Self-stabilizing Algorithms in Wireless Sensor Networks

Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

von

Gerry Siegemund

aus

Eisenach, Deutschland

2017

Date of Oral Examination	June 30th, 2017
Chair of Examination Board	Prof. Dr.-Ing. Herbert Werner Institute of Control Systems Hamburg University of Technology
First Examiner	Prof. Dr. Volker Turau Institute of Telematics Hamburg University of Technology
Second Examiner	Prof. Dr.-Ing. Jörg Nolte Distributed Systems / Operating Systems Group Brandenburg University of Technology Cottbus - Senftenberg

Acknowledgment

Thanks to all the people I've got inspired by, in endless brainstorming sessions in front of the coffee maker: Sven Köhler, Laurence Pilard, Johanne Cohen, Stefan Unterschütz, Andreas Weigel, Florian Kauer. Furthermore, thanks to all the people I've worked with directly to "publish and not to perish": Christoph Weyer, Stefan Lohs, Khaled Maâmra. Thanks to my 2nd Supervisor Jörg Nolte who early in the told me: "To master a phd thesis get your sinus rhythm under control, that is all you have to do! Don't get overwhelmed by mishaps or demotivation". I guess I kind of did.

Volker Turau was not very pleased with my work when I've started in the institute. He challenged me to get better. He pushed me. And I think I came out much improved on the other side. Thank you very much for that.

Thanks so much for my friends Julian Ohrt and August Betzler for reminding me that there is always beer if all else fails. I still want to write a paper with each of you, we'll see if we ever get around to doing so. And last and foremost thanks to my wife Katrin, my daugther Amy and my son Luka for always having my back and for all the cheering up over the years. I love you.

Gerry Siegemund

Moisburg, June 2017

Abstract

The presented dissertation focuses on the applicability of self-stabilizing algorithms in systems using wireless communication. Especially wireless sensor networks (WSN) which use low power radios that are prone to message loss and corruption. Furthermore, temporary node failures (e.g., due to exhausted batteries) are common sources of nonconformances. Thus, distributed algorithms, middleware systems, and applications have to respond to these faults. A typical approach is to foresee such error situations and program routines to react to them. Algorithms defined in a self-stabilizing manner (SSA) on the other hand always converge to a defined system state and remain in it while no fault occurs. Hence, the anticipation of error situations is no longer a necessity.

Entities in a distributed system (nodes) share certain informations among their neighborhood (adjacent nodes) and react following the distinct routine of the used SSA. To this day self-stabilization is primarily a theoretical approach, well studied concerning, e.g., the bounds of execution steps. Profound practical evaluation, especially in the presents of rapidly changing neighbor states, as common in WSNs, is still an open issue.

This work firstly establishes necessities to use SSAs in the wireless domain, concluding that a certain degree of forced stability concerning a nodes neighborhood is vital. Nevertheless, such a topology control cannot be rigid, e.g., by using a fixed predefined setup, because node additions or removals cannot be supported. Hence, a topology control algorithm (TCA) is introduced, generating a trade-off between forced stability and agility.

Using this TCA as a cornerstone, multiple SSAs are evaluated, and high level algorithms are developed, culminating in a publish/subscribe middleware defined in a self-stabilizing fashion. The publish/subscribe system relies on a self-stabilizing spanning tree algorithm and a novel self-stabilizing virtual ring algorithm. Furthermore, the publication routing uses shortcuts in the virtual ring, decreasing routing paths in the process.

The presented algorithms are evaluated using simulations employing realistic radio models, as well as implementation on sensor node hardware with low power radios, low computation power, and restricted memory. The novel publish/subscribe system is executable on such limited hardware, uses less messages to deliver data to publishers than a comparable tree-based approach, due to the mentioned shortcuts, and scales well with the network size. It achieves a compromise between the size and maintenance effort for routing tables and the length of routing paths.

Concluding, the dissertation provides an incentive to use self-stabilization algorithms in wireless sensor network applications. As shown, even high level systems like a publish/subscribe middleware can be realized with this inherently fault-tolerant approach.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

From Theoretical Self-stabilization to Fault-tolerant WSN Middlewares

As computer systems interface more and more with every day lives, be it the tracking of mail enabling to monitor packets over the Internet or smart watches that track heart rates and steps taken during the day, we get closer and closer to the vision of the Internet of Things (IoT). Ubiquitous sensing and pervasive computing in Cyber-Physical Systems (CPS) such as the IoT need a communication infrastructure able to handle a multitude of devices while being portable. Wireless Ad hoc NETworks (WANETs) or Wireless Sensor Networks (WSNs) can form such an infrastructure.

A WSN is composed of sensor nodes, these are small devices, in the physical as in the computational sense, with constraint memory. This leads to a very cheap product which can be produced in large quantities. Equipped with various sensors and the ability to wirelessly communicate, the hope is to use networks of sensors for an arbitrary large number of monitoring scenarios. Kahn et al. discuss in their 1999 paper, Smart Dust [KKP99], the vision of a WSN where nodes are only millimeters in size.

On one hand, the networking ability of sensor nodes is their greatest strength, while the maintenance of a network of such error prone devices introduces many issues as well. Back in 2003, Woo et al. stated that *the dynamic and lossy nature of wireless communication poses major challenges to reliable, self-organizing multi-hop networks* [WTC03]. Despite undeniable progress for small scale networks and hundreds

of new MAC and routing protocols, the challenge remains even after thirteen years of intense research.

It is widely agreed that unattended large scale WSNs must self-organize in response to node failures and additions and must adapt to changes in the wireless channel. Mainly because human assistance is not always possible, due to the environment a sensor network may be deployed to, or hands-on maintenance is simply to inefficient or costly. Improving fault tolerance traditionally focuses on fault masking approaches. Fault-scenarios are predefined and handler routines react in the defined way.

Self-stabilization is one particular variant of self-organization, it belongs to the category of non-masking approaches. Instead of modeling individual errors that may occur and providing corresponding recovery routines, self-stabilizing systems are based on a description of the error-free system and rules to reach and maintain this state. Thus, self-stabilization does not handle individual failures separately and therefore takes a more comprehensive view on fault tolerance. Lamport even went so far as to call Dijkstra's introduction to self-stabilization [Dij74] *a milestone in the work on fault tolerance*, even though he probably did not have wireless networks, but the concept of self-stabilization itself in mind [Lam85].

A wireless system experiences transient faults, such as message loss, and, consequently, applications must be prepared to handle them. Self-stabilizing algorithms *automatically* correct such faults, as they converge (back) to a defined system state. Nevertheless, engineers in the field of WSNs substantiate their skepticism against the usage of self-stabilizing algorithms (SSAs) with basically two arguments. Firstly, the stabilization property of an SSA is usually proven under models and assumptions that are not applicable when dealing with wireless communication. Secondly, the time spans of non-availability can not be predicted, and their length is potentially unbound. Both objections are valid, although, it is no reason to completely refuse the usage of SSAs. One can still expect that for specific examples the times of non-availability are short enough to be tolerated. The first point has been acknowledged by research, for example by Herman et al. [Her03, HT04]. Still, it has to be pointed out that many SSAs have never been evaluated in real deployments and that most simulations of these algorithms use at best an oversimplified wireless channel model or none at all.

A major shortcoming of many proposed SSAs is that they assume a static network topology represented by an undirected graph, i.e., a fixed node and edge set. If an

edge is synonymous to being able to successfully send and receive a message at any time, then this assumption is certainly invalid. A failed link can be regarded as a temporal fault. But there obviously is a limit for the rate of link failures, as an SSA needs fault free periods to stabilize. Beyond that limit, the system may behave chaotically.

A well known remedy to minimize message loss, are selective neighborhood tables based on link quality estimators, so called neighborhood management protocols or Topology Control Algorithms (TCAs). They reduce the observed neighborhood to potentially stable neighbors only, hence, mitigating transient faults due to message loss.

Even with a powerful TCA in place that generates the ability to use SSAs, most of these SSAs are not designed to enhance a wireless system per se. Use cases for typical SSAs have been proposed but few have actually been implemented and tested in WSNs. For instance, self-stabilizing spanning tree algorithms may be used for routing to collect data, besides that novel application fields for SSAs may be identified to motivate their use in WSNs further.

WANETs are decentralized and provide a well structured data dissemination architecture after deployment. IoT applications require a seamless integration into back office environments by Internet Protocol-based technologies. While the communication stack for IoT is ready up to the network layer, the structure of the data dissemination layer is still an open issue. Implementations relying on network-layer multicast, e.g., [ABM+16], do not provide the needed flexibility.

Due to the sensing capabilities of entities composing the infrastructure, i.e., sensor nodes, event driven and data-centric architectures are ideal for IoT or in a broader sense for CPS. The four basic functions of a CPS are data capture, data transfer, data analysis, and command distribution. The second and the last functions require the many-to-many communication paradigm, e.g., for the dissemination of *exhaust data* [BTT+15]. This refers to data generated as trails of digitized processes. Dynamic forms of the many-to-many communication style are best supported by the publish/subscribe paradigm instead of using request-reply messaging.

Publish/subscribe systems describe a loosely coupled distributed information dissemination middleware. Senders (publishers) distribute their data (publications) to recipients (subscribers) asynchronously and without knowledge concerning the interested entities. A sensor can be understood as publisher while actuators, controllers, and analyzers are subscribers. Subscribers define their interest in topics either to the

message content which is locally filtered, or by a categorization done by the publisher. Such categories are also referred to as channels. Channels can be used to represent conditional communication between entities, resembling interest or disinterest in various topics.

Joining self-stabilization and publish/subscribe can eventually create a novel middleware that is natively fault-tolerant, and applications using this middleware will inherit the unique properties of SSAs. Multiple challenges are to be overcome to reach this goal. Firstly, convincing arguments that self-stabilization is possible in sensor networks need to be presented and necessities for their employment must be derived. Parts of this work have already been published in various articles and conference [ST17, TS17, LNST16b, LNST16a, BLN+15, STW15, STM15, STM14, STW+13, STLN13, SL13, LSNT12]

This Thesis in a Nutshell

Main goal of this work is to show that self-stabilizing algorithms in wireless sensor networks are feasible. Starting from the premises that the execution of SSAs leads to correct results, limits become clear quickly. The necessity for forced stability in the ever changing environment of wireless communication is presented in Chapter 3. Evidently motivating the quality demands on such a system. Therefore, Chapter 5 presents a distributed algorithm that selects a subset of quality neighbor links to minimize fluctuations caused by inferior and fluctuating links (TCA).

With the TCA in place, higher level applications can be created. Figure 1.1 shows a network stack with all necessary entities to complete complex high level tasks. In particular a middleware that provides applications with an interface to utilize a self-stabilizing publish/subscribe system in WSNs. The following chapters of this work are structured bottom up, since top layers do not influence lower layers (collateral composition, Section 2.2.7). Each layer serves a necessary purpose to enable the publish/subscribe middleware.

Spanning tree, virtual ring, and the publish/subscribe message dissemination layer are defined in a self-stabilizing manner. The TCA is self-organizing, it fulfills safety and liveness properties. Each layer of the publish/subscribe system presented in Fig. 1.1 is briefly described in the following and in depth examined in the according section of the dissertation:

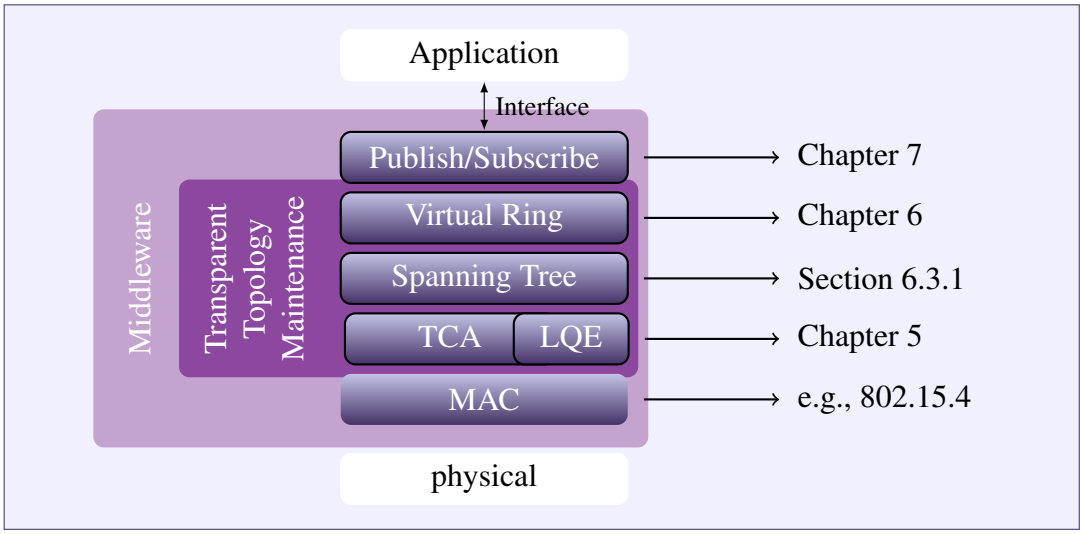


Figure 1.1.: Layered system architecture and reference to related chapters

The media access control (MAC) protocol needs at least three features: Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA), broadcast, and unicast mechanisms to send messages. With unicast a certain reliability is associated, i.e., acknowledgments and a fixed number of retransmissions. The IEEE 802.15.4 standard [MBC+04] incorporates these features and is our choice for all experiments, be it simulation or hardware.

The TCA incorporates and augments two approaches, a basic neighborhood management protocol by Weyer et al. [WUT08] and a modified leader election algorithm [PD02]. For the link quality estimation our TCA resorts to the Holistic Packet Statistics (HoPS) [REWT11]. Chapter 5 focuses on design, implementation, and evaluation of the novel TCA.

Self-stabilizing spanning tree algorithms are quite common. Many different such algorithms with various features have been proposed. We use a version of the algorithm by Huang et al. [HC92]. Each node stores the children it has in the tree which is a necessity for the virtual ring algorithm.

The virtual ring algorithm is an augmentation of a work by Hélary [HR87]. They use a depth-first traversal (DFT) of a tree, where every node visit is recorded with an incremented value, which determines the positions on the

virtual ring. With this approach each node v has as many ring positions as v has neighbors in the spanning tree.

The novel self-stabilizing publish/subscribe middleware builds upon the virtual ring. Publication routing exploits the virtual ring structure and uses shortcuts to decrease the length of routing paths. Achieving a trade-off between memory usage and optimal routing.

The presented thesis structure enables a high level view of the requirements for SSAs in WSNs as well as a detailed view of each subsystem necessary to actually allow the utilization of a complex application. The bottom up approach naturally builds up the network stack and enhances comprehensibility.

Contribution

Firstly, convincing evidence is provided that indicates that self-stabilizing algorithms can be used successfully in wireless networks. This motivates the necessity of a stable neighborhood relation among nodes in the distributed system. Therefore, a novel topology control algorithm (TCA) is presented. It is also shown that without such a TCA self-stabilization does not yield a productive outcome. Furthermore, a middleware is developed that consists of self-stabilizing algorithms only. It describes a novel self-stabilizing publish/subscribe middleware, employing a new virtual ring algorithm and a spanning tree algorithm as its basis. The TCA and the complete publish/subscribe system are evaluated in simulation and on hardware. The work substantiates the claim that SSAs are feasible in WSNs, even for high level applications.

Chapter 2

Problem Domain and Definitions

Fundamentals and Formal Models

In the following the problem domain is be defined, starting from sensors that make up a wireless sensor network to the granularly view of a network stack. Furthermore, the concept of self-stabilization is presented in detail.

2.1. Network of Nodes

Firstly, the physical WSN is presented in the upcoming section. Then two similar but different approaches are introduced to model the described physical system. Table 2.1 gives an overview of the used terminology.

Real World	Model
wireless sensor network	distributed system
sensor node	node/vertices
physical topology	graph
sensor node v received a message from node u	directed edge(v, u)

Table 2.1.: Terminology real world and model

2.1.1. Sensor Node – Wireless Sensor Network

A WSN consist of sensor nodes, which come in a wide variety of shapes and sizes. For example, the current Wikipedia list of sensor nodes [Fou16] accounts for more than 150 different models, meant for a multitude of applications, e.g., area monitoring, forest fire detection, or industrial monitoring. Depending on the mounted sensors, their field of operation is manifold. Even though some computationally *strong* nodes with increased Random Access Memory (RAM) and Read-Only Memory (ROM) exist, e.g., Oracles sensor node!SunSpot running a 180MHz processor clock and 512kB RAM, most devices use *weaker* systems to conserve energy. The MEMSIC TelosB being one famous representative with a processor speed of 8MHz and 10kB RAM.

The sensing capabilities of the employed nodes are not taken into account in this work. That is, none of the presented algorithms depend on sensed data itself, but rather on sent and relayed data. The network is expected to be homogeneous, i.e., only one node type is present during runtime. If the setup is heterogeneous the only necessity is that the same network stack is employed. A higher level of asynchrony, experienced due to heterogeneity is not expected to be of hindrance to any presented algorithm, i.e., the asynchrony is handled by the SSAs. On the network stack a MAC layer including at least CSMA/CA, broadcast, and unicast mechanisms are a requirement. The IEEE 802.15.4 standard [MBC+04] incorporates these features and is used in all implementations. We assume that generated data fits into a single 802.15.4 packet. Furthermore, the memory footprint of all created algorithms must respect the ROM and RAM constraints of current hardware targeting IoT applications (e.g., RAM and ROM less than 64KBytes). We do not assume the existence of a global clock, i.e., there is no common time across all nodes.

General Assumptions Each node carries a network wide unique identifier. Messages are dropped if message queues are full. Messages may arrive in arbitrary order. Some authors make use of geographic information, we make no such assumptions. Moderate dynamics may be experienced, that is, node additions and removals in bounded intervals. Mobility of nodes is excluded from the scope of this work.

