Towards Floating Managers in Scopes

Autonomous Scopes Maintenance

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Scopes

Problem Description

Proposed Solution

Implementation Details

Results - Conclusions

Scopes Framework

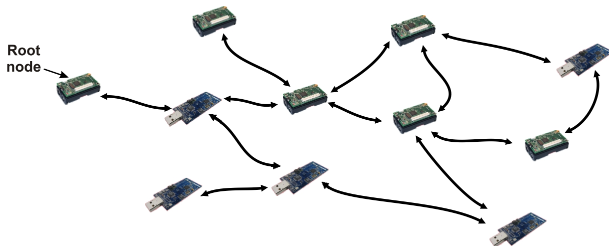
Introduction

- The Problem
  - sensornets are composed of large numbers of nodes
- What was done?
  - a mechanism to declaratively split the network into logical groups
- What is it *for*?
  - enable multitasking networks (Daniel's Diplomarbeit)
  - deploy modules selectively (ongoing Diplomarbeit)
  - define event sources, action executants (Pablo's workflow approach)
  - delimit secure boundaries (coming, CASED)
  - ...

Scopes Framework

Introduction (2)

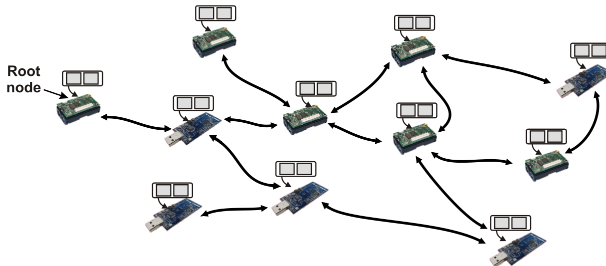
- How does it work?
  - hierarchical structure
    - base scope (World)
    - normal scopes
    - nested scopes



Scopes Framework

Introduction (2)

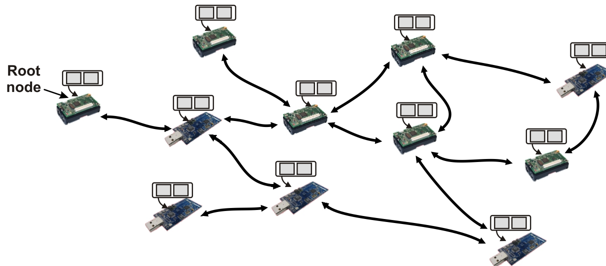
- How does it work?
  - hierarchical structure
    - base scope (World)
    - normal scopes
    - nested scopes



Scopes Framework

Introduction (2)

- How does it work?
  - hierarchical structure
    - base scope (World)
    - normal scopes
    - nested scopes

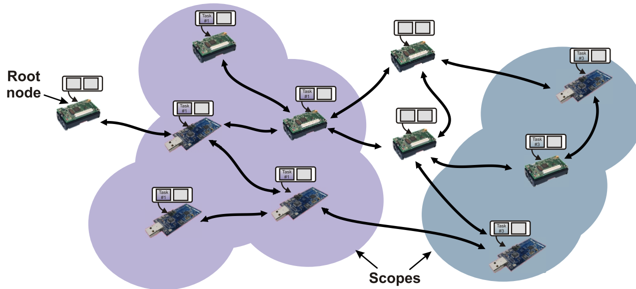


Scopes Framework

Introduction (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- How does it work?
  - hierarchical structure
    - base scope (World)
    - normal scopes
    - nested scopes



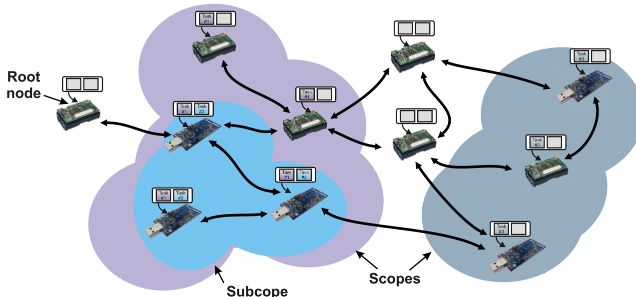
DVS

Scopes Framework

Introduction (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- How does it work?
  - hierarchical structure
    - base scope (World)
    - normal scopes
    - nested scopes



Scopes Features

TECHNISCHE UNIVERSITÄT DARMSTADT

- Declarative language for network partitioning
- bidirectional communication pattern
  - root node $\longrightarrow$ scope members
  - scope members $\longrightarrow$ root node
- automatic maintenance
  - fault-tolerant against (re)joining/leaving nodes
    - due to network dynamism or node failure

Scopes Features

- Declarative language for network partitioning
- bidirectional communication pattern
  - root node $\longrightarrow$ scope members
  - scope members $\longrightarrow$ root node
- automatic maintenance
  - fault-tolerant against (re)joining/leaving nodes
    - due to network dynamism or node failure

