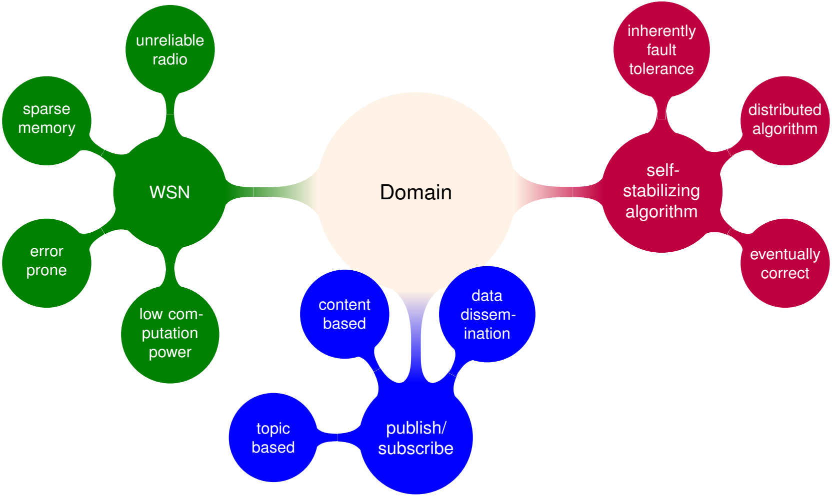
A Self-stabilizing Publish/Subscribe Middleware for WSN

Gerry Siegemund, Khaled Maâmra, and Volker Turau

International Conference on Networked Systems
March 10, 2015

Institute of Telematics
Hamburg University of Technology
TUHH

Domain Overview



Self-stabilization

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 fault occurs (closure).

- any : a faulty state (e.g., due to message loss), or wrongly initialized $\Rightarrow$ not specified
- correct : well defined: for pub/sub routing tables correct & all published message reach all subscribers

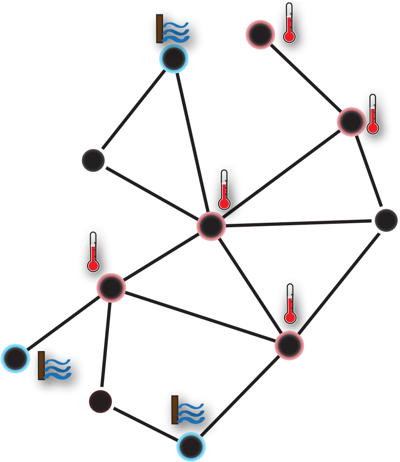
Considered model in WSN:

Distributed, unfair scheduler with message passing

Topic Based Pub/Sub Systems in WSN

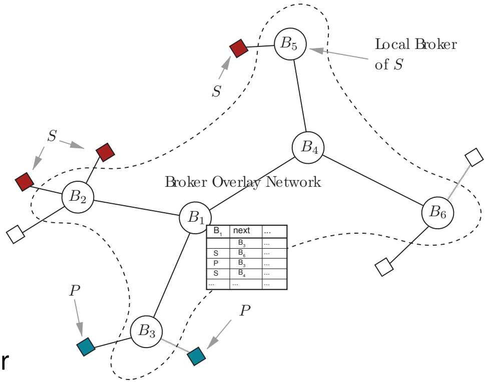
Usage in WSN

- 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



Broker Overlay as Routing Structure

- Broker nodes (Fig.)
- Broker overlay acyclic (alternatively spanning tree)
- Subscribers only attach to "leaf"-brokers
  - P advertises generation of data
  - S subscribes to content
  - Rooting tables are built to "connect" P & S at the broker nodes (filter)

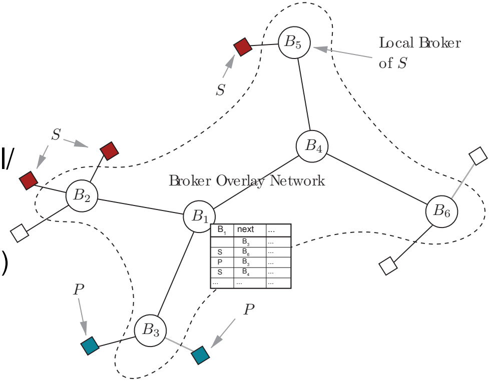


B_1	next	...
	B_3	...
S	B_6	...
P	B_3	...
S	B_4	...
...	...	...