When nodes have different roles, e.g., in a publish/subscribe system, we consider the possibility of role changes, while the number of such changes has to be bounded, for a single node, and for the whole network. While a node changes its role certain

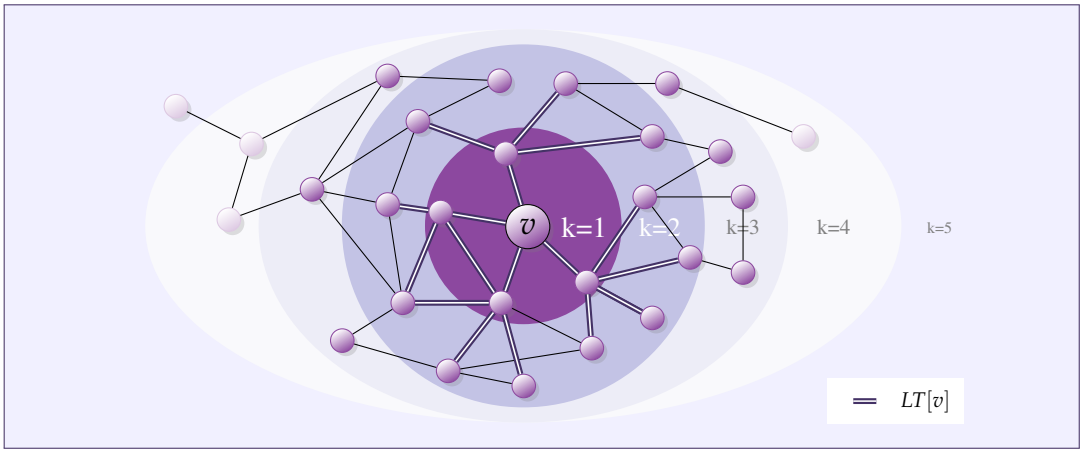


Figure 2.1.: k-hop neighborhood of v, with $k = \{1,\ldots,5\}$, double edges indicate $LT[v]$

limitations one defined properties may be experienced for a limited time. For instance, when becoming a subscriber a certain setup time may pass until the first data message is received.

2.1.2. Distributed System

A WSN is a distributed system. It is commonly modeled as an undirected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, where V denotes the set of nodes and an edge $(v_1, v_2) \in E$ represents a communication link. *Neighbors* are nodes that share a common edge, the set of all neighbors of a node v is referred to as *open neighborhood* $N(v)$ of v. In the *closed neighborhood* $N[v] = N(v) \cup \{v\}$ the node v is included.

The 2-hop neighborhood of v excluding edges between nodes with distance 2 to v, is referred to as the *local topology* $LT[v]$ of v, it includes v. Figure 2.1 shows the difference between the k-hop neighborhood and $LT[v]$.

2.1.3. Distributed System– Dynamic

Modeling a WSN in the previously described way has a major shortcoming. Communication links vary when dealing with wireless communication: this is not represented. One possibility is to assign edge probabilities, making the graph directed in the process. Each $e_p \in E$ has an assigned probability $0 \leq p \leq 1$, defining the chance of successful message delivery.

With the assigned edge probability the overall behavior of a graph over a given timespan can be perceived, but a concrete representation of G at a time t stays unknown. Hence, $G(t)$ is introduced, it is a snapshot of G at time t [RGNH13, LKF05]. To cope with the time dependent nature of the communication links the edge set at time t is denoted by $E(t)$. The set of vertices may not be fixed overtime, hence, V may be time dependent too. A communication graph at time t is therefore represented by a directed graph $G(t) = (V(t), E(t))$, with an edge $e = (v_1, v_2) \in E(t)$ representing a communication link between two vertices $v_1, v_2 \in V(t)$, i.e., at time t a messages can be sent successfully from v_1 to v_2.

Definition 2.1 (Evolving Graph Series [XFJ03])**.** *Given $G = (V, E)$, along with an ordered sequence of corresponding subgraphs $\mathcal{S}_G = G(1), G(2), \ldots, G(t)$, then the system $\mathcal{G} = (G, \mathcal{S}_G)$ is called an* evolving graph series*.*

On $G(t)$ the following metrics are defined: *degree* $deg_t(v) = |N_t(v)|$ of a node v and *maximum degree* $\Delta_t = \max\{\deg_t(v) | v \in V(t)\}$ of all nodes. $deg(v)$ for a node in the graph G over the complete time period T corresponds to: $deg(v) = \max\{deg_t | t \in T\}$, as well as, $\Delta = \max\{\Delta_t | t \in T\}$. The *diameter* D_t is defined as the longest shortest path between any two nodes, hence, $D = \max\{D_t | t \in T\}$. Furthermore, $|V| = \max\{|V(t)| \mid t \in T\}$ and $|E| = \max\{|E(t)| \mid t \in E\}$.

In a directed graph we differentiate between outgoing $\deg_{\rightarrow}(v)$ and incoming node degrees $\deg_{\leftarrow}(v)$. Where $\deg_{\rightarrow}(v)$ are the edges from v to all neighbors while $\deg_{\leftarrow}(v)$ describes all the edges ending at v. Note that in the physical world this responds to successfully send and receive messages, respectively.

Definition 2.2 (Connected Component)**.** *A connected component is a non-empty maximal connected subgraph $CC(t) \subseteq G(t)$.*

It is always time dependent, hence, we use CC and $CC(t)$ interchangeably. The smallest connected component is one node, while the biggest is G.

2.2. Self-Stabilization

fault tolerance is the ability of a system to continue working, or at least to recover from a failure. Two standard approaches to fault tolerance are redundancy and fault handlers designed to react to specific faults. Self-stabilization as a concept to

model distributed algorithms that are inherently fault-tolerant was introduced by Dijkstra [Dij74] in 1974. About 30 years later Herman [Her03] devised the first model to use self-stabilizing algorithms in wireless systems.

2.2.1. Convergence and Closure

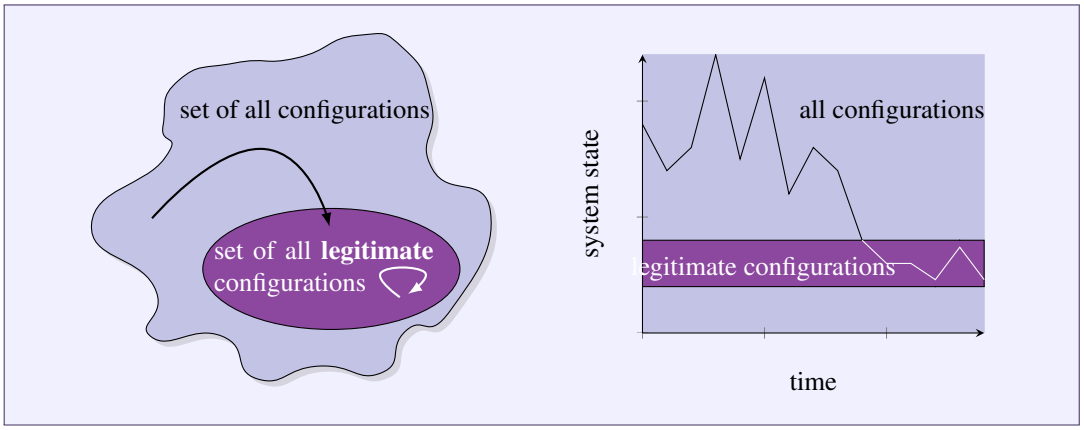


Figure 2.2.: Two equivalent representations of closure and convergence

SSAs are designed to describe legitimate system states. A system converges to a legitimate system state regardless of the initial configuration and persists to be correct while no error occurs. Arora et al. coined the terms convergence and closure for these, progress and safety requirements, respectively [AG93]. Figure 2.2 shows two equivalent representations of the self-stabilization paradigm. From any non-legitimate configuration an algorithm converts to a legitimate one. Eventually in a legitimate state, the system remains in it.

Algorithms designed in a self-stabilizing manner are inherently fault-tolerant, but they provide *non-masking* fault tolerance. While a system is not in a legitimate state, i.e., when it is converging, the behavior of the system is undefined. A node can locally determine that the system is not in a globally legitimate state but the inverse statement is not true. The time the system needs to reach a globally legitimate state is defined by the term *convergence time* or stabilization time.

2.2.2. Faults

Faults that can be handled by self-stabilization are arbitrary, transient state perturbations, hence, Byzantine behavior can only be detected in specialized cases, e.g., [SOM04, DMT15]. Note that some SSAs are defined to withstand Byzantine behavior, e.g., [MT07], but this is not the common case.

On a sensor node the program code and data stored in ROM, e.g., node identifiers, are defined as incorruptible. Otherwise it can not be guaranteed that a devised algorithm is indeed self-stabilizing as its core would be changeable by faults.

2.2.3. Algorithm Definition

A SSA is a distributed algorithm, i.e., the executed protocol is run on each network node. All variables defining a protocol are called a nodes' state. A nodes' state influences the state of nodes in the k-hop neighborhood directly (indirectly other nodes may be influenced too after a state change). k is kept small (usually $k < 3$) to minimize delays inflicted by multi-hop relay, and in case of WSNs due to the restricted message size. The collective set of variables in $N[v]$ for any node $v \in G$ builds v's *local view*. Different models have been defined to describe how nodes share information with their neighborhood.

Sharing the current state among the neighborhood When Dijkstra postulated his work on self-stabilization he considered entities (processors) that where directly connected, hence, sharing information was not considered an issue, therefore the shared memory model was used. In a WSN this model neglects the possibility of message loss, duplication, or corruption, and does not consider arbitrary delays.

As messages are dispatched to share data among nodes in a WSN, commonly the message passing model is used. This implies that nodes cannot (directly) update data at other nodes but only *read* provided (shared via a communication paradigm) data. For more formal definitions of communication models we refer to [Dol00, Chapter 2].

In the physical world, messages may be permuted, duplicated, or lost while being transmitted over the wireless channel. The messages size is considered to be limited. If an algorithm requires to send larger messages, fragmentation is performed. All presented algorithms are designed bearing in mind the development for WSNs, hence, the maximum payload (e.g. 802.15.4 [MBC$^+$04] : 127 bytes) is not exhausted.

Notation Self-stabilizing algorithms are commonly defined as a set of *rules* which are guarded commands in the form

$$guard \rightarrow statement;\ statement;\cdots \quad .$$

This definition can be used interchangeably with (standard) *if* ... *then* statements. Where the *if-part* is the *guard*. The guard of a rule is a Boolean predicate. An algorithm can only *read* variables in the local view. To distinguish which variable *var* is read by the algorithm, if not stated otherwise, $v.var$ defines a read operation on a local variable of node v, while $u.var$ is the shared variable from a neighboring node u.

2.2.4. Time

The execution of a rule is referred to as *move*. If a guard evaluates to true this node is called *enabled*. Between two moves the state of a node changes. Should a subset of enabled instances make a move in parallel this type of execution is referred to as *step*. Finally, a *round* denotes a time frame during which each node that is able to make a move, does so.

In general it holds that the number of $|rounds|$, $|steps|$, and $|moves|$ during an execution follow the following correlation:

$$|round| \leq |step| \leq |move| \ .$$

In a sensor network a move denotes the moment when a node broadcasts its current state to enable neighbors to react to the move (message passing model). If each node broadcasts its current state during a certain time interval, then each of those intervals is the equivalent to one step or one round. It can be though of as a round, since the state is broadcasted even if no data was changed since the previous broadcast, i.e., some nodes may not (actually) have made a move [Her03]. Nevertheless, the notion of a step is more proper, as an enabled node may be delayed and therefore misses the broadcast during a certain time interval.

2.2.5. Scheduler

In a wireless network, the speeds and latencies of communication links vary over time, e.g., depending on the current workload handled by the nodes. Hence, the receipt of message and the reaction to altered data is non-deterministic. To describe the progress of SSAs different execution models have been derived. In the following three of those are mentioned, and it is also made clear why productive sensor networks are best represented by the most general, i.e., least restrictive model.

Either only one entity makes a move, all entities that can make a move execute it at the same time, or the execution can be arbitrary. The latter more formally: a non-empty subset of enabled entities is selected to make a move. These execution semantics are referred to as schedulers (or daemons), in the above order: *central scheduler*, *synchronous scheduler*, and *distributed scheduler*. While the first two schedulers are specializations of the last one.

Schedulers can be further categorized by a notion of fairness. Fairness in this context referrers to the possibility of being selected to make a move while being enabled. An *unfair* scheduler may never select a continuously enabled entity.

The central scheduler works against the spirit of distributed systems since it removes concurrency and it forces global control that does not exist. For WSNs the synchronous and central scheduler can be enforced with certain transformations, applying mutual exclusion, e.g., based on coloring algorithms [KY02] or token passing algorithms. Forcing synchrony or mutual exclusion entails a lot of overhead, and does not scale. Furthermore, nodes may be delayed arbitrarily long, concluding that a WSN is best modeled considering an unfair scheduler.

Note that rounds may take arbitrarily long as the distributed scheduler may delay to select an enabled node for an arbitrary amount of time. Therefore, the term step is better situated to describe the progression of time. Nevertheless, commonly the phrase *a round of broadcasts* is used when referring to WSNs, hence, if not explicitly stated otherwise a step in the execution model corresponds to a round (of broadcasts) in the physical world.

2.2.6. Self-stabilization in Wireless Networks

As already mentioned, enforcing a central or synchronous scheduler on a wireless network generates overhead and does not scale. To execute a SSA, system states are evaluated, rules are executed, and the data has to be shared among the neighborhood.

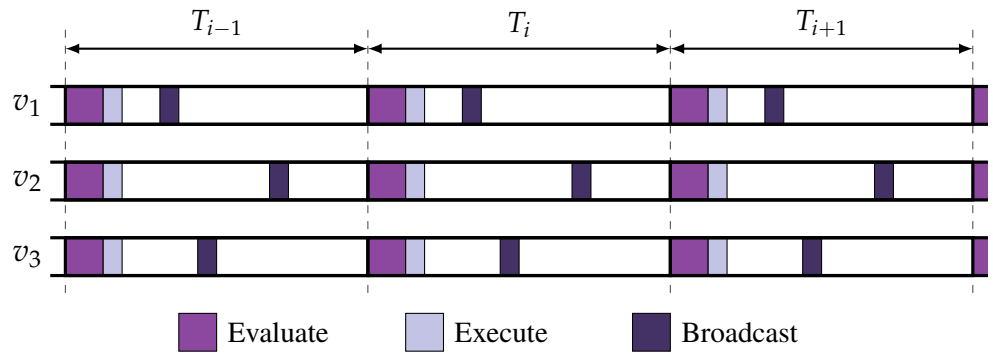


Figure 2.3.: Synchronous execution schedule, $v_{1,2,3}$ are neighbors

Converting a self-stabilizing algorithm stated in an abstract computational model into a program in the sensor network model is referred to as *transformation*.

An example for a synchronous execution schedule is presented in Fig. 2.3. The period length T must be chosen long enough to reduce the probability of collisions, it greatly depends on the network density. As can be seen, each node collects the state information of all neighbors, then executes its enabled rules, if applicable, and then broadcasts its current state variables at a given time slot.

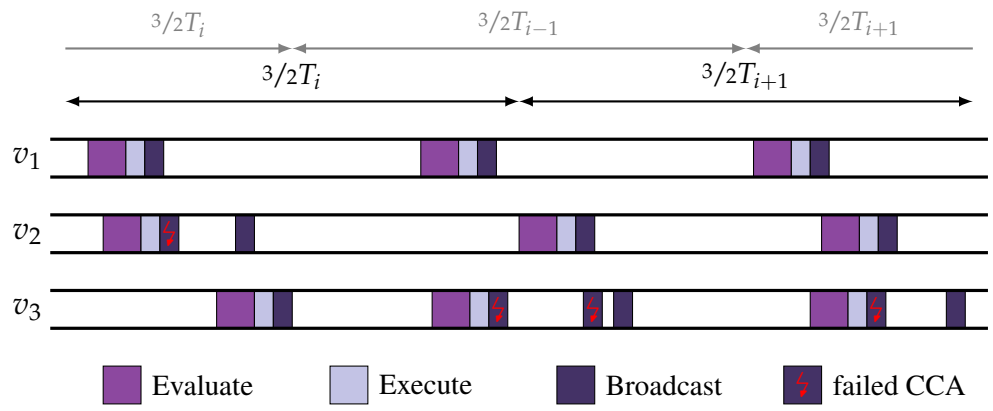


Figure 2.4.: Asynchronous execution schedule, $v_{1,2,3}$ are neighbors

The main problem with the synchronous schedule is that the nodes need to be synchronized, a task that introduces more messages, hence, reduces the channel bandwidth. Leading to prolonged periods T, and to scaling issues.

The first model to use self-stabilizing algorithms in sensor networks is due to Herman [Her03]. He assumes a fixed topology but considers message loss and corrup-

tion. His main contribution is the Cached Sensornet Transformation (CST), where each node maintains a copy of the state of each neighbor, with respect to the fixed topology. Nodes periodically broadcast their state. They only perform an action when an uncorrupted message from each neighbor, since its last action, was received. Under the assumption that each message is received with a fixed probability and that message transmissions are probabilistically independent events the system will eventually reach a legitimate system state with probability 1. A practical application of the CST seems questionable since it is unknown how to set up a fixed topology and furthermore, failed links may block the stabilization process for unknown durations of time.

Yoshida et. al. [YKM08] introduces a similar asynchronous transformation based on periodic timers. Upon packet reception the guards are evaluated and the node performs a move if it is enabled. Periodically the current system state is broadcasted provided that the Clear Channel Assessment (CCA) evaluates to true. To prevent collisions and concurrent execution the period is randomly chosen in the range of $1/2T$ and $3/2T$. Due to the random back-off timer of the underlying CSMA-based MAC layer a randomized convergence is achieved ([TW09]). Figure 2.4 shows a possible execution schedule of the transformation. Throughout this work this transformation is employed if not stated differently.

2.2.7. Collateral Composition

Given two SSAs $\mathcal{A}_1$ and $\mathcal{A}_2$ that are independent of one another considering their variables, then a collateral composition, as introduced in [Her92], is the Algorithm $\mathcal{A}_1 \cup \mathcal{A}_2$ arising from the assignment from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Definition 2.3 (Collateral Composition)**.** *A composition consisting of two self-stabilizing algorithms $\mathcal{A}_1 \cup \mathcal{A}_2$ where the latter may read the variables of the former but not vice versa is called collateral.*

For example, consider a tree algorithm consisting of two algorithms. In this example $\mathcal{A}_1$ is a leader election algorithm, while $\mathcal{A}_2$ builds a tree starting from the defined leader. $\mathcal{A}_2$ reads the *outcome* of $\mathcal{A}_1$, i.e., the currently determined leader. While $\mathcal{A}_1$ never reads a variable of $\mathcal{A}_2$. Compositions are not limited in the number of algorithms that take part.

Chapter 3

Applying Self-stabilization in Wireless Sensor Networks

Demonstration and Limits

This chapter introduces multiple self-stabilizing algorithms. Their implementation is tested in a hybrid approach of simulation and real world deployment as well as directly on actual sensor nodes. In the following the main motivation for the thesis is given as the performance of the presented algorithms is evaluated. Message loss, the main hurdle for the use of self-stabilizing algorithms in wireless networks typically depends on physical phenomena and in such systems each node tries to react to failures in an inherently adaptive fashion by the cyclic observation of its neighbors' states. When the frequency of state changes is too high, the system may never reach a state sufficiently stable for a specific task. The conditions necessary for self-stabilization to lead to fault tolerance in wireless networks are substantiated in the following.