Scopes Features

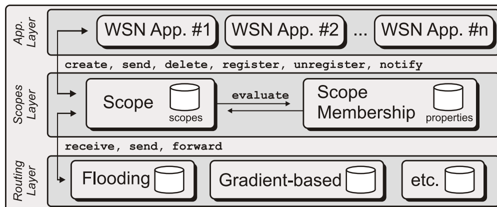
TECHNISCHE UNIVERSITÄT DARMSTADT

- Declarative language for network partitioning
- bidirectional communication pattern
  - root node $\longrightarrow$ scope members
  - scope members $\longrightarrow$ root node
- automatic maintenance
  - fault-tolerant against (re)joining/leaving nodes
    - due to network dynamism or node failure

Scopes Design

TECHNISCHE
UNIVERSITÄT
DARMSTADT

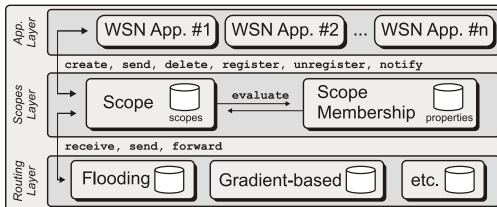
- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers



- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers

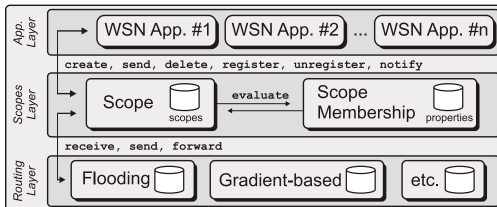


- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers

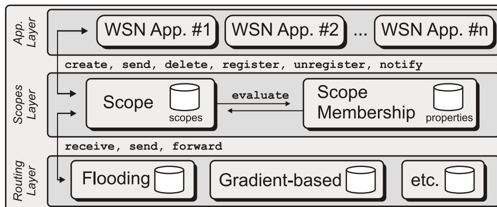


- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers

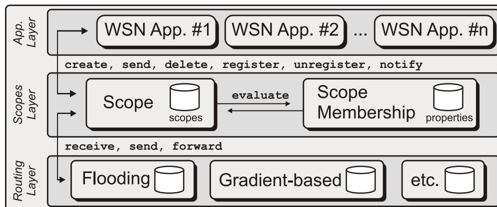


- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers

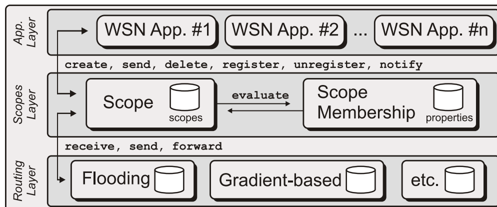


- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers

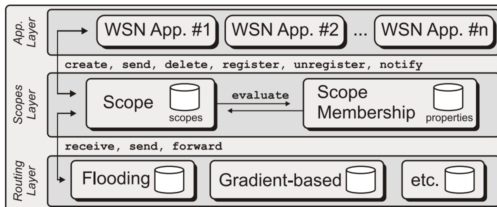


- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Scopes Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Layered architecture
  - routing, scopes and application layers
  - defined interfaces across layers



- Modular system
  - different routing algorithms possible
  - exchangeable membership evaluation
  - multiple applications, OTA deployment

Outline

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Scopes

Problem Description

Proposed Solution

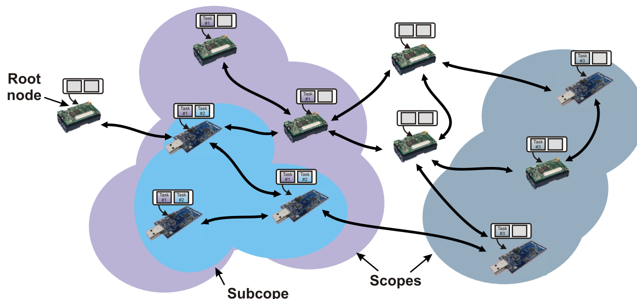
Implementation Details

Results - Conclusions

Problem Description

Dependence on Root Node

- Root node plays important role:
  - executes the *refresh mechanism*
    - if root node dies, *there is no point in keeping scope alive*



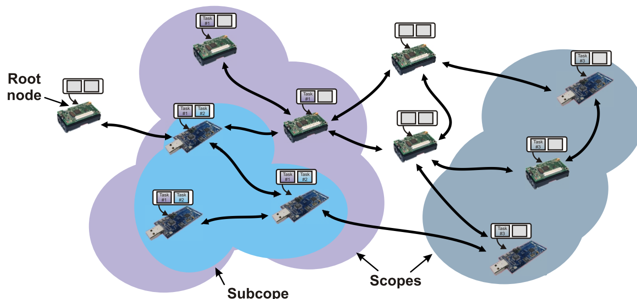
- **but...**

Problem Description

Dependence on Root Node

TECHNISCHE UNIVERSITÄT DARMSTADT

- Root node plays important role:
  - executes the *refresh mechanism*
    - if root node dies, *there is no point in keeping scope alive*