▶ Gero Mühl et al. Self-stabilizing publish/subscribe systems: Algorithms and evaluation. 2005[15]

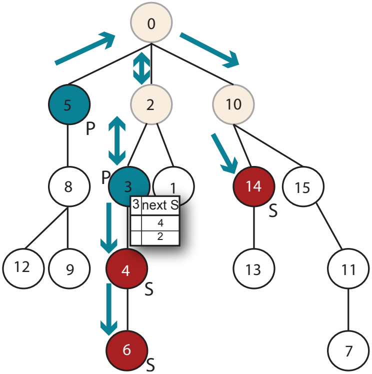
Self-stabilizing Attempt

- Periodic resending of advertisement and subscription (*leasing*)
- On given broker overlay (Mühl/ Jaeger)
  - Overlay needs to be constructed (self-stabilizing)
- Disadvantage:
  - Dedicated routing nodes
  - Adding notes that are only connected to leaf nodes ?
  - Higher density, more useful communication links unused



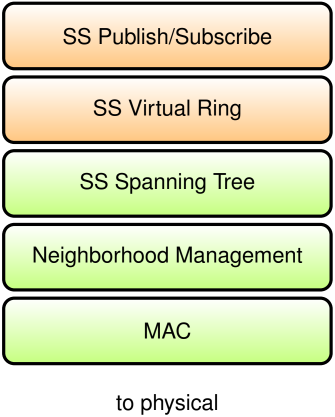
Self-stabilizing Attempt II

- Sub-Msg “flooded” (send to every node) trough the tree
- Routing (Pub-Msg) on a tree
- Routing table states if S in sub-tree/ parent
- Regular exchange of neighboring routing tables (self-stabilizing error correction)
  - Scaling: parts of neig. table are exchanged (bloom filter)
  - No renewal of routing tables
  - Self-stabilizing property uncertain



► Zhenhui Shen. Techniques for building a scalable and reliable distributed content-based publish/subscribe system. 2007[14]

Network Stack



Network Stack

Medium Access Control - Layer

- Broadcast (Back-off)
- Unicast (Back-off, ACK, Retry, . . .)

Network Stack

Neighborhood Management - Layer

- Stable neighborhood relation
- Restricted set of neighbors, i.e, Topology control
- Dynamic, addition/removal of nodes possible

▶ Gerry Siegemund et al. Brief Announcement: Agile and Stable Neighborhood Protocol for WSNs. 2013[4]

Network Stack

Self-stabilizing Spanning Tree - Layer

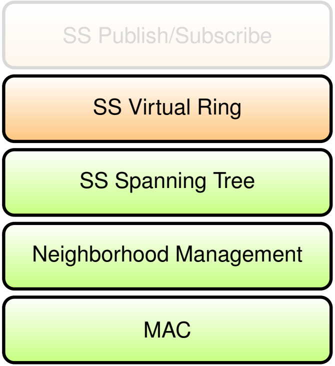
- Well known concept
- Distance to dedicated root node is shared with neighbors
- Closest node to root is excepted as parent
- Each node shares parent id

⇒ Parents can deduce children

Network Stack

Self-stabilizing Virtual Ring - Layer

- Ordered structure of node positions
- Each node has one prede-/suc-cessor
- Routing over all positions is straight forward

