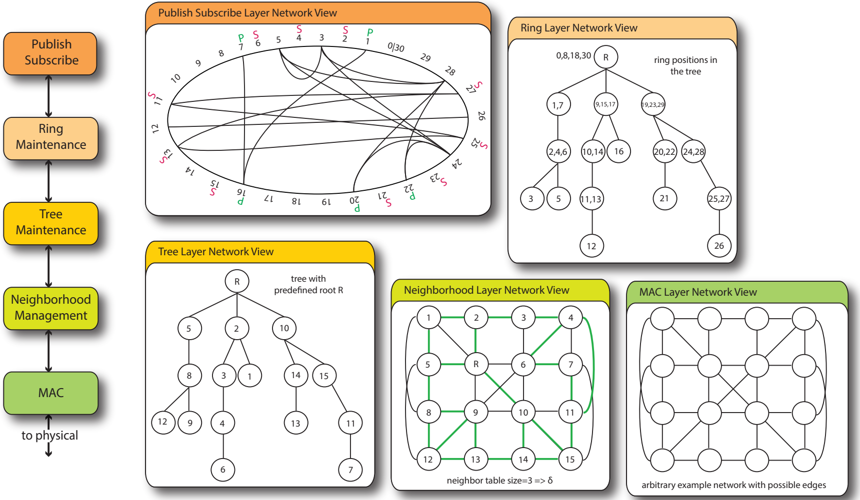
Child compute Pos & send dcm

Building a Virtual Ring from a Tree



Position aware tree is done

Overview


Publish Subscribe
Ring Maintenance
Tree Maintenance
Neighborhood Management
MAC
to physical
Publish Subscribe Layer Network View
Ring Layer Network View
0,8,18,30
ring positions in the tree
Tree Layer Network View
tree with predefined root R
Neighborhood Layer Network View
neighbor table size=3 => δ
MAC Layer Network View
arbitrary example network with possible edges

Complexity & Usability

Stacking multiple SSAs on top of each other will be SS but:

- Each SSA has its own stabilization time
  - Not known a priori
- *Upper* algorithms have no idea if *lower* one is finished
- While *lower* algorithm isn't finished *upper* one will (mostly) not converge
- Our approach stacks 3(4) SSAs (Pub/Sub - Ring - Tree - (NH))
- Convergence times (collateral composition)
  - Tree: $O(n^2)$
  - Ring: $O(n)$
  - Pub/Sub $O(n)$

Gerry Siegemund, Volker Turau, and Khaled Maâmra
Brief Announcement: Publish/Subscribe on Virtual Rings, Proceedings of the 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), 2014.

Gerry Siegemund, Volker Turau, and Khaled Maâmra
A Self-stabilizing Publish/Subscribe Middleware for Wireless Sensor Networks, Proceedings of the 2nd International Conference on Networked Systems (NetSys), 2015.

Wrap up

- Self-stabilizing algorithms can be used in WSNs
- None-masking fault-tolerance but higher traffic
- Without neighborhood management not possible
- Pub/Sub system valid example to show feasibility

Self-stabilizing Algorithms in the Context of Wireless Sensor Networks

Semina

Gerry Siegemund

Research Fellow

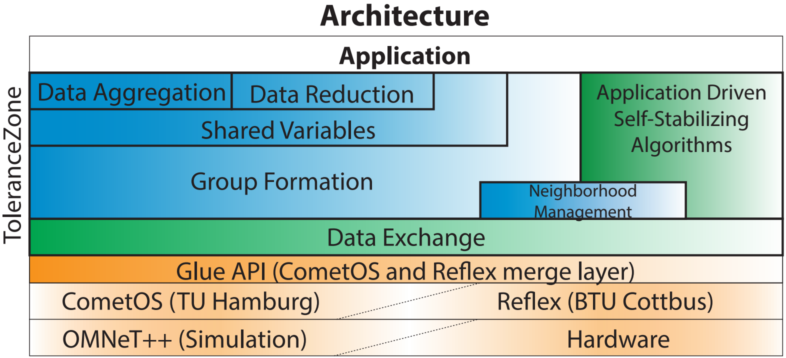
Phone +49 / (0)40 428 78 3448

e-Mail gerry.siegemund@tu-harburg.de

http://www.ti5.tu-harburg.de/staff/siegemund

Long Term Goal

- Integrating SSA in a middleware
- Offering fault-tolerance to system developers without knowledge of self-stabilization



Self-stabilizing Algorithms in the Context of Wireless Sensor Networks

Semina

N

Gerry Siegemund

Research Fellow

Phone +49 / (0)40 428 78 3448

e-Mail gerry.siegemund@tu-harburg.de

http://www.ti5.tu-harburg.de/staff/siegemund