3.1. Overview

In the following light is shed on self-stabilizing algorithms themselves. Further, related work is presented concerning WSNs and SSAs, distinguishing between SSAs

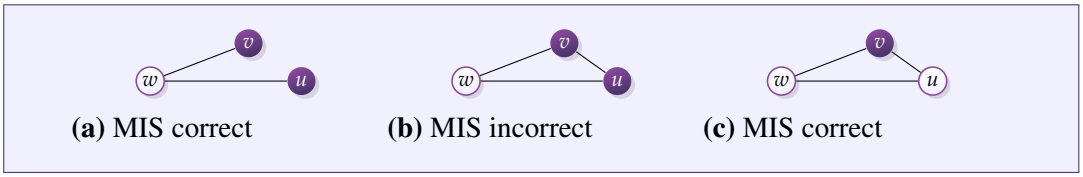


Figure 3.1.: Maximal Independent Set (MIS), dark nodes are *in* the set. Change of undirected edge (v, u) leads to *fault* situation

defined for WSNs and SSAs that have undergone actual testing in WSN simulations or on hardware.

3.1.1. Motivation

Algorithms finding and repairing faults automatically sound like a great idea, as fault scenarios do not have to be identified but *merely* the fault-free states have to be defined. This is easier said than done, since the correct system states can make up a large set. A small number of rules is mostly used to describe a SSA, the more rules the more difficult the proof of convergence and closure. Moreover, most SSAs are proven for a central scheduler as only a single node makes a move at any given time, execution stays in comprehensible order. For the use in WSN, transformations exists to allow the use of SSA proven primarily for central schedulers. The well founded theory of self-stabilization could in theory be valuable for WSN. So far it has mainly been discussed in the field of distributed algorithms [Dol00]. Therefore, this chapter shows limits to the applicability of SSAs facing wireless communication.

The change of neighborhood relations in a graph can be considered a fault in a SSA. Considerer the simple example in Fig. 3.1 describing the current state of a Maximal Independent Set (MIS) algorithm (as defined in Section 3.2.2). (In essence: Each node needs a dominator and two dominators cannot be neighbors.) If an edge emerges between node v and u, then the MIS is faulty (Fig. 3.1b) and has to be reassessed, e.g., in Fig. 3.1c. Hence, a fault in a WSN can be a successfully transmitted message (which does sound counterintuitive). Nevertheless, as messages are usually broadcasted and interference is common, arising and disintegrating *links* in the graph model of WSNs are common, calling for a different view on SSAs in WSNs compared to wired approaches.

3.1.2. Wireless Systems Employing Self-stabilizing Algorithms

Most SSAs are not designed for WSNs, this is obvious since Dijkstra founded the idea of SSAs in 1974 [Dij74] and Herman's model to use them in WSNs was published in 2003 [Her03]. Most algorithms proposed before 2003 use the shared memory model while the message passing model received more attention after Herman's paper.

Algorithms designed to use the message passing model often motivate their work pointing out that the algorithm may be usable in WSN, e.g., [HT04, MFT+05, LS13, BOBBP13]. Nevertheless, the algorithms in the mentioned papers have never been tested in WSNs, and even when simulated, there is no mention of radio models, message loss, or corruption. To show that such algorithms stabilizes, usually a time frame is assumed during which no error is expected, without showing that such a time frame exists. In [BOBBP13] an algorithm is proposed to save energy in WSNs without stating radio models or collision in description of there simulation environment, nor considering evaluation on real hardware in their future work. On the other hand, in the motivation or introduction part of such papers the short-comings of WSNs are usually stated quiet precisely.

Another example is the work by Ba et al. [BFH+13], they evaluate a self-stabilizing clustering protocol using OMNeT++. Proposed topologies consist of up to 1000 nodes and fixed node degrees. The simulation description neither mentions a radio model, nor collisions, or message loss, hence, the simulations do not represent a WSN, even though the authors explicitly proclaim their work to be useful for WSN.

3.1.3. Self-stabilizing Algorithms Tested for WSN

Practical evaluation of self-stabilizing algorithms for wireless sensor networks has not been carried out extensively. Arora et al. deployed a 90 node sensor network in a regular grid to evaluate an intrusion detection system [ADB+04]. This included a self-stabilizing routing protocol Logical Grid Routing Protocol (LGRP) that can tolerate node fail-stop. LGRP is based on geographic information, uses periodic beacons and a heuristic to avoid loops. They observed an effective reliability of less than 50 percent, resulting in poor application performance. They stated that even with self-stabilizing algorithms network unreliabilities cannot be ignored. As a remedy they designed a reliable communication service to improve per-hop and end-to-end reliability.

The work by Yoshida et al. [YKM08] is one of the few works on SSAs that actually has been tested on (five) real sensor nodes. They describe a lightweight transformation based on the mentioned works by Herman [Her03] and Turau [TW09]. They can indicate that transformations work and that algorithms proposed for a central daemon can be used in WSN. Nevertheless, testing with five sensor nodes is merely a starting point in the right direction. The wireless channel with all its peculiarities and the interference produced by many, dense sensor nodes has not been put into perspective.

In 2011 Unterschütz et al. [UT11] proposed a novel algorithm to find connected dominating sets in ad-hoc networks. They test their self-stabilizing approach on a real world testbed consisting of 15 nodes. Furthermore, simulations are conducted using up to 2000 nodes. In the simulation no propagation model was applied and unit-disk graphs where considered. Nevertheless, packet-collisions are considered.

SelfTDMA [LKNL12] uses a minimal spanning tree to setup a Time Division Multiple Access (TDMA) routing scheme. The work is simulated in OMNeT++ but instead modeling the radio channel, a scenario manager is used to induce errors. Their scenario manager is able to inject link breaks and node failures. Furthermore, SelfTDMA is tested on hardware. Nine nodes where placed in different environments, e.g., office, lawn. Scaling and density of the approach cannot be verified by a small number of nodes. In the simulation up to 200 nodes where deployed, while the density was fixed to at most 18.

In [PST14] Petig et al. use COOJA to simulate the TinyOS implementation of their TDMA approach. They mention the implementation for nodes with IEEE 802.15.4 compatible radio transceivers but do not elaborate on simulation properties like message loss, interference, or collision. Hence, it remains unclear if a radio-model was applied.

Kulkarni et al. implemented and deployed a self-stabilizing TDMA in a 10×10 communication grid with MICA-2 sensor nodes [KA06]. The experiments assumed knowledge about local neighborhood and required time synchronization. The TDMA protocol is used to transfer algorithms written in the shared-memory model to the more realistic write all with collision (WAC) model. Details about the course of the stabilization process were not reported.

Choi et al. evaluated a self-stabilizing grid routing protocol that maintains an incoming spanning tree rooted at the base station to route data messages from any sensor to the base station [CGZA06]. The experimental results showed that the protocol delivers 72-99 percent of data messages to the base station under bursty and heavy

traffic. To achieve this delivery rate they performed off-line experiments to estimate link quality so that during the experiment reliable links were known. Their protocol limits the connectivity of the sensors in a network such that some sensor nodes do not find a path to the base station, even when such a path exists.

To the best of our knowledge no comprehensive analysis of self-stabilizing algorithm in real wireless networks has been conducted thus far.

3.2. Methodology

A wide range of experiments has been conducted to give a proof of concept that SSAs can be used in WSN. To enable comparison of these results multiple metrics are introduced. Furthermore, the hybrid approach mentioned in the introduction of the chapter is explained and justified. The selected algorithms are clarified and the applied communication model for the SSAs is declared.

3.2.1. General Topology Metrics

Two metrics have been identified to compare different topologies, represented as graphs. They are independent of the employed SSAs. In the following they are used to put the gathered communication traces into perspective.

i. Connectedness

A prime criteria for the quality of a communication network is its connectedness. Failing links can lead to situations where the network is split-up into separate connected components. To measure the degree of network decay, the connectedness metric is introduced. The *connectedness* of a graph G is defined as the quotient of the number of nodes in the largest connected component and the total number of nodes n. E.g., the connectedness equals 1 if and only if all nodes can communicate pairwise via multi-hop routing or a connectedness of 0.5 means that the largest connected component comprises of half the nodes in G.

ii. Similarity

The *similarity* of G is the difference between two consecutive graphs $G(t)$ and $G(t+1)$. It can be computed by the scalar product of a vector representation of the two

graphs divided by the product of the norms of the vectors as defined by Birand et al. [BZZL11]. Hence, the similarity is based on the number of common edges of two graphs. E.g., a similarity of 0 implies that two graphs have no edge in common while a similarity of 1 means that both graphs are equal.

With the above defined metrics it is possible to evaluate an evolving graph series and to illustrate their behavior over time. This allows to grasp the performance of a particular self-stabilizing algorithm more precisely and over the course of each round.

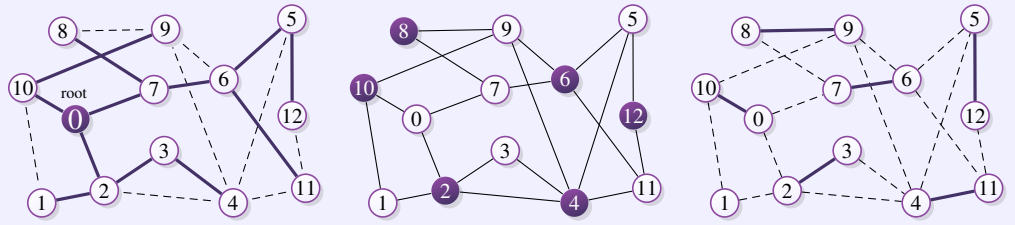
3.2.2. Self-stabilizing Algorithms and Correctness

When motivating SSAs developers usually mention their robustness to faults, they may even propose the possibility to use message passing models and that transient memory errors can be tolerated. Moreover, the typical model assumes precisely scheduled execution, atomic operations, and mutual exclusion. When dealing with more diverse environments a series of transformers has been proposed [Dol00]. In order to fairly assess the potential of SSAs for WSNs the contribution of the transformers cannot be ignored.

These transformers themselves must be self-stabilizing, i.e., some SSAs have to be executed directly on the network node and must deal with the dynamic and lossy nature of wireless channel. It is well known that with a probabilistic scheduler even SSAs that make strong assumptions about atomic operations and mutual exclusion will eventually stabilize as was shown by Turau and Weyer [TW09]. The execution environment of a wireless network exhibits a high degree of randomnesses through its MAC layer due to, e.g., CCA and random back-off. We therefore argue that the analysis of the execution of a SSA without one of the mentioned transformers introduce novel perspectives to evaluate the potential of SSAs for WSNs.

Three SSAs are used to gain an insight into the concepts of self-stabilization in the wireless domain. These are a spanning tree algorithm $\mathcal{A}_{TREE}$, a maximal independent set algorithm $\mathcal{A}_{MIS}$, and a matching algorithm $\mathcal{A}_{MATCH}$. Each algorithm fulfills a task commonly performed in the context of network protocol design and each has been studied extensively in the theoretical domain.

A topology consisting of 13 nodes demonstrating a valid execution of each algorithm presented in the following is given in Fig. 3.2. Formal definitions and metrics to determine their correctness are presented for each algorithm individually.



(a) Spanning tree **(b)** MIS **(c)** Maximal matching

Figure 3.2.: Valid result of each employed algorithm

i. $\mathcal{A}_{\text{TREE}}$

The first selected algorithm builds a spanning tree [Dol00] starting at a defined root node. A tree is the typical routing structure for collecting data from multiple source nodes to one gateway (aka, sink node).

Algorithm Description Algorithm 3.1 consists of two rules. The node with identifier 0 is defined as root node, this parameter may be changed, or could also be determined at runtime. Rule R1 is executed by the root node only, it resets the parent variable p and the distance variable d in the fault case. Each other node v determines p from their local neighborhood $N(v)$, as stated in Rule R2.

Algorithm 3.1 Self-stabilizing Spanning Tree

Nodes: v the current node
Variables: $v.p$: Node identifier of parent
$v.d$: Integer stating the distance to the root node
Predicate: $minDist(v) \equiv \min\{u.d : u \in N(v)\}$

$$isRoot(v) \equiv \begin{cases} \text{TRUE} & \text{if } v = 0 \\ \text{FALSE} & \text{otherwise} \end{cases}$$

do
[R1] $isRoot(v) \wedge \neg((v.p = null) \wedge (v.d = 0)) \rightarrow v.p = 0; v.d = 0$ ▷ Reset
[R2] $\square\ \neg isRoot(v) \wedge \neg(v.d = minDist(v) + 1)$
$\rightarrow v.p := argmin\{minDist(v)\}; v.d := minDist(v) + 1$ ▷ Choose Parent
od

Metric In wireless communication, messages are lost frequently. For an algorithm running on network nodes it is usually unimportant that all edges are potentially usable, it is only important that a link is available when a message has to be sent. Hence, a tree can temporarily become disconnected while messages from an arbitrary node still reach the root. Additionally, sending messages over multiple hops decreases the overall chance that they are delivered. Retries are a common remedy to overcome temporary unavailability.

Considering this, Xuan et al. [XFJ03] coined the term *journey*. A route between two nodes only allowing one sending event per cycle is called a journey, if at each time step the according edge on the route can be traversed.

Definition 3.1 (Journey)**.** *A* route $\mathcal{R}(v_1, v_2) = \{e_1, e_2, \ldots, e_k\}$ *is defined, where each* $e_t \in E(t)$*. Let* σ *be a time schedule denoting the edge traversal. A* journey $\mathcal{J}(v_1, v_2, \sigma) = \{\mathcal{R}(v_1, v_2), \sigma\}$ *is then defined if and only if* σ *allows for a traversal from* v_1 *to* v_2 *in* $\mathcal{G}$*.*

Journeys are directed, and only consider current and future events. As a metric, it is checked for each node whether a journey to the root node exists. We allow a single retry for each sending event. The percentage of nodes in a tree connected by a journey allowing *one* retry is referred to as the *journey metric* $m^{\text{TREE}}_{journey}$.

ii. $\mathcal{A}_{\text{MIS}}$

The second representative SSA is a MIS algorithm. It was already mentioned and informally introduced in Section 3.1.1 and in Fig. 3.1. A set S of nodes in G is independent if no two members of S are adjacent. S is maximal if no proper superset of S with the same properties exists. In networks, independent sets can be used to select cluster heads for routing or aggregation purposes.

Algorithm Description Algorithm 3.2 states a well known representative for a MIS selection [SRR95, GHJS03]. Rule R1 adds a node to the set, if none of its neighbors are in it. If a node v that currently is within the set recognizes that the same holds for a neighboring node v, then, due to Rule R2, v leaves the set.

This algorithm fails in strict synchronous setups. To break symmetry, rules are executed with a fixed probability, i.e., a randomized transformation of the SSA, as described in [WTLN09], is applied. Note that there exist MIS algorithms that can

also be used in a synchronous setups [Tur07], but they are more complex. Such a transformation usually leads to longer stabilization times. In hardware experiments the wireless channel ensures randomized execution, as messages can only be sent when the CCA succeeds [YKM08].

Algorithm 3.2 Self-stabilizing Maximal Independent Set

Nodes: v the current node
Variables: $v.s \in \{\text{IN}, \text{OUT}\}$
Predicate: $inNeighbor(v) \equiv \exists u \in N(v) : u.s = \text{IN}$

do
[R1] $v.s = \text{OUT} \wedge \neg inNeighbor(v) \rightarrow v.s := \text{IN}$
[R2] $\square$ $v.s = \text{IN} \wedge inNeighbor(v) \rightarrow v.s := \text{OUT}$
od

Metric A MIS is correct if and only if nodes in S are not neighbors, and if all nodes in $G \setminus S$ have a neighbor that is in S. To obtain the quality value $m^{\text{MIS}}_{correct}$, the number of nodes not violating this definition, i.e., all correct nodes in S, is divided by n.

iii. $\mathcal{A}_{\text{MATCH}}$

Lastly, the maximal matching algorithm is presented. A matching M in $G = (V, E)$ is a subset of E where no pair of edges is adjacent. M is maximal if there exists no $\widehat{M} \subset M$ where $\widehat{M}$ is also a matching. A matching can be used to pair up nodes, e.g., as a client/server structure for backup purposes.

Algorithm Description Algorithm 3.3 [MMPT09] works as follows: each node chooses the maximum neighbor identifier as a candidate for a pairing, as stated in Rule R3. Variable $v.p$ points to the candidate. An unpaired node u that is pointed to by a neighboring node v selects v as a partner (RuleR2). With the aid of predicate $PRmarried(v)$ the existents of a correct partner relation is evaluated. Rule R1 confirms the pairing if both nodes point to each other by setting m to $true$. If a node points to a no longer available candidate, then Rule R4 clears the partner selection.

Algorithm 3.3 Self-stabilizing Maximal Matching

Nodes: v the current node
Variables: $v.m \in \{\text{TRUE}, \text{FALSE}\}$
$v.p \in \{null\} \cup N(v)$
Predicate: $PRmarried(v) \equiv \exists u \in N(v) : (v.p = u \wedge u.p = v)$

do
[R1] $v.m \neq PRmarried(v) \rightarrow v.m := PRmarried(v)$ ▷ Update
[R2] $\square\, v.m = PRmarried(v) \wedge v.p = null \wedge \exists u \in N(v) : u.p = v$
$\rightarrow v.p := u$ ▷ Marriage
[R3] $\square\, v.m = PRmarried(v) \wedge v.p = null \wedge \forall w \in N(v) : w.p \neq v \wedge$
$\exists u \in N(v) : (u.p = null \wedge u > v \wedge \neg u.m)$
$\rightarrow v.p := \max\{u \in N(v) : (u.p = null \wedge u > v \wedge \neg u.m)\}$ ▷ Seduction
[R4] $\square\, v.m = PRmarried(v) \wedge v.p = u \neq null \wedge u.p \neq v \wedge (u.m \vee u \leq v)$
$\rightarrow v.p := null$ ▷ Abandonment
od

Metric To determine the quality of the matching algorithm all nodes that have a correct matching are counted. If there exist unmatched nodes that cannot be matched, than they are counted as correct as well. The $m_{correct}^{\text{MATCH}}$ metric is the quotient of correct nodes and n.

iv. General Algorithm Properties

The correctness metrics $m_{journey}^{\text{TREE}}$, $m_{correct}^{\text{MIS}}$, and $m_{correct}^{\text{MATCH}}$ help to evaluate the usability of SSAs for high level network tasks, e.g., routing and group forming. All presented algorithms are *silent* SSA, hence, it holds that an algorithm is correct when none of the nodes running it have an enabled rule, such a state is also referred to as stable.