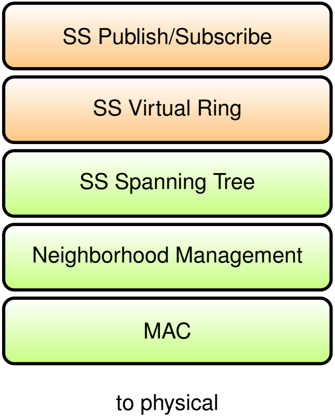


to physical

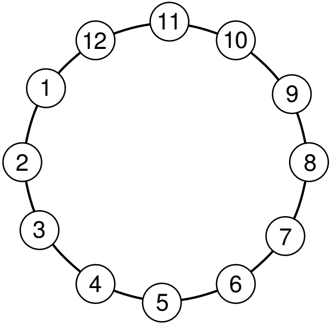
Network Stack

Self-stabilizing Publish/Subscribe - Layer

- Multiple channels
- Routing over virtual ring



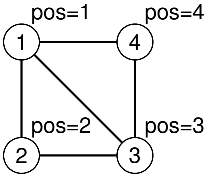
Ring Routing Structure



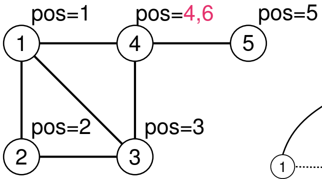
- Ordered structure of node positions
- Each node has one prede-/suc-cessor
- Routing is straight forward
- Not every topology is a ring

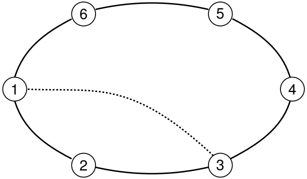
$\rightarrow$ therefore **virtual** ring will be used

A Virtual Ring



- each **physical** node may have multiple positions on **virtual** ring
- physical structure might not contain straight forward ring
- example: only positions depicted (original node ids transparent to upper layer(s))

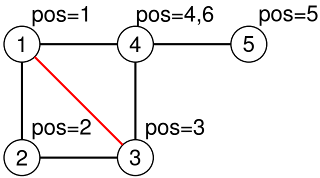


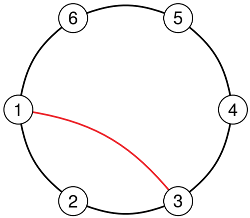


A Virtual Ring with Short-Cuts

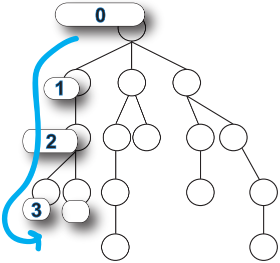
Short-Cut

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





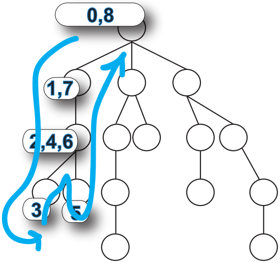
How to Build a Virtual Ring



- A tree is used as e.g. Bosilca proposed
- Sequential (depth first) search could be used
- For the ring itself Pos would not be explicitly necessary

▶ George Bosilca et al. Constructing Resiliant Communication Infrastructure for Runtime Environments. 2009

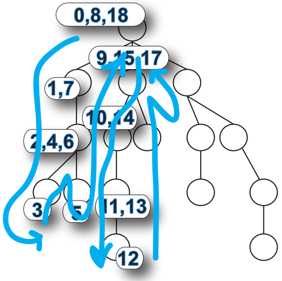
How to Build a Virtual Ring



- A tree is used as e.g. Bosilca proposed
- Sequential (depth first) search could be used
- For the ring itself Pos would not be explicitly necessary

▶ George Bosilca et al. Constructing Resiliant Communication Infrastructure for Runtime Environments. 2009

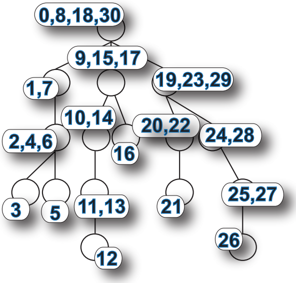
How to Build a Virtual Ring



- A tree is used as e.g. Bosilca proposed
- Sequential (depth first) search could be used
- For the ring itself Pos would not be explicitly necessary

▶ George Bosilca et al. Constructing Resiliant Communication Infrastructure for Runtime Environments. 2009

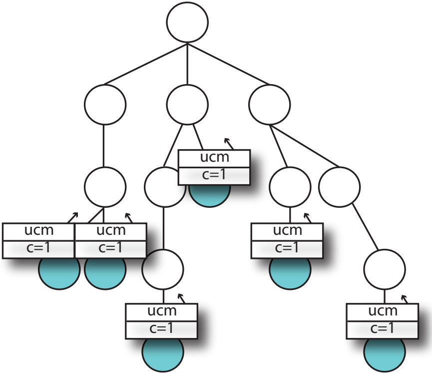
How to Build a Virtual Ring



- A tree is used as e.g. Bosilca proposed
- Sequential (depth first) search could be used
- For the ring itself Pos would not be explicitly necessary