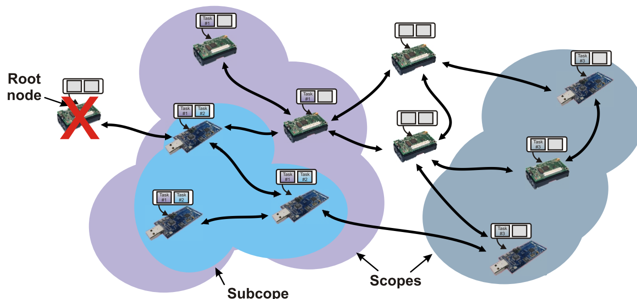
- **but...**

Problem Description

Dependence on Root Node

TECHNISCHE UNIVERSITÄT DARMSTADT

- Root node plays important role:
  - executes the *refresh mechanism*
    - if root node dies, *there is no point in keeping scope alive*

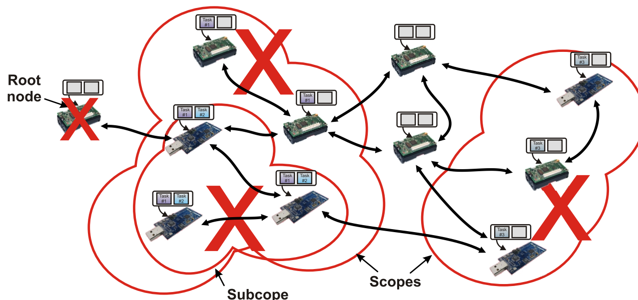


- but...

Problem Description

Dependence on Root Node

- Root node plays important role:
  - executes the *refresh mechanism*
    - if root node dies, *there is no point in keeping scope alive*



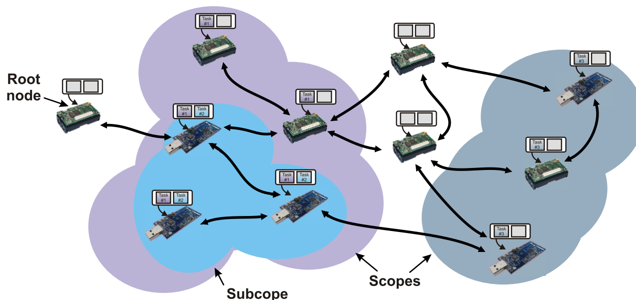
- **but...**

Alternative Behavior

Autonomous Scope Maintenance

TECHNISCHE UNIVERSITÄT DARMSTADT

- In some cases, **autonomous** maintenance is desired
  - long(er)-living scopes, e.g. building automation, railways
    - deploy once somewhere, keep alive until explicit removal

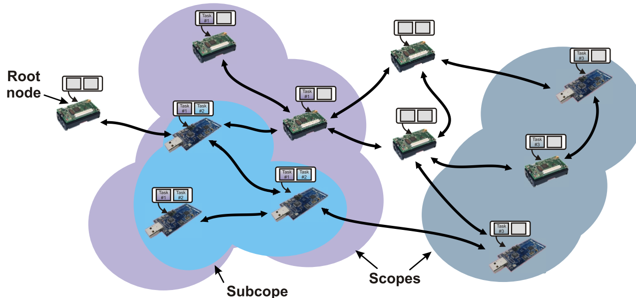


Alternative Behavior

Autonomous Scope Maintenance

TECHNISCHE UNIVERSITÄT DARMSTADT

- In some cases, **autonomous** maintenance is desired
  - long(er)-living scopes, e.g. building automation, railways
    - deploy once somewhere, keep alive until explicit removal



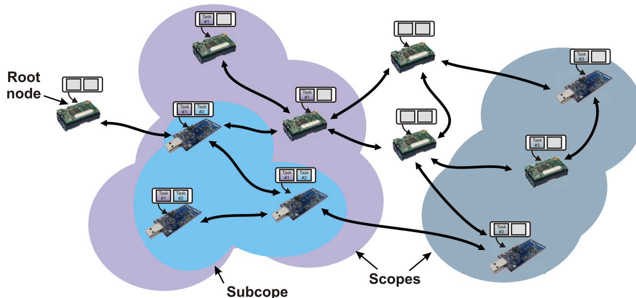
DVS

Alternative Behavior

Autonomous Scope Maintenance

TECHNISCHE UNIVERSITÄT DARMSTADT

- In some cases, **autonomous** maintenance is desired
  - long(er)-living scopes, e.g. building automation, railways
    - deploy once somewhere, keep alive until explicit removal