The *stability metrics* s^{TREE}, s^{MIS}, and s^{MATCH} are defined for all three representative algorithms. They show the percentage of stable nodes in the system, i.e., the number of nodes with an enabled rule divided by n. The longer this metric stays constant the more promising and useful the algorithm can be expected to be.

3.3. Experimental Setup

Communication traces where gathered to reproducibly simulate the presented algorithms. These traces are examined by the graph metrics for connectedness and sim-

ilarity introduced in Section 3.2.1. Lastly, an understanding of the communication pattern applied for the shared data among the nodes is given.

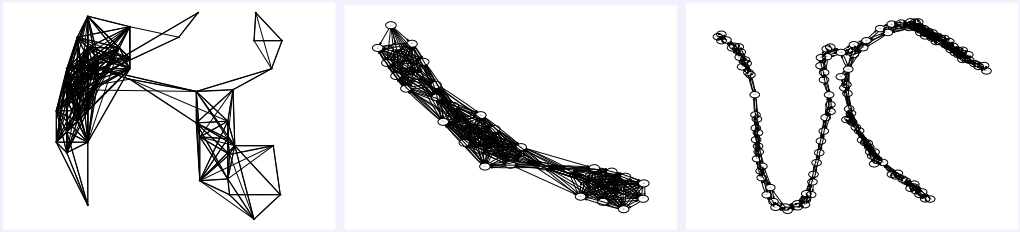
3.3.1. Real World Traces

To evaluate the usability of self-stabilizing algorithms, simulations are insufficient. Even with radio and path-loss models, the sheer countless features of real radio communication in the low power domain can not be reproduced. For instance, some close by nodes might not be able to communicate most of the time, but nodes barely in communication range have a strong connection. Thus, evaluations on real hardware are mandatory. Unfortunately, hardware experiments are non-deterministic, i.e., not exactly reproducible, and therefore comparison among results is delicate.

Furthermore, a deep understanding of the behavior of SSAs requires a global view. Retrieving the necessary information to generate this view from a testbed experiment requires appropriate sensor nodes and additional hardware. For this reason a hybrid approach is employed. Collected communication traces of real hardware deployments replace the radio model in the execution of the SSAs. Enabling the reproducible examination of the behavior of SSAs at any given time.

Communication data at the FIT IoT-LAB [FFH+14], a large WSN test-bed developed in France, was recorded. Herein, various deployments of *WSN430 open nodes* based on a low power MSP430 platform (800 MHz band) and *M3* nodes, with an ARM Cortex (2.4 GHz band), were used. The well known *Contiki* operating system with a *Rime* [DÖH07] communication stack was remotely configurable at the FIT IoT-LAB. Contiki and Rime enable scheduled broadcasts, the only necessity for our measurements. Every 10 seconds plus a random delay between 0 and 5 seconds, each node broadcasted a message including a sequence number and the sender's identifier. Every node that received such a message logged its occurrence with the received sequence number. Of the set of recorded traces two representatives referred to as WSN430 and M3 are used. WSN430 consists of 40 nodes and M3 of 143. During each round of broadcasts each node sends one message.

Additionally, communication traces collected by Ortiz et al. [OC08], referred to as *Soda*, are employed. Their setup consists of 46 IEEE 802.15.4-compliant TelosB motes which are deployed in an indoor environment. Each node transmits 100 packages every 20 milliseconds. Meanwhile, all nodes that receive a message record the



(a) Soda **(b)** WSN430 **(c)** M3

Figure 3.3.: Topology of collected traces, links with PRR above 70 percent

received sequence number. There are 17 repetitions over all 16 frequencies leading to 27200 generations of received or missed messages.

Each logs from each measurement (WSN430, M3, and Soda) is used to build one adjacency matrix per sequence number (i.e., round). Hence, evolving graph series are obtained. Each of these graphs $G(t)$ is used as a network topology to evaluate the SSAs. Whether a link is present during a round is therefore predetermined by the discrete adjacency matrices. In our measurements, we reproduce the exchange of states between neighbors in a real application. Notably, this includes *collisions* and *interference*. In the case of the Soda trace, the exchange was sequential, which eliminates this radio communication characteristic, but not other environmental influences, e.g., obstruction or weather conditions.

3.3.2. Characteristics of Topology Traces

Before evaluating the behavior of the SSAs on the gathered data, the network topologies themselves are put into perspective. Figure 3.3 shows the topologies of the chosen traces as directed graphs. Communication links with a Packet Reception Rate (PRR) of less than 70 percent are omitted. Firstly, because including those links would lead to complete clutter and secondly because later low quality links will be filtered. Since node coordinates were available for the Soda trace, the plotted graph is accurate. In case of the WSN430 and the M3 traces a graph drawing tool generated the positions, as they are not specified. It can be observed that all traces are different in density, diameter, and size. The diversity of the chosen traces ensures that our findings are not based on a specific topology.

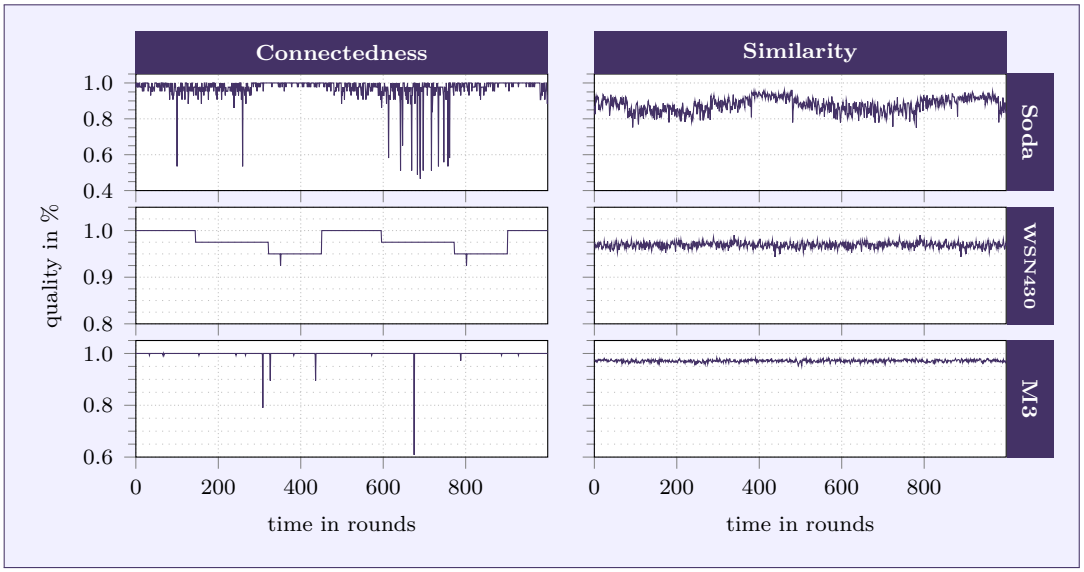


Figure 3.4.: Comparison of topology traces Soda, WSN430, and M3

Figure 3.4 shows the connectedness and the similarity for all traces. Temporary network separations are common when dealing with wireless communication. As can be verified, all used traces are connected most of the time. The Soda trace changes considerably, it is divided into two clusters connected by a few inter cluster links. Those links have a PRR between 70 and 80 percent. Leading to a high probability of network separation, which has to be considered in the evaluation.

The similarity metric is normalized over all possible edges, hence, graphs with many nodes will generate fluctuations in a smaller visible margin. With other words if a small amount of edge-changes occurs in a graph with few nodes the same disturbance will hardly be noticeable in a graph with many nodes. Thus, comparing the similarity of the Soda and the M3 trace is delicate in Fig. 3.4. Important facts are that the M3 topology, even though it is mostly connected, constantly experiences similarity fluctuations, i.e., message loss. Secondly, the Soda and the WSN430 trace can be compared as their number of nodes is roughly equal and the Soda trace changes significantly more over time.

3.3.3. Communication Scheme

For the execution of the SSAs they are integrated into a middleware layer for embedded wireless systems. Its task is to control the neighbor state exchange, the caching

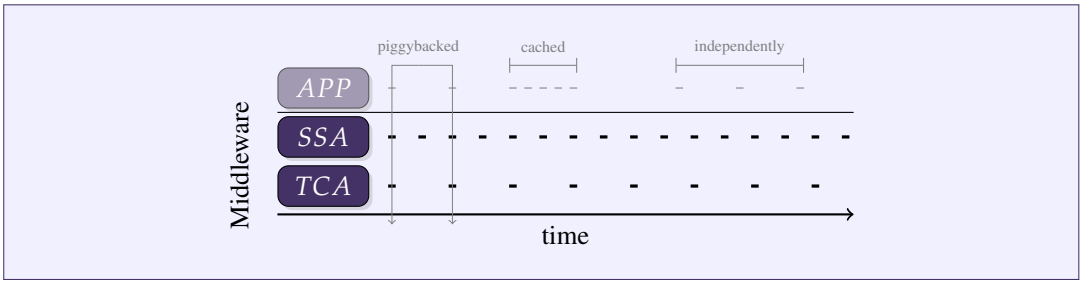


Figure 3.5.: Example frequency setting of the middleware

of received states [Her03], and the evaluation and execution of the rules [TW09]. The layer has a cycle based execution driven by the frequency of the state exchange f_{SSA} and the frequency of the neighbor exploration f_{TCA}. Figure 3.5 shows example frequencies of all components. In general, the resulting communication frequency f_{OBS} of the middleware has to be at least the maximum of both frequencies f_{SSA} and f_{TCA}. f_{OBS} describes how often the local state is exchanged, i.e., how often changes of the local neighbors can be observed.

f_{TCA} dependents on the rate of topology changes, and thus on environmental conditions. The convergence time of the selected SSAs influences the choice of f_{SSA}. In general, the frequency of the SSA should be higher than or equal to the neighbor exploration rate to respond to changes in the topology. Although the frequencies can be set differently, we assume in our evaluations that the frequencies are identical and constant, i.e., $f_{SSA} = f_{TCA}$. Hence, the middleware is able to aggregate both message parts, i.e., local state and neighbor list. f_{OBS} declares the broadcast timing for the communication step. In further evaluations, the cycle frequency f_{OBS} is set identical to the cycle of our obtained traces.

i. Space Requirements

Each applied SSA shares its local view, i.e., all variables stated in the pseudo code. Even if 4 Bytes would be needed per variable, the space requirement for message and local storage are small, e.g., 8 Bytes for $\mathcal{A}_{TREE}$. The TCA on the other hand needs more local storage, and as the stored neighbors have to be shared the necessary memory grows with the node degree. As proposed this is be fixed to C_N. In Chapter 5 this discussion will be continued, while advantages and disadvantages of different values of C_N are debated and evaluated.

Data produced by an application can be piggyback on messages of the middleware layer, if no timing constrains forbid this. If this is not possible due to the application behavior, the messages can be cached for later transmission. A further possibility is to send the data independently between two transmissions from the middleware. Each case is depicted in Fig. 3.5. With a piggybacked scheme, the channel can be relieved and the overall bandwidth can be increased. Also making the usage of the SSAs transparent to the network. For the following considerations it is assumed that no additional application messages are sent.

3.4. Directly Applied Self-Stabilization

The execution of self-stabilizing algorithms directly on a wireless network is the first study that is undergone to motivate their usage in real deployments. Each SSA evaluates its rules on the state of each neighboring node at each round defined by the gathered traces. The usability of the execution can be verified in Fig. 3.6.

Depicted is the stability metric s and the quality metric m for each trace. None of the algorithms produce a usable outcome. The tree is never built to the point where all children are able to communicate with the root (even with one retry). The $\mathcal{A}_{MIS}$ and $\mathcal{A}_{MATCH}$ do not yield a stable or correct system state. Constant changes in the topology as seen in Fig. 3.4 Section 3.3.2 cause frequent rule executions, hindering the algorithms to stabilize or to be correct.

To the contrary of the Soda and M3 trace, the $\mathcal{A}_{TREE}$ result $m^{\text{TREE}}_{journey}$ of the WSN430 trace indicates a functional tree. This results mainly from the high density of the topology. Nevertheless, the created tree is hardly suitable for routing purposes as it is to unstable.

In contrast, the stability of the $\mathcal{A}_{MIS}$ stays stable for over 90 percent of the observed time. As a disadvantage, the algorithm fails to establish a suitable minimum independent set as $m^{\text{MIS}}_{correct}$ on average equals 75 percent. The $\mathcal{A}_{MATCH}$ algorithm performs similar on all three traces. The algorithm never stabilizes neither the $m^{\text{MATCH}}_{correct}$ metric reaches a correct state. The Soda trace results are still inferior to the WSN430 and the M3 trace.

The results show that it is not feasible to use SSAs based on a network topology without a certain degree of stability. The algorithms autonomously react to topology changes appropriately, but fail to reach a stable state for a time-span that is usable

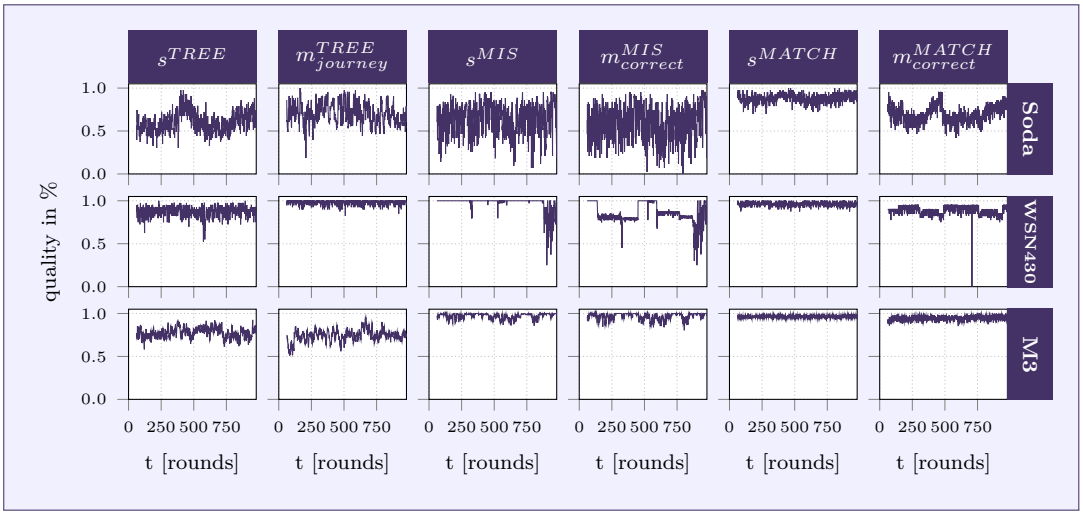


Figure 3.6.: Trend of stability and correctness metrics over time, for the SSAs $\mathcal{A}_{TREE}$, $\mathcal{A}_{MIS}$, and $\mathcal{A}_{MATCH}$ without a TCA

for network protocols, their reaction is too agile. A first step to reduce the number of ongoing changes is to employ a Topology Control Algorithm (TCA). The requirements of SSAs on an TCA are discussed and investigated in detail in Chapter 5. In the following we give a basic understanding of the concept to further motivate its necessity.

3.5. Self-Stabilization with Forced Stability

SSAs evaluate local state information and act upon it. Local in this context, is a node's memory and the k-hop neighborhood. In a wireless network the information about the neighborhood is only available when nodes send messages with state data frequently. Missed messages lead to update delays and possibly stale local states.

Depending on the link quality between nodes, update delays can become arbitrarily large, leading to misinterpretations of neighbor states and possible errors in the self-stabilizing algorithm. A first step to stabilize links is to assess the link quality between nodes and to ignore messages from nodes which are below a chosen threshold Q_{min}. The quality is asset using the PRR measured through sequence numbers to detect lost and received packages and to derive a quality value for a specific link.

3.5.1. Topology Control Algorithm

To assess the quality of a communication link between a neighboring node, a certain number of received messages is necessary. To build a neighborhood of *good* nodes, information (e.g., the quality mapped to the node identifier) about reliable neighbors has to be stored until a satisfying number of qualified neighbors has been established. Information about unreliable neighbors may be stored too, if the memory constrains allow. Usually, they are stored for a limited amount of time, until the TCA is satisfied that the node will not become a valuable asset of the neighborhood in the near future. Additionally, as mentioned before, symmetric (bidirectional) links are of utmost importance.

A sophisticated TCA is presented in Chapter 5. In general, it features dispatch of maintenance messages via broadcasted periodically in every round. Including sequence numbers and the current 1-hop neighborhood. Thus, the symmetry of links can be deduced, and a quality measure can be defined. Fixed-size lists of size C_N are used to store the current neighbor set.

In topologies where the number of high quality (larger Q_{min}) neighbors is lower or equal to the designated storage space all good neighbors are recorded. Since the quality is continually assessed, declining links can be identified and respective neighbors are discarded dynamically. In dense networks (storage space smaller than memory reserved for neighbor states) neighbors are chosen by further requirements to guarantee a connected topology for details. For the upcoming results C_N is chosen big enough to omit the handling of replacements, i.e., all physical neighbors fit into the neighbor list of the TCA.

The employed TCA is used as a filter layer, above the MAC-layer and below the SSAs (recall Fig. 3.5). Only messages from nodes on the neighbor list that are symmetric will be forwarded to the SSA-layer, while messages from other nodes are ignored.