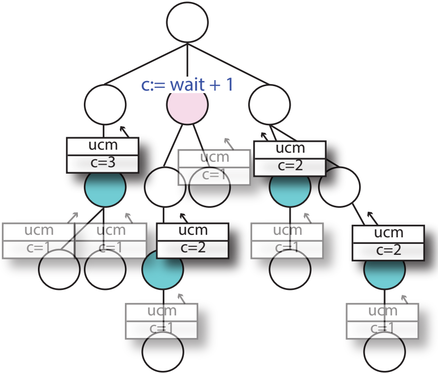
▶ George Bosilca et al. Constructing Resiliant Communication Infrastructure for Runtime Environments. 2009

Building the Ring from a Tree



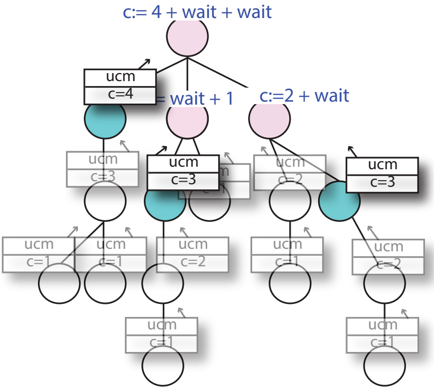
- Tree is build
- Each leaf node sends an *up-cast* message (ucm) with number of children $c = 1$

Building the Ring from a Tree



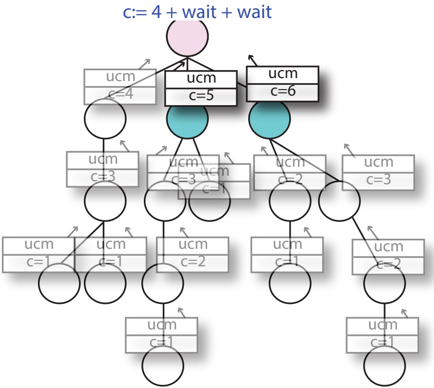
- Parent waits until all children sent dcm
- Aggregates number of sub-tree children and sends dcm with $c = \sum c_s + 1$ up to their parent

Building the Ring from a Tree



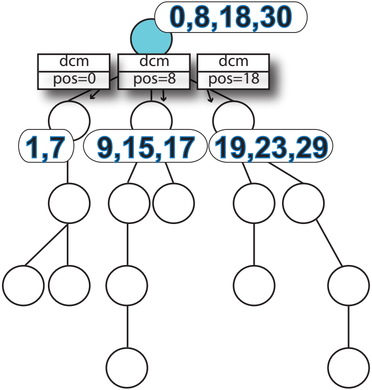
- Parent waits until all children sent dcm
- Aggregates number of sub-tree children and sends dcm with $c = \sum c_s + 1$ up to their parent

Building the Ring from a Tree



- Root collects data of all children

Building the Ring from a Tree

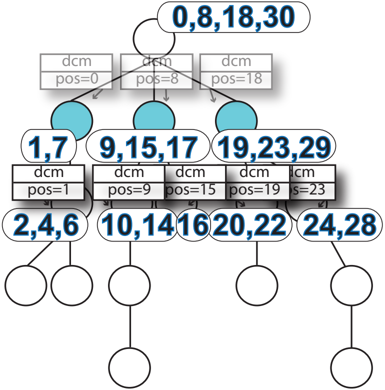


- Calculate root Pos

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

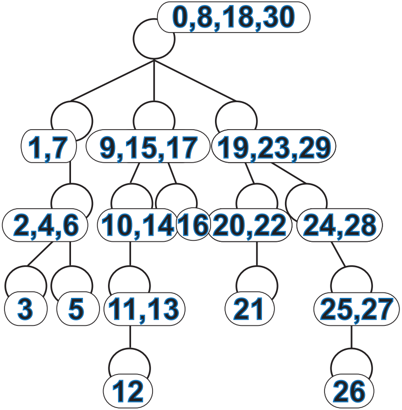
- Root sends dcm with corresponding position to appropriate child
- Each child computes its Pos