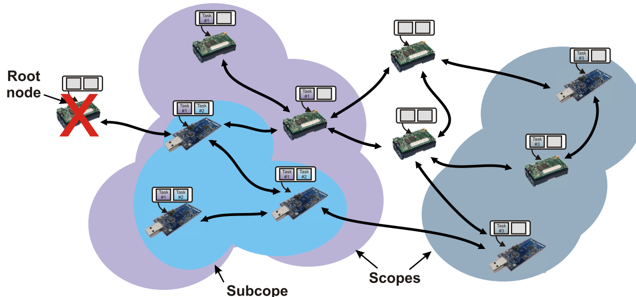
DVS

Alternative Behavior

Autonomous Scope Maintenance

TECHNISCHE UNIVERSITÄT DARMSTADT

- In some cases, **autonomous** maintenance is desired
  - long(er)-living scopes, e.g. building automation, railways
    - deploy once somewhere, keep alive until explicit removal

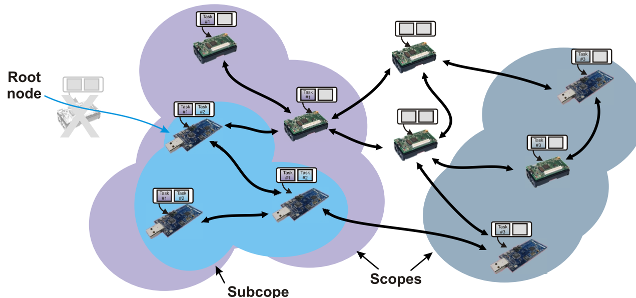


Alternative Behavior

Autonomous Scope Maintenance

TECHNISCHE UNIVERSITÄT DARMSTADT

- In some cases, **autonomous** maintenance is desired
  - long(er)-living scopes, e.g. building automation, railways
    - deploy once somewhere, keep alive until explicit removal



Approach: Floating Managers

TECHNISCHE UNIVERSITÄT DARMSTADT

- Decouple injecting node from root node's responsibility $\rightarrow$ *floating* manager
- Under certain conditions, floating mgr's responsibility moves to another node
- Goal is:
  - enhance fault-tolerance by detecting manager's absence
  - extend network lifetime by moving manager's role around
- Challenges:
  - from *distributed* to *autonomous* system, where any node can contact a scope's manager (e.g., to update/remove it)
  - in/out *traffic* cost monitoring to decide best placement of floating manager

Approach: Floating Managers

TECHNISCHE UNIVERSITÄT DARMSTADT

- Decouple injecting node from root node's responsibility $\longrightarrow$ *floating* manager
- Under certain conditions, floating mgr's responsibility moves to another node
- Goal is:
  - **enhance fault-tolerance by detecting manager's absence**
  - extend network lifetime by moving manager's role around
- Challenges:
  - **from *distributed* to *autonomous* system, where any node can contact a scope's manager (e.g., to update/remove it)**
  - in/out *traffic* cost monitoring to decide best placement of floating manager
- Focus of this work

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Scopes

Problem Description

Proposed Solution

Implementation Details

Results - Conclusions

Proposed Solution

TECHNISCHE UNIVERSITÄT DARMSTADT

This problem can be solved in two steps:

1. Detection of root node failure
   - *Proactive* → send msg's to actively detect failure
   - *Reactive* → wait till an event happens to react
2. Selection of new root → floating manager
   - Leader Election Algorithm
     - Bully algorithm
     - Ring algorithm

Proposed Solution

TECHNISCHE UNIVERSITÄT DARMSTADT

This problem can be solved in two steps:

1. Detection of root node failure
   - *Proactive* → send msg's to actively detect failure
   - *Reactive* → wait till an event happens to react
2. Selection of new root → floating manager
   - Leader Election Algorithm
     - Bully algorithm
     - Ring algorithm

Proposed Solution

This problem can be solved in two steps:

1. Detection of root node failure
   - *Proactive* → send msg's to actively detect failure
   - *Reactive* → wait till an event happens to react
2. Selection of new root → floating manager
   - Leader Election Algorithm
     - Bully algorithm
     - Ring algorithm

Detection of Root Node Failure

TECHNISCHE UNIVERSITÄT DARMSTADT

- A scope's root node sends a *"scope refresh msg"* every x seconds
  - if there is no activity in a scope after a determined *lease time*, node just removes that scope from its table
- To detect root failure, the same **reactive** mechanism is used
  - when the lease time of a scope expires, node reacts reporting a **root failure**

Detection of Root Node Failure

TECHNISCHE UNIVERSITÄT DARMSTADT

- A scope's root node sends a *"scope refresh msg"* every x seconds
  - if there is no activity in a scope after a determined *lease time*, node just removes that scope from its table
- To detect root failure, the same **reactive** mechanism is used
  - when the lease time of a scope expires, node reacts reporting a **root failure**