3.5.2. Impact of the Topology Control Algorithm

Including the TCA layer has multiple effects. Firstly, the perceived neighborhood, i.e., globally speaking the topology, becomes altered from the SSA layer point of view. Only high quality symmetric neighbors are recognized and evaluated when executing rules. Secondly, few benefits come without a cost, which will be evaluated

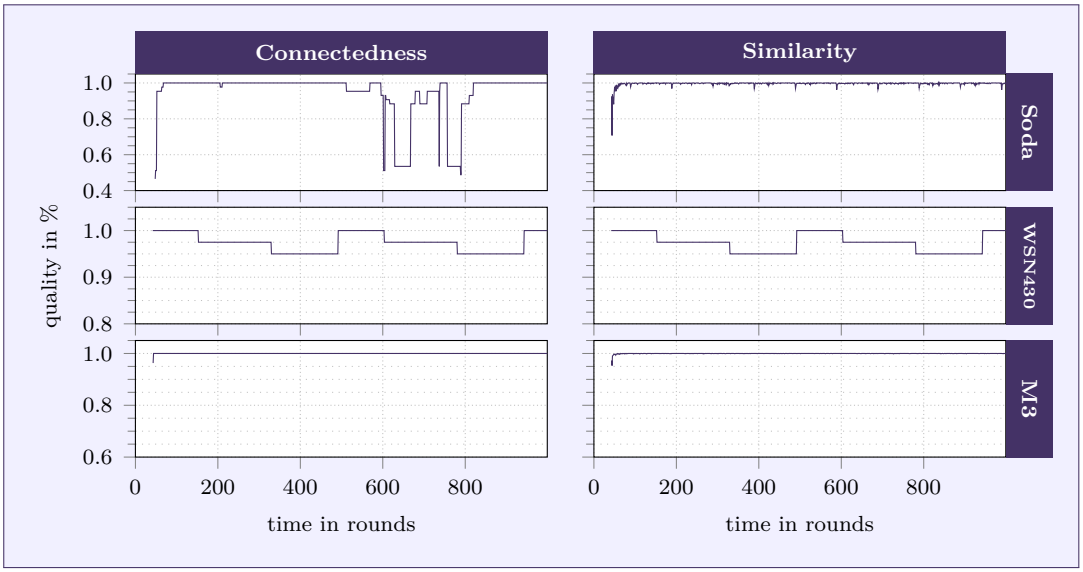


Figure 3.7.: Comparison of topology traces Soda, WSN430, and M3

too. To show the influence of the TCA, the experiments from Section 3.4 are repeated with the TCA in place.

i. Induced Topology by the Topology Control Algorithm

Figure 3.7 shows the influence of the TCA on the observed topology. It corresponds directly to Fig. 3.4. As can be seen the observed graph is smoothed substantially. After a setup phase, the neighborhood stays considerably more stable. This means, if a set of nodes with a quality of at least Q_{min} was chosen, only a small amount of nodes is removed from or added to the perceived neighborhood. This can also be witnessed when comparing the similarity. Without changes in the neighborhood relation at any node, the induced graph does not differ. Adverse effects are that strong links, which occur only for a limited amount of time, are not utilized, because their quality value does not surpass the threshold, i.e., the stability and agility of the TCA are opposing forces which can not both be optimized simultaneously.

An example of the stability and agility of the TCA can be witnessed in the Fig. 3.7 and 3.4 for the Soda trace. The first two short network separations ($t = 100$, $t = 250$) are ignored by the TCA. While the strong disturbance in the connectedness is captured yet smoothed at rounds 600 to 800.

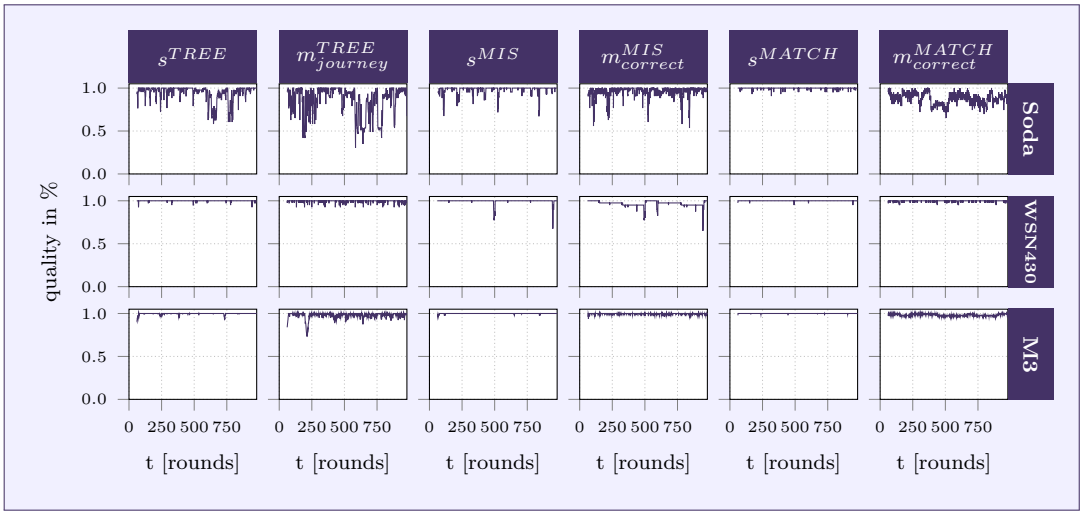


Figure 3.8.: Trend of stability and correctness metrics over time, for the SSAs $\mathcal{A}_{TREE}$, $\mathcal{A}_{MIS}$, and $\mathcal{A}_{MATCH}$ employing a TCA

ii. Influence of the Topology Control Algorithm on self-stabilizing algorithms

Figure 3.8 shows the similarity and correctness for all traces and algorithms with the TCA employed. It can directly be compared to Fig. 3.6. All results look much smoother and mroe promissing with the TCA in place. The WSN430 and the M3 trace show usable results for all algorithms. While the tree, i.e., the journey metric shows only 95 percent of nodes to be able to route messages to the root node. Considering the number of nodes, more than 140, a considerable improvement.

For the Soda trace the $\mathcal{A}_{MIS}$ and $\mathcal{A}_{MATCH}$ results are increased significantly too. Nonetheless, many repair actions are performed by the SSA and its usability for higher level applications is doubtful. Apart from that, the WSN430 and M3 trace are stable over 90 percent of the time.

The SSAs actually stabilize if the underlying topology allows. This is not different to traditional algorithms. Naturally, if the underlying topology consists only of edges with a PRR smaller than Q_{min}, the induced topology cannot be connected. Especially the tree algorithm is very sensitive to changes in the topology. A single error, e.g., link break, has varying strong impact on the quality, depending on their position in the tree. Again this is hardly different to traditional algorithms, with the difference that certain faults are not masked. Nevertheless, our results show that SSAs react appropriately to maintain the tree in presence of errors.

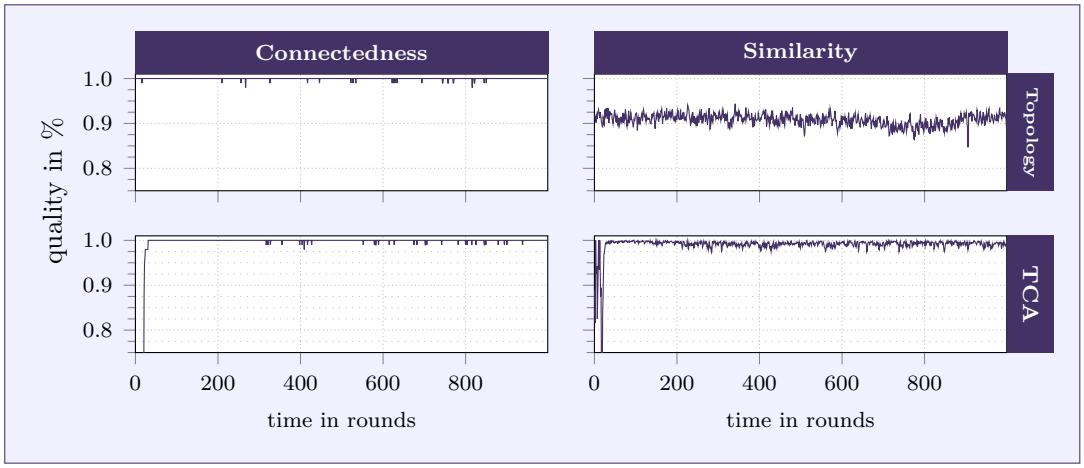


Figure 3.9.: Real deployment test, topology analysis

The figures indicate that without a TCA SSAs in wireless networks are infeasible as long as message loss is common. To reinforce this fact additional tests directly on hardware in a real deployment have been conducted.

3.6. Evaluation on Hardware

To confirm the applied methodology the experiment of the MIS algorithm was repeated directly on hardware. Thus, a set of 100 M3 nodes at the FIT IoT-LAB was chosen. To make the influence of the TCA visible, each incoming package was logged in addition to the changes in the neighbor list, and the state algorithm $\mathcal{A}_{MIS}$. Note that this data can be collected conveniently at the FIT IoT-LAB without storing all the data on the sensor nodes itself [FFH+14]. The observation frequency f_{OBS} was set to 1Hz.

Figure 3.9 shows the connectedness and similarity of the obtained topology and neighborhood. The network is connected most of the time. The unsteady similarity evolving around 0.9 percent indicates frequent changes of the topology, this also leads to fluctuations in the neighborhood. Nevertheless, the TCA is able to smoothed these fluctuations by excluding low quality links from the created neighborhood relation.

The results for algorithm $\mathcal{A}_{MIS}$ are as promising. The correctness $m^{\text{MIS}}_{correct}$ and the stability s^{MIS} of the algorithm are shown in Fig. 3.10. The correctness varies almost every round but remains close to 100 percent, as the topology changes significantly

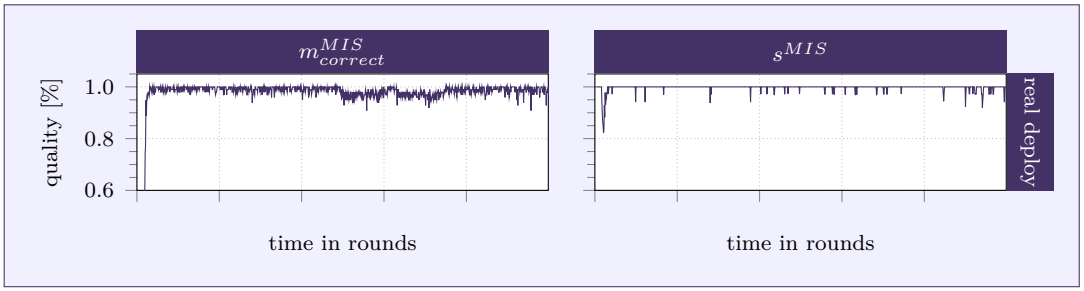


Figure 3.10.: Real deployment test, result of algorithm $\mathcal{A}_{MIS}$

too. Still the algorithm does not react to every single perturbation, and hence, stays stable for the mayor part of the execution.

The data from the hardware evaluation shows similar results as the evaluation with our hybrid methodology (recall Fig. 3.8). The results of algorithm $\mathcal{A}_{MIS}$ are analogical. Even though the created topology fluctuates more, what may be caused by various effects, foremost, the utilization of the FIT IoT-LAB at the time of the experiments, the hardware tests substantiate the methodology of the hybrid evaluation approach. Furthermore, they give a proof of concept for a TCA in a physical environment.

3.7. Concluding Remarks

Self-stabilizing algorithms have been proposed and analyzed for years, but to virtually no practical avail in the wireless communication domain. The usefulness of SSAs as a technique to implement fault-tolerant applications for sensor networks or the Internet of Things has been promoted by the theoretical community. This chapter laid the foundation to substantiate that claim, while indicating limits and bounds.

With the hybrid approach of real wireless communication data and reproducible simulation, different representative self-stabilizing algorithms have been tested for correctness and stability to assess their overall benefit for standard tasks in this area. Self-stabilization without some sort of forced stability is infeasible, due to a fluctuating communication range, which causes permanent disturbances in the neighbor relations of the graph induced by the wireless network. The experiments show that the lengths of the time spans of non-availability are bounded. Thus, mitigating this strongest objection to use self-stabilizing algorithms in the wireless domain. If self-

stabilizing algorithms shall be used in the wireless domain, focus firstly needs to be on an appropriate TCA that fulfills certain standards, e.g., scales with the number of nodes and has manageable memory demands, considering current sensor node hardware.

Chapter 4

Literature Review

An Excerpt of Interconnected Achievements

In Chapter 3 it became clear why self-stabilization in WSNs is difficult and needs a certain amount of rigidity to function. In this chapter important advancements in the research field of self-stabilization for ad-hoc networks are presented, with close attention to topology control. Moreover, various publish/subscribe solutions are advertised as this thesis focuses on this particular use-case for higher-level applications.

4.1. Topology Control

To enable the usage of SSAs in the context of WSNs a relatively stable neighborhood relation needs to be forced (see Chapter 3). To decide if neighbors are *good* the quality of the communication link is assessed. Usually neighbors with high quality links are preferred. Since a node's degree can be arbitrarily high, but the designated memory to maintain the neighbors quality is restricted, it may become necessary to exclude good neighbors. Certain requirements on the neighbor list are stated in Chapter 5 and Section 4.1.2.

4.1.1. Link Quality Estimator

To asses the quality between nodes, i.e, defining a quality for a communication link Link Quality Estimators are used, these are a first step to develop a TCA. LQEs

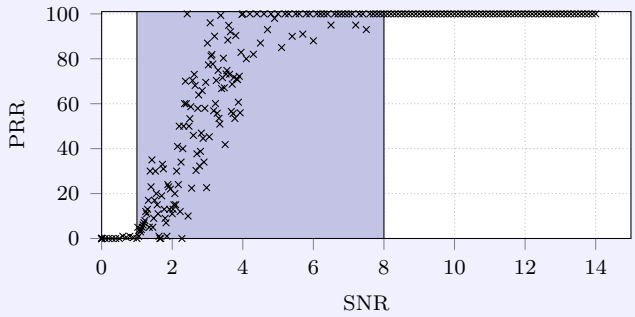


Figure 4.1.: PRR compared to SNR, transitional region, analogous to [BKY+10]

can be categorized into two groups *physical* and *logical*. Physical indicators use the Received Signal Strength Indication (RSSI), the Signal-to-noise ratio (SNR), or the Link Quality Indication (LQI) which are based on the receiver hardware, they are measured using incoming packets. These metrics have the advantage of being free of additional cost, in the sense of overhead, since they are measured every time packets are received.

Their main disadvantage is that their quality is debatable. The measurement quality depends primarily on the used hardware, e.g., a low RSSI value does not necessarily mean that a link will fail to produce reliable communication. The *transitional region* is the typical term used to describe the area where the RSSI or SNR value seems low, while packets still have a chance to be correctly received. The size of this region depends on multiple factors, e.g., used hardware, indoor or outdoor deployments [ZK04]. Figure 4.1 shows a typical transitional region for an experiment measuring PRR and SNR [BKY+10]. Lal et al. [LMH+03] compare the Packet Success Rate (PSR) [CWPE05] and its significance to predict message reception. They show that in case of long-range links the RSSI and the SNR cannot be used to distinguish strong and failing links. One other problem is that lost packets do not provide quality measurements.

Senel et al. [SCL+07] use a Kalman filter to asses the RSSI of successfully received packets, which then is used to estimate the SNR. On the positive side, the continues observation done by the Kalman filter does not fluctuate as much as window based approaches with short windows. Furthermore, a prediction is inherently possible. A shortcoming of this approach is its complexity and the impression caused

by the restricted correlation between SNR and PSR.

Logical indicators collect statistics of lost and received packets which are assessed through sequence numbers. Hence, overhead is generated lowering the maximum message size or introducing additional maintenance messages. Furthermore, a certain amount of memory needs to be reserved for, e.g., the neighbor identification and the current sequence number. Nevertheless, their quality is superior compared to physical indicators, they are not hardware dependent, and many different approaches exist. Most logical metrics are calculated using an Exponentially Weighted Moving Average (EWMA), e.g., Woo et. al [WTC03] use a Window Mean Exponentially Weighted Moving Average (WMEWMA), i.e., a success rate over time is computed.

The Expected Transmission Count (ETX) protocol [CABM03] estimates the number of necessary transmissions to deliver a packet. The number of received packets for certain time intervals are compared to the expected number of packets. Each node thus broadcasts maintenance messages periodically. The length of the evaluation window influences the stability of the metric, long windows lead to fewer ETX updates, short windows to increased fluctuation.

Evaluating the losses during a window is discussed by Cerpa et al. [CWPE05]. They propose that burst errors should be rated worse than random errors, even though their magnitude, i.e., the number of lost messages during an evaluation period, is equal. Hence, the introduced Required Number of Packets (RNP) metric takes the underlying loss distribution into account.

Four-bit [FGJL07] uses a hybrid approach to estimate the link quality. Among physical information, ETX is used as a logical indicator. Two EWMA are applied to smooth the estimator values and to generate a single quality measure. Combining metrics from different sources with EWMA filters can lead to unstable link quality estimation [BKJ+09].

Merging multiple metrics is also applied by the Fuzzy Link Quality Estimator [BKY+10]. The packet delivery rate (WMEWMA), the link ASymmetry Level (ASL), SNR, and the link stability are taken into account. Asymmetry can be deduced by the difference of uplink and downlink PRR, access to these measurements is taken for granted in [BKY+10]. To calculate the ASL each node needs to broadcast its current PRR for all its neighbors, the additional overhead is not accounted for, especially in dense networks the feasibility of this approach is uncertain.

The Holistic Packet Statistics [REWT11] is the first Link Quality Estimator (LQE) that splits its quality measurement into multiple (four) values. These are short- and

long-term estimations, a trend, and a stability indication. An EWMA approach is used to calculate the short-term value, these are fed into another EWMA filter to obtain the long-term value. Link dynamics, i.e., trend and stability are then efficiently calculated lower and upper deviation of the short-term estimation to the long-term estimation. HoPS empowers a system designer to uses a wider spectrum of information than with any other known LQE. Nevertheless, Renner et al. propose two methods to merge the HoPS values into a single value if required.

A comprehensive survey of LQEs was done by Baccour et al. [BKJ+09, BKM+12]. Many of the presented LQEs are compared in broad detail.

4.1.2. Topology Control Algorithms

Logical tracking of the quality of communication links builds a history of successfully received messages, these are linked to neighbor identifiers. Hence, the quality measure and according neighbor identifier are stored locally. Since memory is a sparse resource in WSN, the size of such a *neighbor list* is limited. Therefore, different approaches have been proposed to maintain the lists and possibly evict neighbors to make room for (hopefully) better ones.

Algorithms that collect potential neighbors into a list with constant size automatically use a restricted amount of memory. If the list is full, then either no new neighbor can be added or neighbors have to be removed. Woo et al. [WTC03] compare several replacement schemes. In an experiment with 220 nodes, neighbor list sizes of 40 were necessary to ensure a connected topology. Since links are not being evaluated before insertion, the biggest shortcoming is that neighbors reachable via poor links become part of the neighbor list. Even though poor links will be evicted at some point, the fluctuation is high, resulting in an unstable topology.

TinyOS contains the Link Estimation Exchange Protocol (LEEP) as a neighborhood management protocol [Gna07]. LEEP uses a fixed neighborhood table size and ETX as the LQE. Each node sends out its current knowledge about all its neighbors. The eviction policy for full neighborhood lists is to replace the node with the lowest quality value. If all neighbors have roughly the same quality, small differences can lead to frequent topology updates. Furthermore, even high density networks may be unconnected because LEEP chooses its neighbors exclusively based on the quality value. Another shortcoming of LEEP is that no *ageing* in the quality value is applied, hence, disconnected or switched off nodes stay in the neighbor list until they

are replaced. In sparse networks such stale entries are never removed, this can have adverse effects for upper layers.