Building the Ring from a Tree



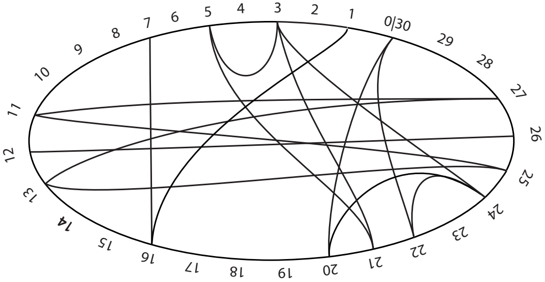
- Children compute Pos & send dcm

Building the Ring from a Tree



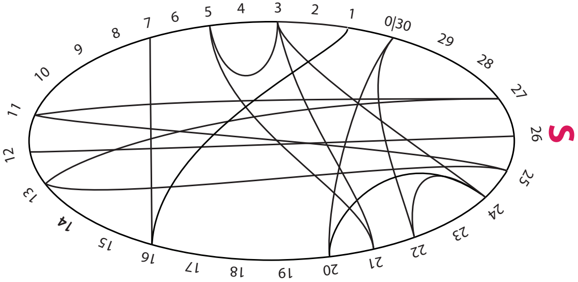
- Position aware tree is done, i.e., the virtual ring
- Number of positions on the ring : $2(n-1)$ (same as Bosilca)

Routing on a Virtual Ring with Short-cuts



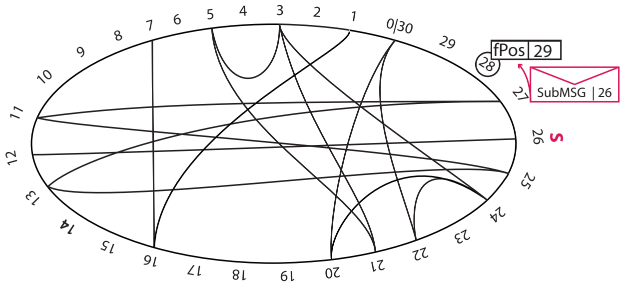
- Each node shares its positions with its neighbors

Routing on a Virtual Ring with Short-cuts



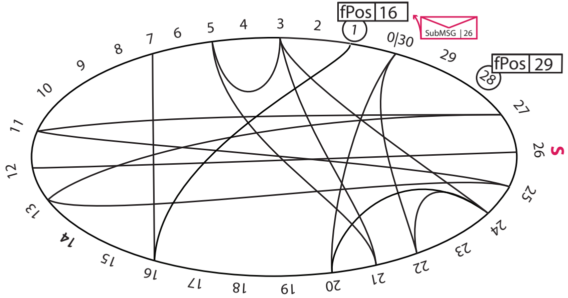
- Example with (at first) one subscriber S at position 26
- Subscribes to *1* channel

Routing on a Virtual Ring with Short-cuts



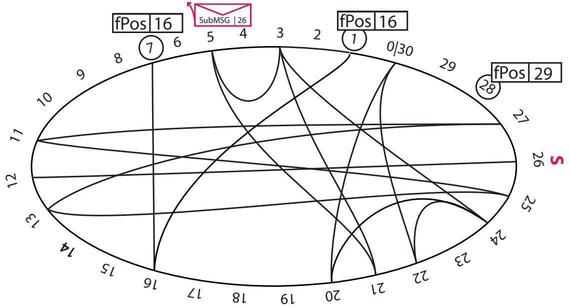
- Sub-Msg travel around the ring → received by every node SUBMSG <all *Positions* of S>
- Each node calculates and stores, for all its *own positions*, the next position on the way to the next subscriber S
- self-stabilization ensured through leasing (periodic refreshing of subscriptions / unsubscribe ⇔ timeout)