Selection of New Floating Manager

Leader Election Algorithm

- Idea: enforce *toughest* node to reign
- Based on The *Bully* Algorithm
  - Election mechanism is based on **Priorities**
- Algorithm assumptions:
  - each node has a unique priority
  - nodes are *strongly* connected to each other
  - no packet loss, every message is delivered

Leader Election Algorithm

Protocol overview

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

Protocol overview

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

Protocol overview

TECHNISCHE UNIVERSITÄT DARMSTADT

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

Protocol overview

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

Protocol overview

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

Protocol overview

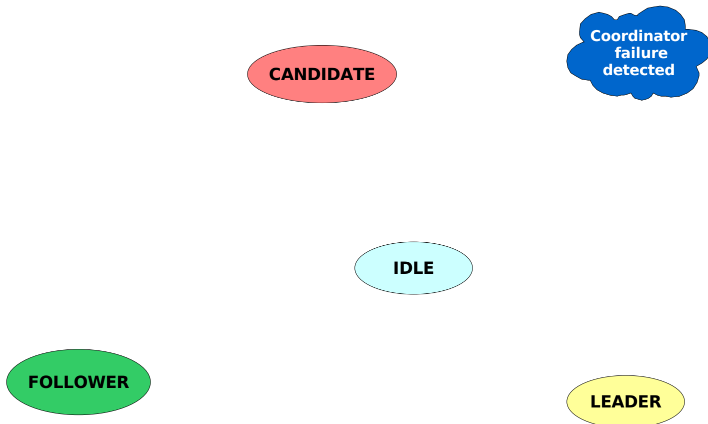
TECHNISCHE UNIVERSITÄT DARMSTADT

- When a node n_i realizes the coordinator has failed, it tries to elect itself as new coordinator:
  - Node n_i sends an *election* message to every other node reporting its priority, becomes a coordinator candidate and waits for time T
- When a node n_j receives an *election* message from n_i
  - if $(priority(n_j) > priority(n_i))$
    $\rightarrow$ Node n_j sends a new *election* message to every other node reporting its priority, becomes coordinator candidate and waits for time T
  - if $(priority(n_j) < priority(n_i))$
    $\rightarrow$ Node n_j goes to follower state and waits for a time T
- The node that completes its election algorithm as a *coordinator candidate* is elected as the new coordinator

Leader Election Algorithm

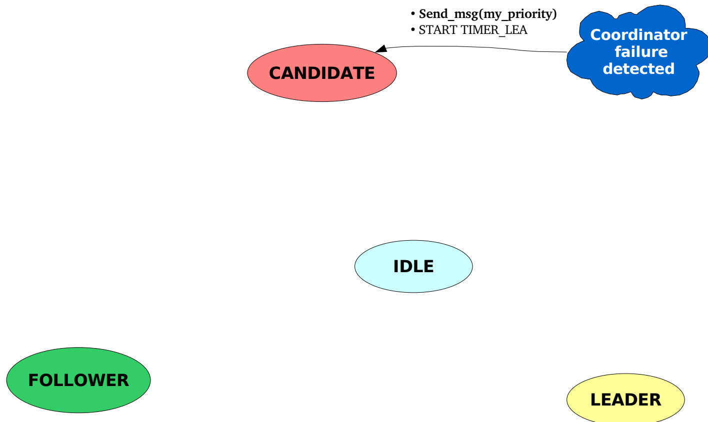
Node State Transition Diagram

TECHNISCHE UNIVERSITÄT DARMSTADT



Leader Election Algorithm

Node State Transition Diagram



Leader Election Algorithm

Node State Transition Diagram

- **Send_msg(my_priority)**
- START TIMER_LEA

Coordinator failure detected

CANDIDATE

- **Rec_msg(priority)** & priority < my_priority
- **Send_msg(my_priority)**
- START TIMER_LEA