TCAs for WSNs are surveyed in [San05, MKP10] showing that many approaches adapt the transmission power to reduce interference. Such algorithms minimize the transmission range until the node degree is manageable while a connected network is ensured. The Adaptive Transmission Power Control (ATPC) algorithm [LZZ+06] permanently collects link quality histories and builds a model for each neighbor. This model represents the correlation between transmission power levels and link qualities providing the basis for a dynamic TCA. Liu et al. [LGCJ04] optimize energy usage by reducing the number of neighbors.

The k-neighbors algorithm [BLRS06] also belongs to this category. All nodes initially use the maximal transmission power to send out their node id. It is decreased as long as each node has at least k neighbors. A node is considered a neighbor if a message can be successfully exchanged. Using the received transmission power, a distance estimate is computed. The challenge is to find the minimal value for k while ensuring that the network stays connected. Blough et al. [BLRS06] report that for networks with 50 to 500 nodes $k = 9$ is sufficient. Their simulations suggest that k can be set to 6 independently of the network's size, if a small percentage of disconnected nodes is tolerated. The k-neighbors algorithm guarantees symmetry, that is, neighboring nodes of a node v add v to their neighbor list. The biggest drawback of the approach is the distance estimation which is as precise as the received RSSI and as mentioned in Section 4.1.1 depends very much on the used hardware.

The Reliable Energy-efficient Topology Control (RETC) algorithm [LSLY13] builds a rooted tree as the overlay topology. The number of children in the tree is not bounded, hence, the memory footprint can exceed the demands of WSN. [LSLY13] makes no mention of link stability, if a better link arises, this link will be chosen in the tree, even if the difference is marginal, this results in constant fluctuation of the created topology.

Similarly to RETC but explicitly self-stabilizing, the approach by Ben-Othman [BOBBP13] builds a Minimum Weighted Connected Dominating Set (MWCDS) on top of a rooted tree. Afterwards, the transmission power is decreased ensuring that this tree stays connected. Therefore, this approach also ensures the symmetry of entries and the connectedness of the created graph. The number of neighbors in the tree is once again not restricted, introducing scaling and memory issues.

Properties	RETC [LSLY13]	EELTC [THHS07]	STC [SG10]	CONREAP [LZN10]	LEEP [Gna07]	XTC [WZ04]	WMCDS [BOBBP13]
Distributed	✓	–	✓	–	✓	✓	✓
Dynamic	✓	–	–	–	✓	–	✓
Stable	–	✓	✓	✓	–	✓	–
Agile	✓	–	–	–	✓	–	✓
Memory does not depend on Δ	–	n.a.	–	n.a.	✓	–	–
Guarantees connectedness	✓	n.a.	✓	✓	–	✓	✓

Table 4.1.: Comparison of TCAs

The XTC protocol [WZ04] bases its decisions on static link qualities such as euclidean distance or energy consumption. Each node orders its neighbors based on this metric. A link between two nodes is used if there is no third node within reach that has an equal or higher quality to both nodes. The resulting topologies is connected whenever the maxpower communication graph is. XTC was extend in [FVW12] by adding all links that exceed a certain link quality to the original XTC links. The resulting algorithm XTC_{RLS} proved in simulations to be superior to XTC and the k-neighbors algorithm with respect to throughput and energy consumption. XTC and foremost XTC_{RLS} have a high memory consumption, exceeding the usefulness in the context of WSN for dense networks.

A completely different approach is used by Zhang [ZCA08]. They use a dedicated node with location information about all nodes, hence, requiring one node with exceptional amount of memory especially with large numbers of nodes. Neighbor lists are not restricted and their algorithm has a single point of failure.

As a wrap-up Table 4.1 gives a comparison of multiple TCAs. The comparison is done in respect to the usage for SSAs in WSNs. Developing a distributed algorithm, that can react to faults in a dynamic way (agile), while ignoring transient faults (stability), and that restricts the memory consumption to an appropriate value (smaller than Δ) are one of the main goals of the TCA designed in Chapter 5.

4.2. Middleware for Wireless Networks

In 1999 Estrin et al. [EGHK99] presented a paper with the title *Next Century Challenges: Scalable Coordination in Sensor Networks*. It was the time when WSN was the *hot new thing*. As programming on those spares nodes needed knowledge of

every part of the device, application, data processing, memory allocation, and message transport, the need for a middleware, making development easier assessable, arose quickly. One year thereafter, the *Sensor Information Networking Architecture* (SINA) [SJS00] was presented, it is the first notable work in this context.

By 2014, middlewares for sensor nodes can be categorizations in five different types [Ond14].

- Application-driven
- Message-oriented
- Database-inspired
- Virtual Machine-motivated
- Agent-based

A further classification of middleware properties is done in the following. For different use-cases and scenarios various middleware features may be desirable, or dispensable overhead. Masri et al. [MM07] propose the following design principles based on related work [YKP04, HM06, MA06, Mar05, RKM02, HR06]. In Chapter 7 a novel middleware for WSNs is presented, keeping in mind the following characteristics. Nevertheless, not all features have been incorporated due to various reasons.

Data centricity For sensing and sending scenarios, as a typical use-case in WSN, most nodes are not interested in the producer of data, but rather of its value. This means data itself plays a more central role than the producer of it. An advantage of data centric architectures is the possibly to apply reduction, aggregation, or other data processing techniques. Data-centric routing and querying are often desirable in WSN, e.g., for publish/subscribe systems.

Energy and resource management Sensor nodes are resource sparse. Especially when considering that they shall be deployed for an arbitrary amount of time and in big numbers battery changing is a nightmare [PS05]. Hence, sensor nodes shall conserve energy to be long living and to allow the use of renewable energy sources, e.g., solar. A middleware needs to take this into account and cannot add to much processing or communication overhead. Furthermore, middlewares for WSN may actively conserve energy, e.g., by load-balancing between nodes with low and high

energy reserves. Other energy saving routinizes, for instance, the support of sleep circles may be handled by an efficient middleware, too.

In-network processing Data aggregation, reduction, or compression are common in-network processing tools to handle the increasing amount of data being generated in the network. For example, when a root node in a tree tries to find the maximum temperature in the network, it is not necessary to transmit the sensor readings of each node, but the parent node of each subtree forwards only the maximum value in its subtree. A many-to-one data flow is more common than end-to-end routing, hence, in-network processing is desirable for most scenarios involving sensor nodes. Even if the full set of data generated is required at certain nodes, compression may be a valuable asset employed by a dedicated middleware.

Quality of Service Support Reliable communication is impossible in WSN. Message loss and radio signal fading due to interference can be mitigated but they are part of a sensor node deployment. Hence, Quality of Service (QoS) support is mostly different to traditional systems. Nevertheless, certain bounds for data freshness, availability can be given by many algorithms. A middleware can include such bounds and should offer certain options to be able to react to the significance of utilized features. E.g., the time it takes to add a node a group of already deployed nodes, or the accuracy of a traced vehicle [SGV+05], may be specified. Note most research considered QoS while designing network level protocols [STT06], but fewer works were reported on middleware level QoS as shown in Table 4.2.

Application knowledge In a layer-based architecture an application is above the middleware which handles certain lower level functions. Injecting application knowledge into the middleware can help to make decisions to increase system performance. E.g., if the maximum of a value has to be determined, then aggregation can be enabled by the middleware. Nevertheless, adding application support will bring a certain amount of overhead and changes the typical layered program flow. Furthermore, the generality of the middleware is lost, since it is directly designed for certain applications. Middlewares supporting application knowledge are usually referred to as application driven.

Scalability As the number of network nodes is usually not restricted in a WSN scalability is an almost indispensable commodity. A middleware that does not scale with the network size can only be used when the maximal number of deployed nodes is known a priori. This limits such middlewares to certain fields or applications.

Dynamic network topology and robustness Sensor nodes are usually not robust and prone to failures. A deployed network takes this into account, as single points of failure are commonly avoided. Due to node shutdowns or additions dynamic adaptation to topology changes can be masked by the middleware. Unreliable communication and variations in the link quality should also be kept transparent to higher levels.

Adaptability Adaptability is closely entangled with *application knowledge*, *Energy and resource management*, and *QoS support*. Changes in the environment, or in the node performance may lead to a necessary change in the current algorithm. E.g., if message loss increases due to a loss in bandwidth lowering the sensing rate may overcome this problem. A powerful middleware that can make such decisions needs to give feedback to an application. Adaptive fidelity algorithms can be interwoven [EGHK99] with such a middleware concept.

Configurability and maintainability Reconfiguration of already deployed nodes is a desirable feature. One of the challenges is to deploy a new firmware without interrupting unchanged algorithms and cooperation of nodes with different firmwares.

Real world integration Refers to time and space monitoring which can be handled by the middleware. E.g., a middleware can enable synchronization. Hence, a local or global clock needs to be accessible. For location information sensors can be used, or a priori location data can be made available.

Security Security is very often considered an add-on feature in the WSN community. Especially because of the resource constrains and the low computation power additional security measures are commonly considered dispensable overhead. Nevertheless, as the IoT becomes reality, security will become more significant.

Heterogeneity Homogeneous and heterogeneous WSN exist. Often more powerful nodes, e.g., *base stations*, are used to build a hierarchical network structure. A mid-

dleware can support such setups. *Software defined heterogeneity* can also be applied by the middleware, e.g., cluster heads or group leaders can be appointed which have different tasks than *regular* nodes.

Table 4.2 give a comprehensive overview over a vast number of middleware solutions for WSNs. It becomes evident from the table that no middleware is able to satisfy every single need a developer can have on a deployment. Some solutions require special treatment, while generic middleware may add to much overhead for possibly necessary features.

The middleware designed in Chapter 7 relies heavily on publish/subscribe. As it is self-stabilizing and has a small memory footprint, certain features have not been incorporated. E.g., application knowledge does not flow back into the middleware layer. Furthermore, QoS support is not explicitly stated, as self-stabilization makes prediction on global system states difficult. Nevertheless, as will be shown in Chapter 7, scalability and dynamic topology and robustness have been mayor design criteria.

To the best of our knowledge, no middleware for WSNs has been derived so far, that it is only designed in a self-stabilizing fashion. Although, publish/subscribe systems exist that incorporate a high level for fault tolerance as well as self-stabilization. Hereon, we focus in the next section.

4.3. Publish/Subscribe

In the following we cover recent publish/subscribe systems which arose in the field of networking with special emphasis on WSN. These are grouped in *tree based* and *overlay* approaches. Since our work focuses on fault tolerance especially through self-stabilization each subsection has a paragraph focusing on works with special interest in fault tolerance as well.

The characteristics of WSNs require lightweight solutions. This also holds for solutions using complex routing structures such as Steiner-trees, rendezvous based [CAR05], informed gossiping, or peer-to-peer networks, e.g., [DGRS03, TP09]. Many such approaches come at high computational costs. Another issue for some of these approaches is scalability. In the case of a Steiner-tree based routing for ex-

Algorithm	Data-Centricity	Energy & Resource Management	In-Network Processing	QoS Support	Application Knowledge	Scalability	Dynamic Topology & Robustness	Adaptability	Configurability & Maintainability	Real World Integration	Security	Hetero-geneity
Database-inspired												
SINA	⊕	⊕	⊕	⊖	⊖	⊕	⊕	⊖	⊖	⊕	⊖	⊖
Cougar	○	○	⊕	⊖	⊖	⊕	⊖	⊕	⊖	⊕		⊖
IRISNet	⊕	⊕	⊕	○	○	⊖	○	⊖	⊕	⊕	⊕	⊖
TinyDB	⊖	○	⊖	⊖	⊖	⊖	⊕	⊖	⊖	⊕	⊖	⊖
DSWare	⊕	⊖	⊖	⊖	○	⊖	⊕	⊖	⊕		⊖	⊖
KSPOT	○	○	⊖	⊕	⊕	⊕	⊖	⊕	⊕	⊖	⊕	⊖
SNEE	⊖	⊕	○	○	⊖	⊕	○	⊖	⊕	⊖	○	⊖
Virtual Machine-motivated												
Maté	⊖	⊕	⊖	⊖	⊖	⊕	⊕	⊖	⊕	⊕	⊕	○
Magnet	⊕	⊖	⊕	⊖	⊕	○	⊖	⊕	○	○	○	⊖
MUSE	○	○	⊖	⊕	⊕	⊖	○	⊖	○	⊖	⊕	⊖
Agent-based												
Impala	⊖	⊕	⊖	⊖	⊖	⊕	⊕	⊕	⊕	⊕	⊕	○
Agila	⊕	⊕	⊖	⊖	⊖	⊕	⊕	⊕	⊖	⊕	⊖	⊖
SWAP	○	○	○	⊖	○	⊖	⊕	⊖	⊕	⊖	○	⊖
MAPS	⊕	⊖	⊕	○	⊕	⊕	○	○	○	⊖	○	⊖
Application-driven												
MiLAN	⊕	⊕	⊖	⊕	⊕	⊕	⊖	⊕	⊖	⊕	⊖	○
AMC	⊕	⊖	○	○	⊕	⊖	⊕	⊕	⊕	⊖	⊕	⊖
MidFusion	⊕	⊕	⊖	⊕	⊕	⊖	⊕	⊕	⊖	⊕	⊕	⊖
Message-oriented												
Mires	⊕	⊕	⊕	⊖	⊖	⊕	⊖	⊖	⊖	⊖	⊖	○
AWARE	⊕	⊕	⊖	⊕	⊖	⊕	⊕	⊖	⊕	⊖	⊕	⊖
WMOS	⊕	⊖	○	○	⊖	⊕	○	⊕	⊖	⊕	⊕	⊖
TinyMQ	⊕	⊖	○	⊖	○	⊕	○	⊖	⊕	⊖	○	⊖
EnviroTrack	⊕	⊕	⊕	⊖	⊖	⊕	⊕	⊖	⊖	⊕	⊖	⊕
TeenyLIME	⊕	⊕	⊖	⊖	⊖	⊖	⊕	⊖	⊖	⊕	⊖	○

Table 4.2.: Overview of WSN middleware solutions

ample, the more subscribers there are in the system, the bigger the Steiner group gets. Publish/subscribe for mobile systems, recently arising in the context of cloud computing [CMMP08, NKAA14], are excluded too.

4.3.1. Overlay approaches

There have been several efforts towards publish/subscribe systems for WSNs. The overlay infrastructure directly impacts performance and scalability, such as the message routing cost.

Directed Diffusion is an early approach [IGE00]. Nodes issue subscriptions by broadcasting a query into the entire network. Upon receiving a query, each node creates a gradient entry in the routing table to point to the neighboring node from which the query is received. Using a gradient path, matching messages are sent toward subscribers.

Another approach is taken by the MQTT-S protocol [HTSC08]. Here, publication matching is carried out by a small number of dedicated brokers located on nodes external to the WSN.

Approaches with focus on fault tolerance Considering robustness, Jerzak et al. [JF09] designed a generic publish/subscribe system to mitigate routing information corruption. A soft state approach is used and their focus mainly lies in the determination of a valid reallocation time for subscriptions and advertisements. The soft state approach is similar to the leasing technique it is used to eliminate stale routing information.

PS-QUASAR [CDRT13] is a publish/subscribe middleware based on the Contiki operating system that handles QoS, with respect to reliability, deadline, and priority. The underlying structure is an acyclic graph where each node is assigned a level and a list of parent nodes [CLKB04]. PS-QUASAR keeps record of the best known distance to each subscriber and to the next relay node, while the neighborhood size is not restricted, thus, the scaling of the approach in respect to memory remains unclear.

Jaeger et al. proclaim a self-stabilizing publish/subscribe system [Jae08]. Their algorithm requires an acyclic *broker overlay network* to route publications to subscribers directly connected to brokers. If no acyclic broker overlay network exists in the deployed topology, a tree can be built as a basis. Subscriptions and advertisements are used to generate routing tables, linking subscribers and publishers. Brokers cannot be subscribers or publishers, they merely serve as data dissemination nodes.

To ensure that the system is self-stabilizing the leasing technique is used to fix possible faults in these routing tables. The renewal is triggered by periodically dispensed subscription messages.

4.3.2. Tree based approaches

Many different variants of tree-based approaches exist in the publish/subscribe domain. The main disadvantage of tree-like approaches in general is that communication links between nodes on the same or adjacent tree levels, not belonging to the spanning tree, are not used for routing. In a worst case scenario: neighboring nodes involve a major part of the network to be able to communicate. Nevertheless, tree-based approaches are the common type of infrastructure used for publish/subscribe.

Scribe [CDKR06] is a decentralized application-level multicast infrastructure implemented on top of Pastry [RD01], a peer-to-peer location and routing overlay on the Internet. Each channel has an owner known to all nodes. Publications are first send to the channel's owner which distributes messages via a multicast tree to the channel's subscribers. Subscriptions to a channel are forwarded to the channel owner using Pastry's routing protocol. Nodes on a route snoop on subscribe messages. If the channel is one to which the current node already subscribes, it will stop forwarding the subscription and add the node as one of its children. This way a treelike routing structure is formed. Scribe relies on Pastry to optimize the routes from the owner to each subscriber. Fault tolerance is accomplished through the use of timeouts and heartbeat messages.

Huang et al. [HGM03] propose a definition of optimality of a publish/subscribe tree, thereby taking the characteristics of the ad-hoc network and the emerged route into account. They use a distributed greedy algorithm to establish the routing tree. Additionally knowledge of the type of subscription can be infused in the system to improve the considered tree while lowering the generality of the approach.

Mires [SGV+05] is a middleware for WSNs (see Section 4.2), it exclusively uses the publish/subscribe paradigm to route messages. It is built for TinyOS. Nodes have to advertise a topic to become a publisher and dedicated sink nodes are then able to act as subscribers. Mires uses different phases to eventually build a routing tree. Topology changes or node crashes can not be handled without a system reset.

Channel-based publish/subscribe systems are often realized as an application-level overlay of brokers connected in a peer-to-peer manner. Overlays are logical networks

on top of real network where links correspond to paths in the underlying network. The overlay infrastructure directly impacts performance and scalability, such as the message routing cost. A particular aim is to find overlays connecting publishers and subscribers using the minimum possible number of edges or the minimum node degree [CMTV07, OR16, CJV16, HHI+15]. Chockler et al. showed that most of these optimization problems are NP-complete. Hence, practical solutions have to get along with sub-optimal solutions.

Hence, Chockler et al. organize all nodes interested in a common channel into a tree [CMTV07]. They present a logarithmic approximation for the problem of the *Minimum Topic-Connected Overlay*, i.e., they try to minimize the complexity of the overlay by finding a tree with low diameter. For a set of subscriptions to different channels, connect the nodes using the minimum possible number of edges, so that, for each channel c a message published for c should reach all subscribers of it by being forwarded only through the nodes subscribed to c. The biggest shortcoming of this algorithm in relation to WSNs is that their algorithm is not distributed.