Routing on a Virtual Ring with Short-cuts



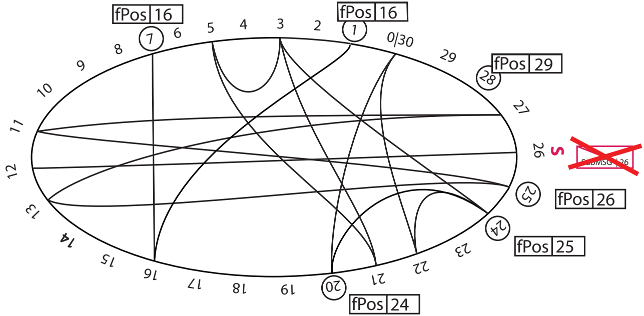
- Sub-Msg travel around the ring → received by every node SUBMSG <all *Positions* of S>
- Each node calculates and stores, for all its *own positions*, the next position on the way to the next subscriber S
- self-stabilization ensured through leasing (periodic refreshing of subscriptions / unsubscribe ⇔ timeout)

Routing on a Virtual Ring with Short-cuts



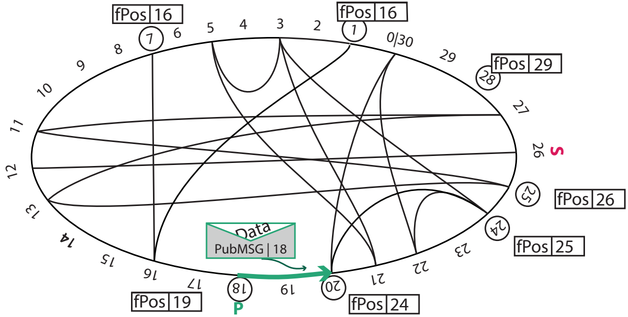
- Sub-Msg travel around the ring → received by every node SubMsg <all *Positions* of S>
- Each node calculates and stores, for all its *own positions*, the next position on the way to the next subscriber S
- self-stabilization ensured through leasing (periodic refreshing of subscriptions / unsubscribe ⇔ timeout)

Routing on a Virtual Ring with Short-cuts



- Sub-Msg travel around the ring → received by every node
  SubMsg <all *Positions* of S>
- Each node calculates and stores, for all its *own positions*, the next position on the way to the next subscriber S
- self-stabilization ensured through leasing (periodic refreshing of subscriptions / unsubscribe ⇔ timeout)

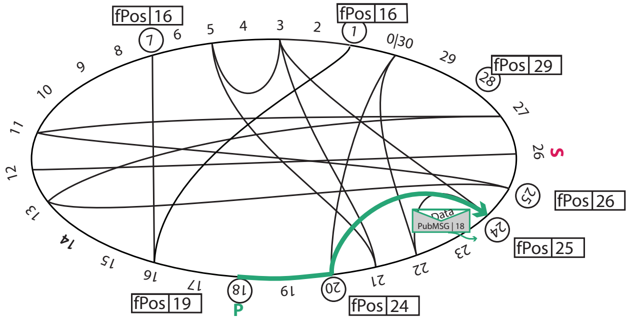
Routing on a Virtual Ring with Short-cuts



→ Pub-Msg are routed from P along the forwarding positions

- Best case routing: from P to the next S to next S, . . . , without using intermediate nodes

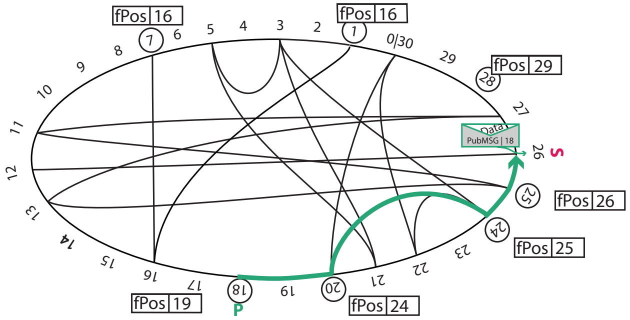
Routing on a Virtual Ring with Short-cuts



→ Pub-Msg are routed from P along the forwarding positions

- Best case routing: from P to the next S to next S, . . . , without using intermediate nodes

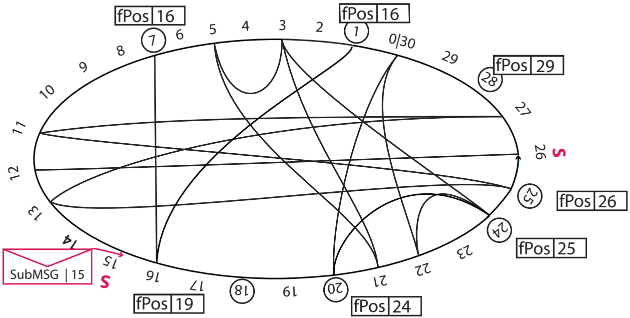
Routing on a Virtual Ring with Short-cuts