IDLE

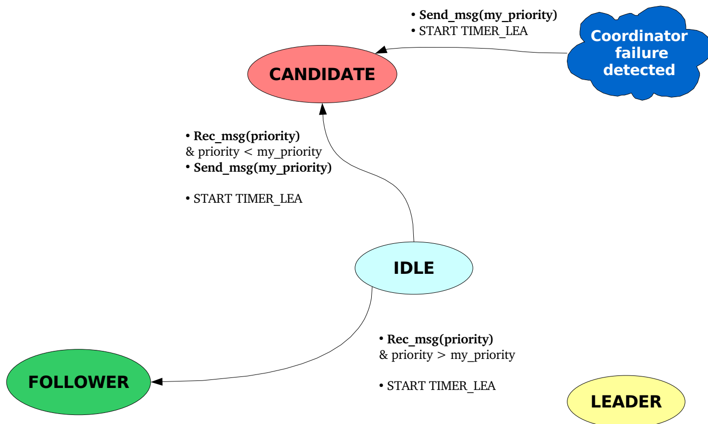
FOLLOWER

LEADER

Leader Election Algorithm

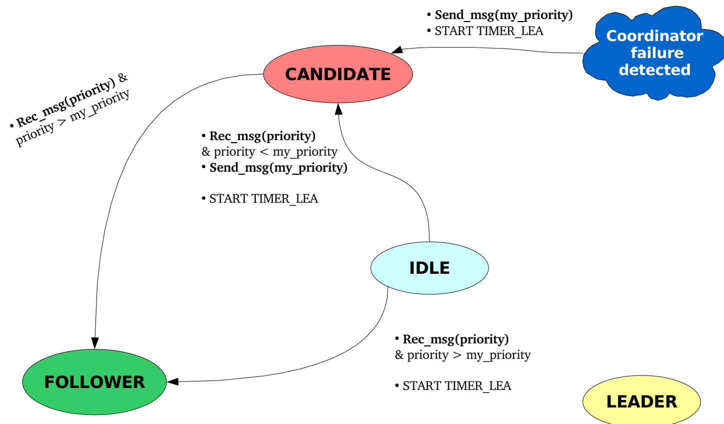
Node State Transition Diagram

TECHNISCHE UNIVERSITÄT DARMSTADT



Leader Election Algorithm

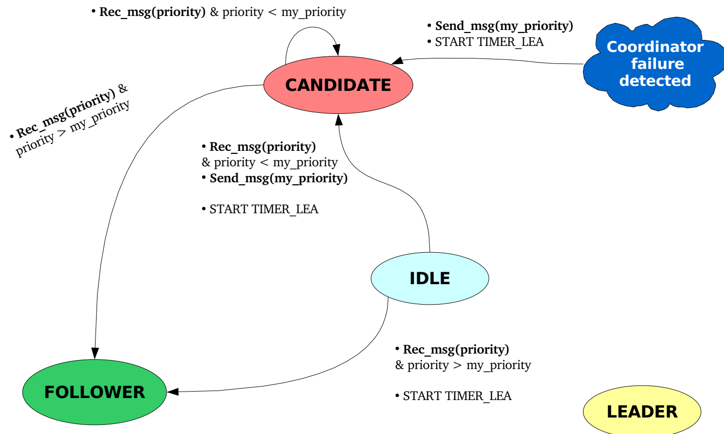
Node State Transition Diagram



Leader Election Algorithm

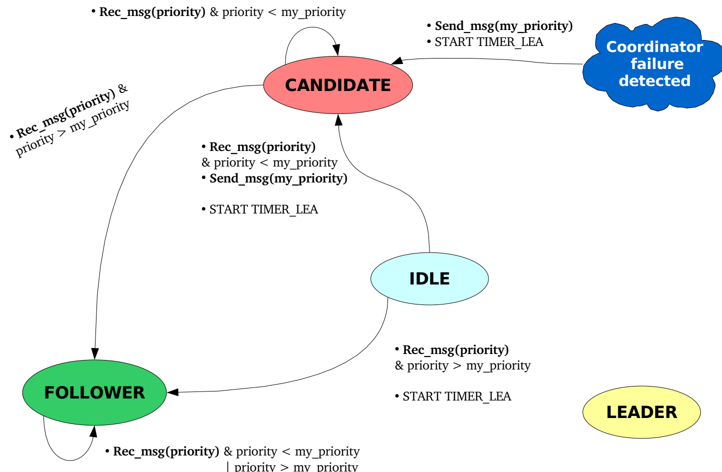
Node State Transition Diagram

TECHNISCHE UNIVERSITÄT DARMSTADT



Leader Election Algorithm

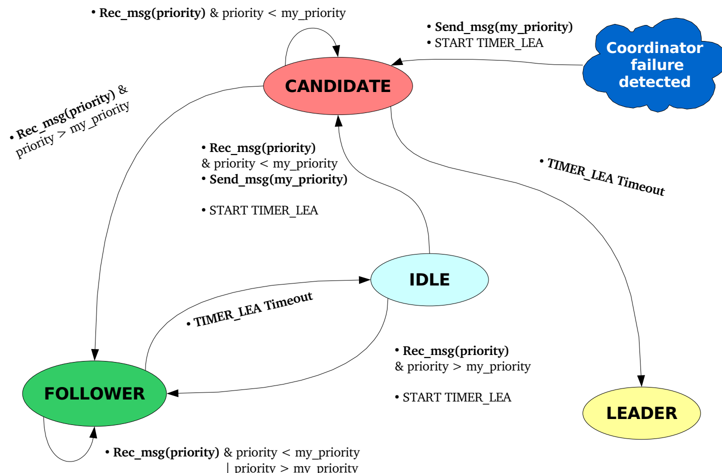
Node State Transition Diagram



Leader Election Algorithm

Node State Transition Diagram

TECHNISCHE UNIVERSITÄT DARMSTADT



DVS

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Scopes

Problem Description

Proposed Solution

Implementation Details

Results - Conclusions

Implementation Details

Leader Election Algorithm for Scopes

TECHNISCHE UNIVERSITÄT DARMSTADT

- There is one *LEA* for each scope whose root node failed
- Node ID's are used as priority
- Nodes manage an extra table:
  - *Scope LEA Table* (dynamic) → LEAs running information
- Nodes manage at most MAX_SCOPES scopes → nodes participate in at least MAX_SCOPES Leader Election Algorithms.

Implementation Details

Leader Election Algorithm for Scopes

TECHNISCHE UNIVERSITÄT DARMSTADT

- There is one *LEA* for each scope whose root node failed
- Node ID's are used as priority
- Nodes manage an extra table:
  - *Scope LEA Table* (dynamic) → LEAs running information
- Nodes manage at most MAX_SCOPES scopes → nodes participate in at least MAX_SCOPES Leader Election Algorithms.

Implementation Details

Leader Election Algorithm for Scopes

TECHNISCHE UNIVERSITÄT DARMSTADT

- There is one *LEA* for each scope whose root node failed
- ~~Node ID's are used as priority~~
- Nodes manage an extra table:
  - *Scope LEA Table* (dynamic) → LEAs running information
- Nodes manage at most MAX_SCOPES scopes → nodes participate in at least MAX_SCOPES Leader Election Algorithms.

Implementation Details

Leader Election Algorithm for Scopes

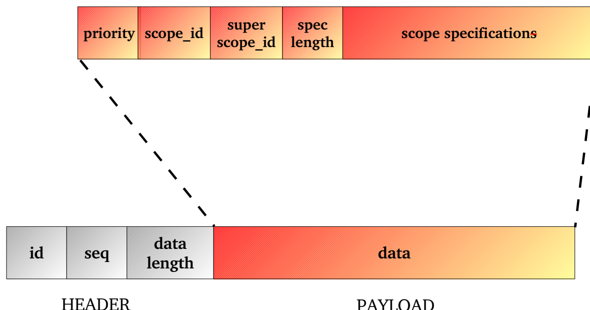
TECHNISCHE UNIVERSITÄT DARMSTADT

- There is one *LEA* for each scope whose root node failed
- Node ID's are used as priority
- Nodes manage an extra table:
  - *Scope LEA Table* (dynamic) → LEAs running information
- Nodes manage at most MAX_SCOPES scopes → nodes participate in at least MAX_SCOPES Leader Election Algorithms.