Tran et al. [TP10] built a tree overlay structure as well. They assign unique positions to all nodes as a binary string. The virtual address space thereby grows linearly with the tree depth and width. Their algorithm is built for peer-to-peer systems, i.e., it is a distributed algorithm, but it is not self-stabilizing.

Approaches with focus on fault tolerance Shen [She07] proposes a system where all messages are routed along a single spanning tree T. Each node v holds a routing table consisting of a set of tuples of the form (c, R) where c is a channel and R is a set of neighboring nodes in T. If a message arrives at v matching a channel c in the routing table, then it is forwarded to all nodes in R, except for the node from which the message arrived. Subscriptions and unsubscriptions are also forwarded along T to all nodes.

To keep this structure intact routing tables are exchanged periodically. To mitigate the fact that these tables can be very big *sketches* selected by Bloom filters are used to send parts of the table and only in a fault case the whole table is exchanged. Inconsistencies between the tables lead to corrective actions. Nodes make local corrections independently and asynchronously. Through a sequence of local corrections the consistency among the distributed routing tables is eventually restored. The drawback of the above described approaches is that messages travel only along tree edges. This leads to unnecessary long paths and thus, to a high network load. Furthermore,

Shen's approach lacks scalability mainly due to the periodic exchange of complete routing tables.

Furthermore, the system may not be self-stabilizing, because of the information loss it cannot be guaranteed that existing corruptions are detected. Jaeger discusses the self-stabilization property of Shen's approach even more skeptical and argues that, due to the fact that the used Bloom filter is computed incrementally, a corruption in a filter may lead to a system state which will never stabilize [Jae08].

Chapter 5

Forced stability

The Cornerstone for Self-stabilization in Wireless Sensor Networks

In the previous chapters it was pointed out that topology control is a necessity to enable self-stabilizing algorithms in the wireless domain. Because SSAs need a non-changing, i.e., rigid graph as changes in the graph, e.g., link additions or removals are considered faults and cause rule executions. If a graph changes at every observed time step, then no SSA can stabilize. In the following a sophisticated TCA is presented. It evaluates its neighbors quality with the help of an Link Quality Estimator, stores a restricted set of neighbors to form the local neighborhood of a node, and dynamically adds and evicts increasing and deterioration nodes, respectively.

5.1. Overview

This section clarifies the term TCA. Furthermore, criteria are defined to compare and evaluate different TCAs among each other.

5.1.1. Topology Management in General

The goal of a TCA for WSNs is to dynamically form a graph G_c representing the communication network to be used by upper layers, e.g., applications, while maintaining global graph properties, foremost, the connectedness of G_c, given the fact

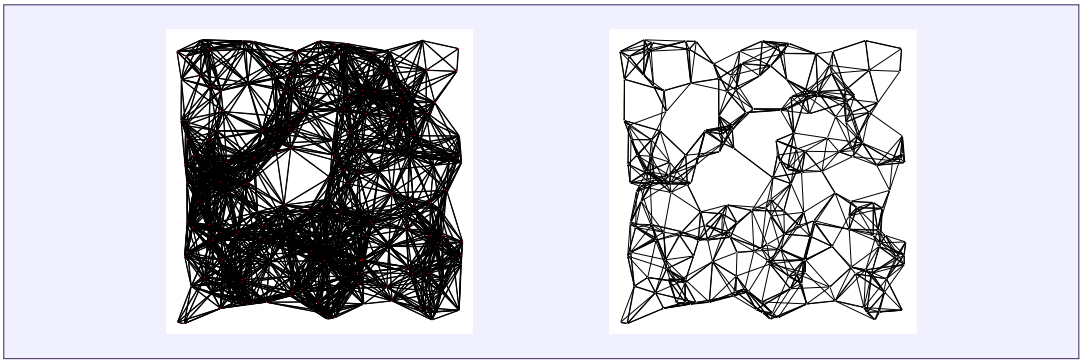


Figure 5.1.: Physical topology thinned out by the proposed TCA

that the graph G representing the physical topology is connected. G_c is a subgraph of G it contains all possible communication links independent of their quality. From a communication point of view, topology control aims to achieve reduced interference, less energy consumption, and increased network capacity. In essence, a deliberate choice is mode on each node's neighbors, i.e., the physical topology is thinned out. In Fig. 5.1 an example is given. It shows the physical topology of 225 randomly placed nodes in a snapshot of a simulation. Each edge in the left graph connects two nodes that where able to communicate over the last 60 rounds. The graph on the right shows the result of the TCA presented in Section 5.2. It fixates the outgoing node degree to at most five, while ensuring the connectedness of the created topology.

Many TCAs disregard fluctuations over time, i.e., they keep a computed topology forever (recall Table 4.1). It is well known that wireless links frequently disintegrate while others are established, as evaluated in previous chapter or by Woo et al. [WTC03]. In particular in WSNs with resource-constrained low-power radios, this makes multi-hop routing as required in data collection applications, a challenge [PGY+11]. The origin of these fluctuations are typically environmental changes, e.g., unstable weather conditions or moving obstacles. Raising the transmission power increases the number of direct communication partners, introducing benefits but also disadvantages. A high transmission power leads to an increased energy consumption. Secondly, a bigger number of neighbors places a heavy burden on the MAC protocol (e.g., for CSMA/CA protocols based on ready-to-send/clear-to-send (RTS/CTS) assessment. To continuously find usable communication partners, link qualities have to be assessed and monitored. Woo et al. suggest to estimate link qualities dynamically through an LQEs using metrics such as PRR. Due to the limited memory in WSNs

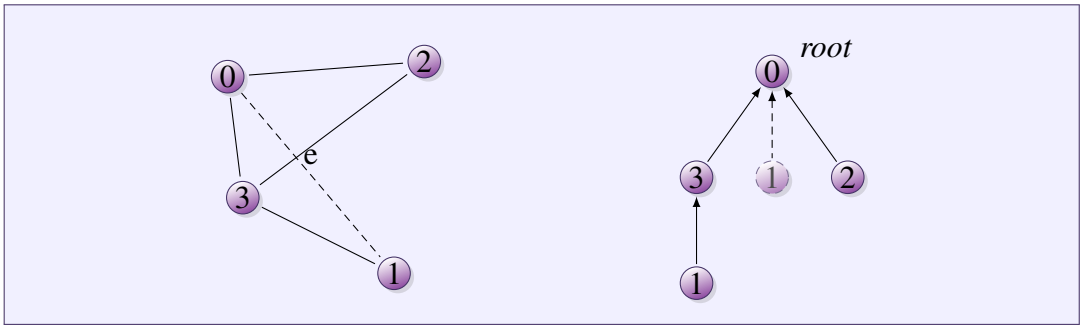


Figure 5.2.: Influence of instability on spanning tree creation.

link status statistics must be maintained in tables of constant size, regardless of the cell density [WTC03].

We argue that TCAs for WSNs must continuously react to changing link qualities. In doing so it must provide agility and stability. Once the quality of a link drops significantly the number of retransmissions will rise, leading to more contention with negative consequences for the network. Thus, TCAs should quickly discard low quality links. While agility is desirable, from an application's point of view, stable topologies are preferred since otherwise provided services may not be guaranteed, e.g., routing towards a target node fails. Once a topology satisfies a *goodness criterion*, stability demands to keep the topology unchanged as long as possible, i.e., to not react to transient faults (e.g., short temporal obscuration due to passing people) prematurely. Agility demands to promptly update a topology as soon as the goodness criterion is violated permanently.

Stability is crucial if neighborhood information is used to compute support-structures, e.g., spanning trees. Consider the execution of a self-stabilizing, distributed spanning tree algorithm for a network with an unstable link e as shown in Fig. 5.2. Unstable in this case may be a PRR value below 70 percent (according to the considerations in Chapter 3). Here a node makes decisions based exclusively on its *local view* (own and neighbor states). Repeated inclusion and exclusion of link e causes the parent of node 2 to oscillate. Such instabilities may lead to loops, with severe impacts on tree routing. Excluding edge e permanently from the list of suitable communication partners, i.e., from G_c stabilizes the tree.

5.1.2. Topological Criteria

A distributed implementation of an LQE periodically computes a quality value $Q(e)$ for each link e in graph G. A TCA dynamically computes a time depended subgraph $G_c(t) = (V, E_c(t))$ of $G(t)$ (i.e., $E_c(t) \subseteq E(t)$) satisfying the following criteria.

1. **Link quality**
   $Q(e) \geq Q_{min}$ for each $e \in E_c(t)$ and any time t. With high probability link qualities below Q_{min} trigger a large number of retransmissions leading to more contention and thus, reduce the overall throughput.

2. **Symmetry**
   Links in $E_c(t)$ should allow for bidirectional communication. Unidirectional links are considered less useful because Acknowledgments (ACKs) cannot be sent directly. Additionally, some SSAs may not be defined on a directed graph, e.g., [MT10].

3. **Connectedness**
   If $G(t)$ is connected, then $G_c(t)$ should also be connected. Otherwise diminishing the usability of the system and excluding nodes or splitting the network into smaller subnetworks.

4. **Bounded degree**
   Each node in $G_c(t)$ should have at most C_N neighbors regardless of the size of V or $E(t)$. An implementation of criterion 1 and 2 requires to store data about each potential neighbor (e.g., the link quality). Thus, the available resources limit the number of nodes that can be maintained.

   Static TCAs do not consider temporal aspects of link behavior, i.e., they do not react to link changes or failures. A simple remedy is to repeatedly apply a static TCA. This can lead to strong discontinuities of the topology, including oscillation and long adherence to low quality links. We therefore demand:

5. **Stability**
   If $e \in E_c(t_0)$ and $Q(e) \geq Q_{min}$ at time t_0, then e should remain in $E_c(t)$ for $t \in [t_0, t_0 + T_N]$ (T_N be a given constant). Furthermore, e should remain in $E_c(t)$ for $t \geq t_0$ as long as $Q(e) \geq Q_{min}$ and the other criteria are fulfilled. A neighbor of a node should not be instantaneously replaced when a neighboring node with a higher link quality emerges.

6. **Agility**
   If $E_c(t)$ does not satisfy the above listed criteria and there exists an $e \in E(t) \setminus E_c(t)$ that can improve at least one criterion, then e should be included into $E_c(t)$. Similarly, if there exist $e \in E_c(t_0)$ with $Q(e) < Q_{tol}$ (a given threshold for a minimal tolerable quality), then e should be discarded from E_c during $[t_0, t_0 + T_N]$.

7. ***k*-Spanner**
   The distance between two nodes in $G_c(t)$ should be at most k-times the corresponding distance in $G(t)$. A more practical criterion is to demand that the average length of paths between any pair of nodes should be small.

The agility and stability criteria can not be optimized simultaneously. For the connectedness and the k-spanner criterion global knowledge seems indispensable.

5.2. The Topology Control Algorithm

Our proposed TCA, referred to as NeighbORhood Management for Ad-hoc Networks (NORMAN) dynamically forms a stable network topology $G_c(t)$ on top of a physical network despite fluctuating link qualities and re-/disappearing nodes. It is a self-organizing algorithm, i.e., it continuously adapts the logical topology to the physical conditions. Each node selects a limited number C_N of nodes as neighbors. NORMAN is built to incorporate the above listed criteria and to find a fair compromise between agility and stability.

5.2.1. Link Quality Estimator – HoPS

The first task before being able to decide if a neighbor may be beneficial is to asses its quality. The LQE used in for NORMAN is HoPS, it evaluates the channel by observing package loss through sequence numbers. As HoPS is a major part of NORMAN its functionality and defining properties are presented in the following [REWT11]

HoPS uses an EWMA filter for a short-term estimate $Q_S(t)$ with a tuning parameter α: Equation 5.1a. Q_S is used to predict a long-term value $Q_L(t)$ with another EWMA filter and an additional tuning factor β with $0 \leq \alpha < \beta \leq 1$:

$$Q_S(t) = Q(t-1)_S \cdot \alpha + (1-\alpha) \cdot q_t \quad , \tag{5.1a}$$
$$Q_L(t) = Q(t-1)_L \cdot \beta + (1-\beta) \cdot Q_S(t) \quad , \tag{5.1b}$$

In addition, HoPS evaluates the dynamics of a link using a third EWMA filter with a tuning factor γ with $0 \leq \alpha < \gamma \leq 1$, to determine the linear upper δ^+ and lower δ^- deviations.

$$\delta^+ = \delta^+(t-1) \cdot \gamma + (1-\gamma) \cdot \zeta(Q_S(t), Q_L(t)) \quad , \tag{5.2a}$$
$$\delta^- = \delta^-(t-1) \cdot \gamma + (1-\gamma) \cdot \zeta(Q_L(t), Q_S(t)) \quad , \tag{5.2b}$$
$$\zeta(x,y) = \begin{cases} x-y, & \text{if } x > y \\ 0 & \text{else} \end{cases} \quad . \tag{5.2c}$$

These values are used to calculate the absolute deviation estimation Q^V and the trend estimation Q^R:

$$Q_V(t) = \delta^+ + \delta^- \, ,$$
$$Q_R(t) = \delta^+ - \delta^- \, .$$

The estimate Q_V supplies an expression to evaluate the stability of a link comparing the variation of Q_S and Q_L. The calculation of a complete deviation is bypassed while the result is as expressive. The value Q_R illustrates the trend of a link. If the long-term quality stays constant Q_R will be close to zero, while a negative or a positive, Q_R value describes a disintegrating or improving link, respectively.

Figure 5.3 gives a graphical representation of the prediction values. Here the situation is considered that a node evaluates a perfect link to one of its neighbors, which is turned off at time $t = 0$. From now on no messages are received anymore, hence, the Q_S and Q_T values gradually decline. Only considering the long-term value, after about 30 missed packets the perceived quality falls below 0.7. If this value is additionally compared to Q_V and Q_R, then, due to the firmly negative trend and the strong variation in stability, a concrete prediction of the state of the communication

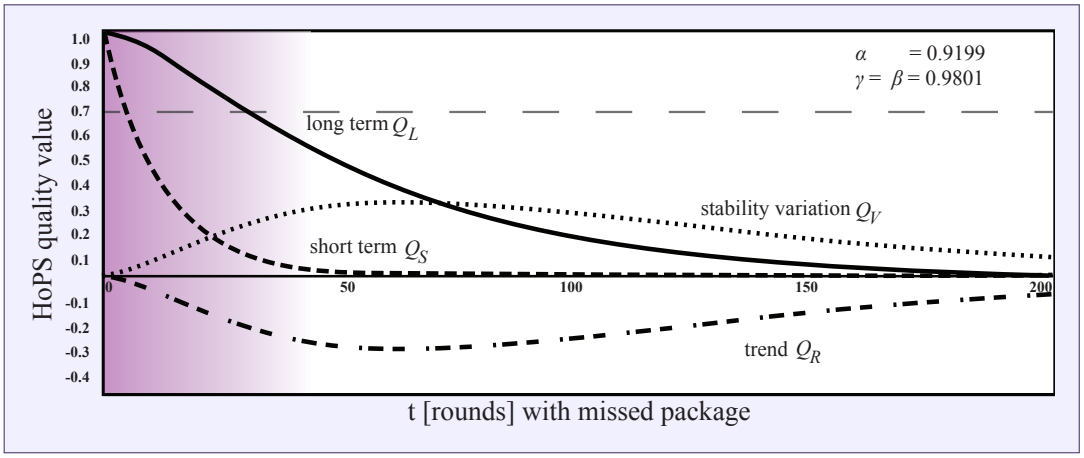


Figure 5.3.: HoPS quality values over time only considering message loss

to the neighbor is possible.

To the best of our knowledge HoPS is the only LQE that provides multiple values to estimate a link. Note that parameters α and β directly influence Q_S and Q_L, receptively. Hence, these parameters can be used to influence the agility and the stability of NORMAN. The more influence lost and received packets have on Q_S and Q_L (decreased α or β) the faster the TCA will react.

NORMAN uses only two of the provided metrics to evaluate the *goodness* of a neighbor. These are Q_L and Q_V. For self-stabilizing algorithm it is of utmost importance to have stable links, hence, the trend of a link was deemed less important. In the following it is also shown that the quality of a link is not the most important factor when faced with the challenge to chose between multiple neighbors. In the following we will omit the parameter t in $Q(t)$, hence, Q is the currently estimated link quality.

5.2.2. Data Structures and Local Topology

Each node maintains a subset of the nodes it can receive messages from in three disjoint lists A, S, and N. Nodes in the neighbor list N form the currently defined neighborhood of node v. Messages from nodes in the physical neighborhood of v that are not on the list are being forwarded to higher level layers. This means, the neighborhood algorithm acts as a message filter. Note that for a certain time it may hold that $v.N \not\subset N[v]$ because $v.N$ has not been updated recently.

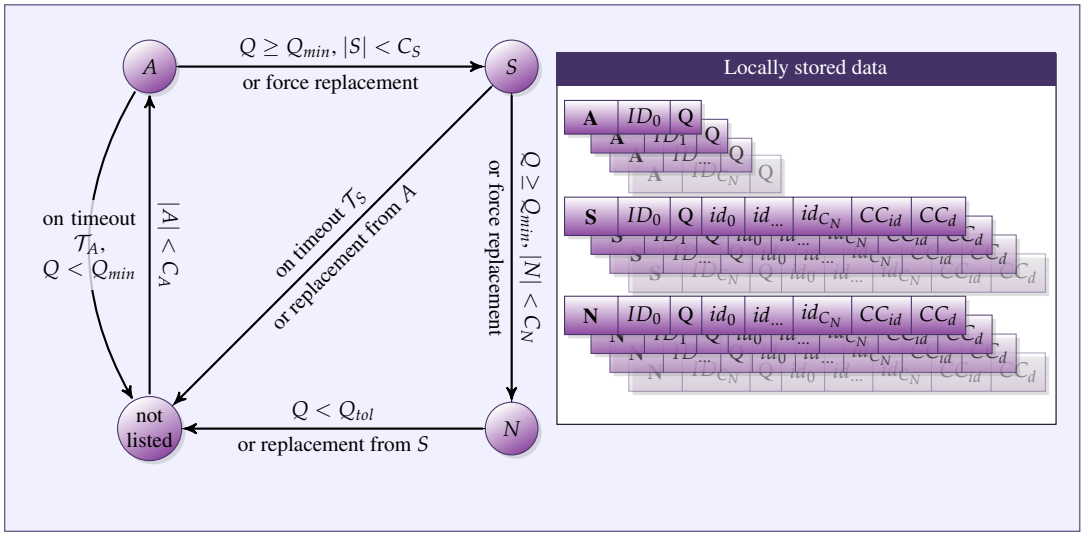


Figure 5.4.: Transitions between lists A, S, and N and data stored per list

The other two lists are used for neighbor assessment. Acceptance into v.A depends merely on the availability of an open slot. Here the quality of a link is assessed using HoPS. To assess links, every node v periodically broadcasts a maintenance message. These messages allow the LQE of a receiving node u to estimate the quality of the link e between v and itself. e is considered useful if $Q(\geq)Q_{min}$. For the purpose of sheer quality assessment each node collects unidirectional information about links in the *assessment list* A of length C_A. After being assessed with a quality of at least Q_{min}, a node can be promoted to the *standby list* S.