→ Pub-Msg are routed from P along the forwarding positions

- Best case routing: from P to the next S to next S, . . . , without using intermediate nodes

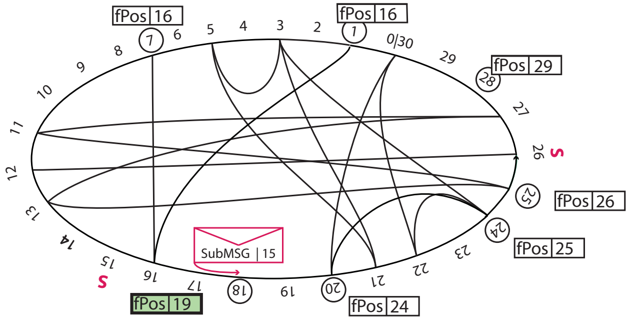
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

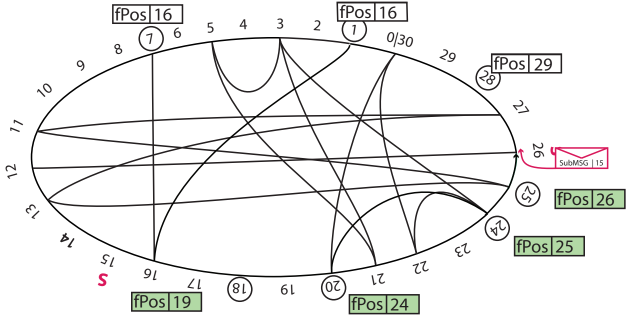
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

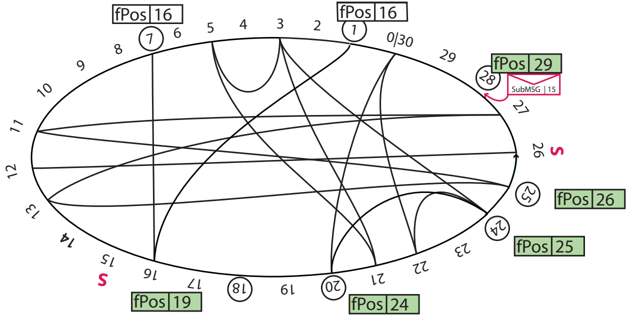
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

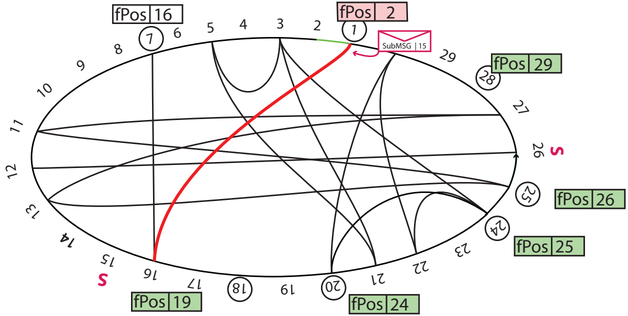
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

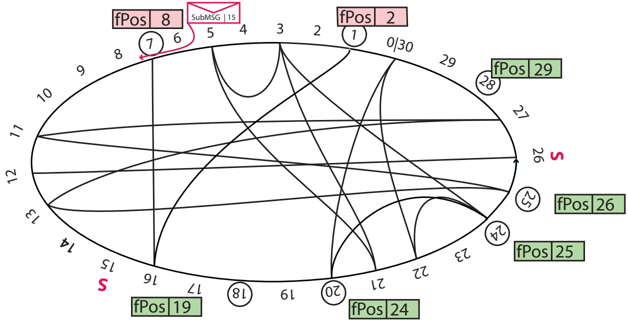
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