Implementation Details

LEA Messages

- There is simply one message type: **LEA msg**



- When a node sends a LEA msg, the algorithm assumes that it is delivered to all nodes in the network.
- LEA msgs are Flooded

Implementation Details

Receiving a LEA message

TECHNISCHE UNIVERSITÄT DARMSTADT

- Routing module

  - Flooding
    - msg is new
      → report Scope module
      → resend msg
    - msg has been seen
      → discard
- Scope module

  - Is there a LEA running for this scope ID ?
    - node has already started the LEA (root failure detected)
    - node has already received a LEA message for this scope

Implementation Details

Receiving a LEA message

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Routing module

  - Flooding
    - msg is new
      $\longrightarrow$ report Scope module
      $\longrightarrow$ resend msg
    - msg has been seen
      $\longrightarrow$ discard
- Scope module

  - Is there a LEA running for this scope ID ?
    - node has already started the LEA (root failure detected)
    - node has already received a LEA message for this scope

TECHNISCHE UNIVERSITÄT DARMSTADT

Implementation Details

Receiving a LEA message (2)

- No LEA running for this scope ID:
  - my_priority > priority(msg)
    - Store scope data into Scope LEA Table
      → go to CANDIDATE state
      → Lowest non-used timer ID
    - Send a LEA msg reporting *my_priority*
    - Start Timer LEA
  - my_priority < priority(msg)
    - Store scope data into Scope LEA Table
      → go to FOLLOWER state
      → Lowest non-used timer ID
    - Start Timer LEA
- LEA already running for this scope ID:
  - my_priority < priority(msg)
    - if node was in CANDIDATE state, it goes to FOLLOWER state for that LEA.

Implementation Details

Receiving a LEA message (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- No LEA running for this scope ID:
  - my_priority > priority(msg)
    - Store scope data into Scope LEA Table
      → go to CANDIDATE state
      → Lowest non-used timer ID
    - Send a LEA msg reporting *my_priority*
    - Start Timer LEA
  - my_priority < priority(msg)
    - Store scope data into Scope LEA Table
      → go to FOLLOWER state
      → Lowest non-used timer ID
    - Start Timer LEA
- LEA already running for this scope ID:
  - my_priority < priority(msg)
    - if node was in CANDIDATE state, it goes to FOLLOWER state for that LEA.