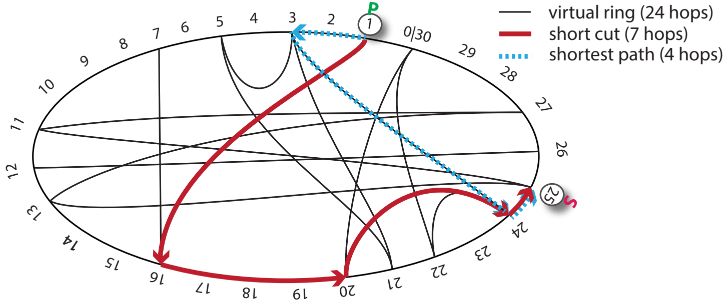
Routing on a Virtual Ring with Short-cuts



Updating routing tables:

- Only the closest S in front of a node (counter clockwise) influences the *fPos*
- Routing table at position 1 (and 7) need to be changed to avoid excluding S at position 15 from routing path

Shorter Path vs Shortest Path

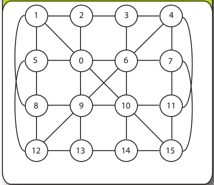


→ much shorter path

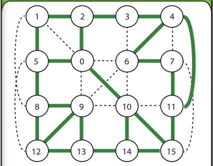
- but not shortest path (light blue)

Complete System Assessment

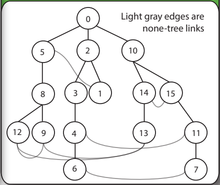
MAC Layer Network View 1



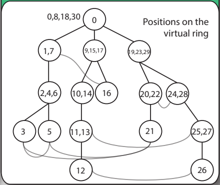
Neighborhood Layer Network View 2



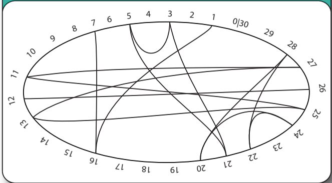
Tree Layer Network View 3



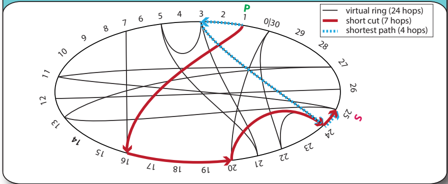
Virtual Ring Layer Network View 4



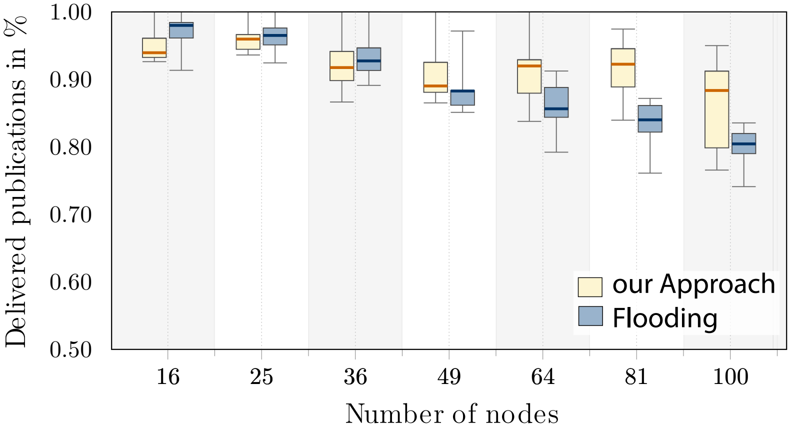
Publish/Subscribe Layer Network View 5



Routing Example on the Publish/Subscribe Layer 6



Comparison vs Flooding



- grid network, multiple densities (up to 20 nodes in range), OMNeT++ with MiXiM (collisions, fading)

Comparison vs Flooding

Message overhead

- Routing structure overhead:
  - Neighborhood management
  - Spanning tree
  - Virtual ring
- More subscribers ⇒ increased routing efforts ⇒ less gain for our approach
- Broadcast ⇒ number of messages constant (not considering message loss)

Timings for rebuilding of routing structure need to be proportional to publication interval (e.g., rebuilding routing structure more regularly then sending publications increases overhead)

A Self-stabilizing Publish/Subscribe Middleware for WSN

Gerry Siegemund, Khaled Maâmra, and Volker Turau

International Co

Gerry Siegemund

Research Associate

Phone +49 / (0)40 428 78 3448

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

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