Implementation Details

Receiving a LEA message (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- No LEA running for this scope ID:
  - my_priority > priority(msg)
    - Store scope data into Scope LEA Table
      $\longrightarrow$ go to CANDIDATE state
      $\longrightarrow$ Lowest non-used timer ID
    - Send a LEA msg reporting *my_priority*
    - Start Timer LEA
  - my_priority < priority(msg)
    - Store scope data into Scope LEA Table
      $\longrightarrow$ go to FOLLOWER state
      $\longrightarrow$ Lowest non-used timer ID
    - Start Timer LEA
- LEA already running for this scope ID:
  - my_priority < priority(msg)
    - if node was in CANDIDATE state, it goes to FOLLOWER state for that LEA.

Implementation Details

Receiving a LEA message (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- No LEA running for this scope ID:
  - my_priority > priority(msg)
    - Store scope data into Scope LEA Table
      → go to CANDIDATE state
      → Lowest non-used timer ID
    - Send a LEA msg reporting *my_priority*
    - Start Timer LEA
  - my_priority < priority(msg)
    - Store scope data into Scope LEA Table
      → go to FOLLOWER state
      → Lowest non-used timer ID
    - Start Timer LEA
- LEA already running for this scope ID:
  - my_priority < priority(msg)
    - if node was in CANDIDATE state, it goes to FOLLOWER state for that LEA.

Implementation Details

Timer LEA Timeout

TECHNISCHE UNIVERSITÄT DARMSTADT

Scope LEA Table

LEA #1	LEA #2	...	LEA #n
scope_id			
⋮		...	
timer_id			
state			

Implementation Details

Timer LEA Timeout

Scope LEA Table

LEA #1	LEA #2	. . .	LEA #n
scope_id			
⋮			
timer_id			
state			

Implementation Details

Timer LEA Timeout

TECHNISCHE UNIVERSITÄT DARMSTADT

Scope LEA Table

LEA #1	LEA #2		LEA #n
scope_id			
⋮			
timer_id		. . .	
state			

Implementation Details

Timer LEA Timeout

Scope LEA Table

LEA #1	LEA #2		LEA #n
scope_id			
. . .			
timer_id		. . .	
state			

- State:
  - CANDIDATE
    $\longrightarrow$ node is elected *new scope manager*
    $\longrightarrow$ starts sending refresh msg
  - FOLLOWER
    $\longrightarrow$ node finishes LEA for that scope

Implementation Details

Timer LEA Timeout

Scope LEA Table

LEA #1	LEA #2		LEA #n
scope_id			
⋮			
timer_id		. . .	
state			

- State:
  - CANDIDATE
    → node is elected *new scope manager*
    → starts sending refresh msg
  - FOLLOWER
    → node finishes LEA for that scope

Implementation Details

Timer LEA Timeout

TECHNISCHE UNIVERSITÄT DARMSTADT

Scope LEA Table

LEA #1	LEA #2		LEA #n
scope_id			
⋮			
timer_id		. . .	
state			

- State:
  - CANDIDATE
    → node is elected *new scope manager*
    → starts sending refresh msg
  - FOLLOWER
    → node finishes LEA for that scope

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

Scopes

Problem Description

Proposed Solution

Implementation Details

Results - Conclusions

Results

- Different priority assignments can be used: energy, distance, traffic, etc.
- Although the algorithm is simple, its implementation requires many lines of code ($\simeq$ 600)
- Program space requirements were increased only ~~in~~ 1.3 KB

Scopes

section	size	addr
.text	44612	16384
.data	342	4352
.bss	5699	4694
.noinit	25	10393
.vectors	32	65504
.stab	2940	0
.stabstr	2572	0
Total	**56222**	

Scopes w/ floating manager

section	size	addr
.text	45948	16384
.data	342	4352
.bss	5699	4694
.noinit	25	10393
.vectors	32	65504
.stab	2940	0
.stabstr	2572	0
Total	**57558**	

- Additional memory space required in runtime is:
  size of scope LEA table entry $\times$ ~~*n*~~º *of LEAs running*

Conclusions

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adding floating manager functionality required minor modifications to existing framework
- A reactive root failure detection can be easily implemented, but a pro-active mechanism would require additional messages exchange.

Future Work

- Algorithm makes strong assumptions that require a deep evaluation
  - nodes are strongly connected
  - no packet loss
  - nodes have a unique priority → how to ~~manage~~ priority ties
- Floating manager ~~re-placement~~ under certain conditions to extend network lifetime

Packet Loss Tests

TECHNISCHE UNIVERSITÄT DARMSTADT