S is organized as a priority queue based on link quality. Nodes in this list are standby nodes to improve the *local topology* defined by the nodes in the neighbor list N. The $LT[v]$ is the subgraph of $G_c(t)$ induced by the nodes in v. A node v can construct $LT[v]$ from the information contained in its list N.

Agility and stability are provided by deliberately promoting nodes to and deleting them from N. Figure 5.4 presents a state diagram depicting the transition between the lists. This process is controlled by two periodic timers. Upon expiration of the *assessment timer* $\mathcal{T}_A$ a node checks if the quality of the nodes in A is sufficient to get promoted to S (details in Section 5.2.4). Upon expiration of the *neighborhood timer* $\mathcal{T}_N$ a node checks whether nodes in S can be used to improve the local topology (see Section 5.3).

The main data structures for a node are the three lists A, S, and N of length C_A, C_S,

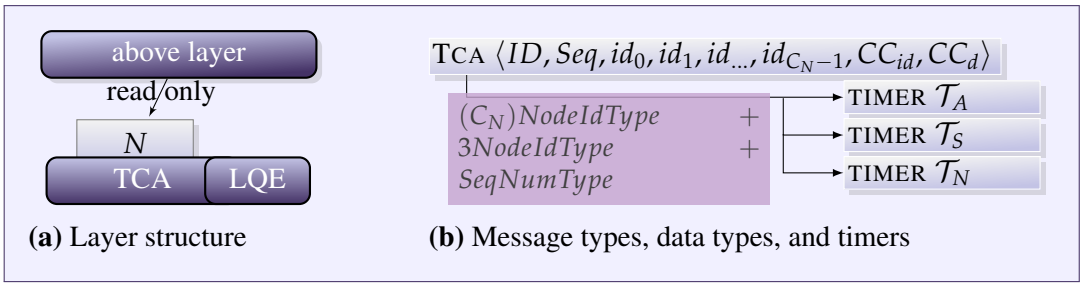
and C_N, respectively. For each link to be assessed, A contains the sender's identifier and information required to estimate the link quality. S and N contain the identifiers of at most C_N nodes along with some additional information such as the link's quality measures (sketched in Fig. 5.4; CC_d and CC_{id} become explicit in Section 5.3.2). Nodes in N are regarded as the node's current neighborhood in G_c. List and timer lengths have to be chosen according to the node density and fluctuation rate. In the evaluation Section 5.4 of NORMAN certain thresholds and recommendations are presented.

5.2.3. Processing of Periodic Messages

The periodically broadcasted messages by a node v include, besides the data required by the LQE, the identifiers in v's list N, i.e., message size is $O(C_N)$. The maintenance message of type TCA is depicted in Fig. 5.5b. A node v stores for each $w \in v.N$ the list of w's current neighborhood as reported in w's last message. With the help of this information a node computes its local topology. The type of information stored in S per node is the same as in N, i.e., the neighbors of each entry. This leads to a total storage demand of $O(C_N^2)$.

Figure 5.5a shows the layer structure. *Above layers* may be application, or routing structures. The neighbor list N is accessible by upper layers. In Fig. 5.5b the typical *NodeIdType* is considered to be two byte, and the *SeqNumType* is one byte, if not stated otherwise.

Upon the reception of a periodically sent message from a node w, a node v checks if w is already contained in A, S, or N. In this case the LQE updates the value of the link quality. If w's identifier is already contained in $v.S$ or $v.N$, then the neighborhood data of w is updated for the respective list. The received information



(a) Layer structure **(b)** Message types, data types, and timers

Figure 5.5.: TCA layer protocol overview

allows v to deduce whether the link to w is bidirectional, by checking if its own identifier is contained in the received list N. If w is not contained in any of the lists A, S, or N and $|A| < C_A$, that is, space is available in A, then w is inserted into A. If A is full, then the link to w is not considered at this instant, with the goal to improve stability, i.e., not to prematurely replace good links.

5.2.4. Periodic Processing of Lists

A node w remains only for a fixed time interval $\mathcal{T}_A$ in $v.A$ of a node v. $\mathcal{T}_A$ depends on the number of messages the LQE requires to assess a link and on the number of nodes within the node's transmission range. After time $\mathcal{T}_A$ the LQE has assessed w's link quality $Q(w)$ and the entry is removed from A to allow other links to be evaluated (see Algorithm 5.1). If $Q(w) < Q_{min}$, then w is discarded, otherwise it is considered as a candidate for being a neighbor of v, i.e., it may be included into S. A node w removed from A is inserted into S if $Q(w) \geq Q_{min}$ and $|S| < C_S$. If S is already full, then w replaces the entry u in S with the lowest quality value, provided the quality of u is lower than that of the link to w, otherwise it is discarded.

Algorithm 5.1 Promoting nodes from A to S

```
Upon expiration of timer T_A for p ∈ A:
  A.remove(p)
  if p.Q ≥ Q_min then
    if |S| < C_S then
      S.add(p)
    else
      q := min_Q S
      if q.Q < p.Q then
        S.remove(q)
        S.add(p)
      end if
    end if
  end if
```

With period $\mathcal{T}_N$ each node updates lists N and S as follows (see Algorithm 5.2 and Fig. 5.4). A node is discarded from N if the quality of the link falls below Q_{tol}. We define $Q_{tol} < Q_{min}$ to spare a newly promoted node from being evicted from N due to a minor degradation of its link quality, thus increasing the stability of the approach. If $|N| < C_N$, then the best $C_N - |N|$ nodes of S are promoted to N. A promotion is only performed if the link's quality is above Q_{min}. Note that link qualities may decrease over time. A node $p \in S$ may supersede a node in N if this increases the *rank* of the local topology (details: Section 5.3). For this purpose the nodes in S with a quality value above Q_{min} are considered in decreasing order. If there exists a

Algorithm 5.2 Promoting nodes from S to N

```
loop with period T_N
 for all p ∈ S in descending order do
  if p.Q ≥ Q_min then
   if |N| < C_N then
    N.add(p)
    S.remove(p)
   end if
   q := replacementCandidate(N, p)
  end if
  if q ≠ null then
   N.remove(q)
   N.add(p)
   S.remove(p)
  end if
 end for

end loop
```

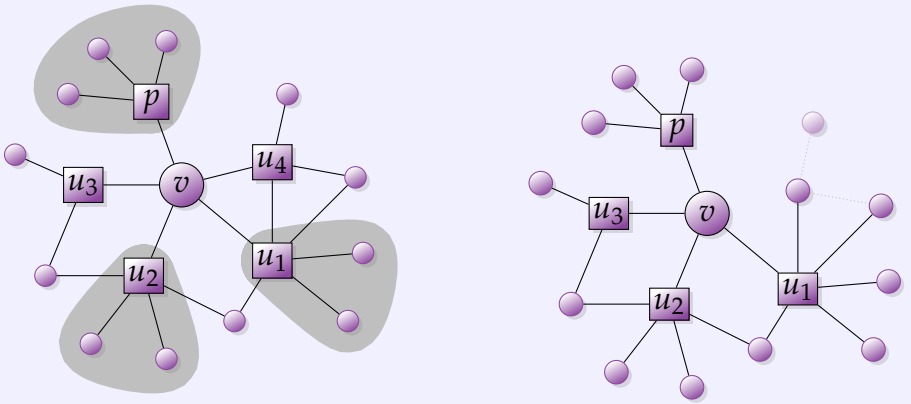
$q \in N$, such that, the local topology defined by $N \backslash \{q\} \cup \{p\}$ has higher rank than the local topology defined by N, node p replaces q in N and is removed from S. The replaced node is not merely downgraded to S but removed entirely from all lists. The rationale for this behavior is as follows: Node q was replaced in N because other nodes formed a higher ranked topology. Therefore, it should not be reinserted too soon to give other nodes a chance to further improve the topology's rank and to avoid oscillation, i.e., to foster stability. This concept realizes a temporary blacklist. For similar reasons nodes only remain for a fixed period $\mathcal{T}_S$ in S. If they are not promoted during that time, then they are discarded to make room for other nodes from A which may improve the rank. The lengths of $\mathcal{T}_N$ and $\mathcal{T}_S$ control the agility of NORMAN.

5.3. The Rank of a Local Topology

At this point it was not explained how Algorithm 5.2 decides to replace a node in N, i.e., how to improve the rank of a local topology $LT[v]$ as implemented by function $replacementCandidate(p, N)$. The function evaluates four metrics trying to improve values for the topology criteria 1, 2, 3, and 7.

5.3.1. Minimizing Paths Length

To assess the k-spanner property of a local topology $LT[v]$ the metric *weight* is introduced. Nodes in $LT[v]$ are called *level 1 nodes* and all other nodes, except v are called *level 2 nodes* of $LT[v]$ (see Fig. 5.6). Thus, in $LT[v]$ every level 1 node w is adjacent to v and those level 2 nodes that w reported in its last message to v. $LT[v]$



(a) $LT'[v]$ before addition of p **(b)** $LT[v]$ after addition of p

Figure 5.6.: p replaces u_4 because $\omega(u_4)$ is the minimum

contains at most $1 + C_N + C_N^2$ nodes. C_v denotes the set of level 1 and 2 nodes of v, henceforth called the *cover* of v.

To construct a global topology with a small average path length we use the following heuristics. Each node v attempts to maximize the number of nodes in C_v. Among the local topologies with the same number of nodes the one which minimizes the distances from v to the nodes in its cover is preferred, i.e.,

$$Len(LT[v]) = \sum_{w \in C_v} dist(v, w)$$

is minimized. Here $dist(v, w)$ is 1 if w is a level 1 node and 2 otherwise. To satisfy a weaker form of the k-spanner property, each node aims to minimize Len as shown in the following.

To test whether $p \in S$ can increase the rank of $LT[v]$, p is inserted into N and the enlarged local topology $LT'[v]$ is analyzed. Let u be a level 1 node in $LT'[v]$. The impact of removing edge (v, u) from $LT'[v]$ is assessed by an integer weight $\omega(u)$. A level 2 neighbor of u is called *dependent* if it is not adjacent to another level 1 node in $LT'[v]$. Denote by $dep(u)$ the number of dependent neighbors of u in $LT'[v]$. Removing the link to a level 1 neighbor u increases the distance from v to all $dep(u)$ dependent neighbors of u. In Fig. 5.6 $dep(u_3) = 1$ and $dep(u_2) = 2$. Now weight $\omega(u)$ is defined as follows:

1. $\omega(u) = dep(u) + 1$ if $N(u) \cap N(v) \neq \emptyset$: u is adjacent to another level 1 node of $LT'[v]$
2. $\omega(u) = 2(dep(u) + 1)$ if $N(u) \cap N(v) = \emptyset$ and not all neighbors of u that are level 2 nodes are dependent.
3. $\omega(u) = C(dep(u) + 1)$ if $N(u) \cap N(v) = \emptyset$ and all level 2 nodes that are neighbors of u are dependent, here C is a constant larger than C_N.

The weight of a level 1 node u of $LT'[v]$ expresses the degradation of the topology when edge (v, u) is removed. In the first two cases the number of nodes in the local topology remains constant, but $Len(LT'[v])$ is increased by $\omega(u)$. In the third case u and its dependent neighbors are no longer connected to v. Since $C > C_N$ this case always gets a higher weight than the other cases.

Consider the extended local topology of node v depicted in Fig. 5.6:

$$\left.\begin{array}{l}\omega(u_1) = 3\\ \omega(u_4) = 2\end{array}\right\} 1st\ case \qquad \left.\begin{array}{l}\omega(u_2) = 6\\ \omega(u_3) = 4\end{array}\right\} 2nd\ case \qquad \omega(p) = 4C\ \ 3rd\ case$$

Nodes within the shaded regions will experience the same degradation of path length to v if the link from v to the corresponding level 1 node is removed. To decide whether $p \in S$ improves the rank, the weight of each node in the extended local topology $LT'[v]$ is computed. If $\omega(p)$ is larger than the smallest weight of the nodes in N, then p improves the rank and p replaces the node in N with the lowest weight. E.g., in Fig. 5.6 u_4 is replaced by p, changing u_4's status to being a level 2 node. In Fig. 5.6b the information about one level 2 node and two connections (barley visible edges) is lost from v's point of view, i.e., $LT'[v]$, due to the removal of u_4.

5.3.2. Connected Components to Identify Bridges

Evaluating the link quality, promoting symmetric neighbors, or minimizing metric *Len* for each node, do not guarantee connectivity. Consider Fig. 5.7 with $C_N = 2$. List $v.N$ contains u_1 and u_2. Since the weights of u_1, u_2, and p are equal to 3, v will not replace u_1 or u_2 by p. However, edge (v, p) may be a bridge, i.e., (v, p) is required to form a connected topology. Without further information it is impossible for v to know that p must be included in N. This requires some kind of global knowledge [CL10].

Algorithm 5.3 Self-stabilizing Connected Component Determination

Nodes: v the current node
Variables: $v.CC_{id}$: smallest known identifier in connected component
$v.CC_d$: distance to node CC_{id}
Constants: $K \geq n$

do
[R1] $(v.d = 0 \wedge v.CC_{id} \neq v) \vee (v.d \neq 0 \wedge v.CC_{id} = v) \vee v.CC_{id} > v$
$\vee v.d \geq K \vee (v.d > 0 \wedge (\nexists u \in N(v) : u.CC_{id} = v.CC_{id} \wedge u.d + 1 = v.d))$
$\rightarrow v.CC_{id} := v; v.CC_{id} := 0$ ▷ Reset
[R2] $\square \exists u \in N(v) : u.CC_{id} < v.CC_{id} \wedge u.CC_d + 1 < K$
$\rightarrow v.CC_{id} := \min\{u.CC_{id} \mid u \in N(v)\}$
$v.CC_d := \arg\min_u\{u.CC_{id} \mid u \in N(v)\}.CC_d + 1$ ▷ Smaller identifier
[R3] $\square \forall u \in N(v) : u.CC_{id} \geq v.CC_{id} \wedge \exists u \in N(v) : u.CC_{id} = v.CC_{id}$
$\wedge u.CC_d + 1 < v.CC_d$
$\rightarrow v.CC_d := \min\{u.CC_d \mid u.CC_{id} = v.CC_{id}\} + 1$ ▷ Smaller distance
od

We use a distributed algorithm that labels the connected components CC of the topology G_c in parallel to the main routine. The algorithm labels every node of CC with the smallest identifier of all nodes within CC (see Algorithm 5.3). In order to provide connectivity, a node selects preferentially links to nodes from a different CC. The algorithm is based on [PD02]. A node v maintains two variables: CC_{id} holds the smallest identifier of a node reachable from v and d stores the distance to that node. Upon the reception of a periodic message, a node first executes the code to update its lists. Afterwards it updates s and d with Algorithm 5.3.

There is one situation that requires special treatment: When the node with the smallest identifier leaves a CC all other nodes refer to an identifier not contained in the component. This identifier is only eliminated when a node with a smaller identifier joins CC. In case this identifier also marks another component, then the two components cannot be distinguished. Variable CC_d solves this issue, its value grows unlimitedly in the described situation. The remedy is to reset CC_d to 0 and CC_{id} to the node's identifier v if CC_d grows beyond the diameter of the network. This requires each node to know an upper bound K that needs to be larger than the diameter.

In the situation depicted in Fig. 5.7 Algorithm 5.3 enables node v to establish that

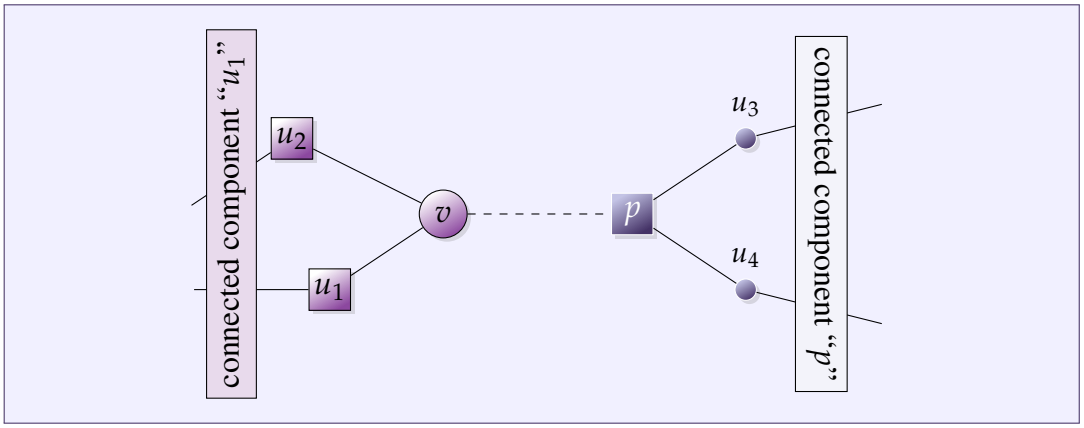


Figure 5.7.: Edge$\{v, p\}$ is a bridge $\omega(u_1) = \omega(u_2) = \omega(p)$

p belongs to a different connected component. v will therefore replace u_1 or u_2 by p to join the two components. Note that once a bridge is part of the local topology it has a high weight, because the third definition of a weight applies. Thus, it is very unlikely that a bridge will be broken again.

5.3.3. Improving the Rank of a Local Topology

With these preliminary considerations we can formulate the concept of improving the rank of the local topology of a node v by an entry p from the neighbor list N. The replacement candidate is defined by a priority based scheme with the following four categories.

Priority List	Indication and Properties
Category 1 (Symmetry)	Link (v, p) is bidirectional and N contains a unidirectional link $R_1 = \{w \in N \mid (v, w)$ is unidirectional$\}$
Category 2 (Connected Component)	$p.CC_{id} \neq v.CC_{id}$ and N contains a node w with $w.CC_{id} = v.CC_{id}$ $R_2 = \{w \in N \mid w.CC_{id} = v.CC_{id}\}$
Category 3 (Weight)	N contains a node w with $\omega(p) > \omega(w)$ $R_3 = \{w \in N \mid \omega(p) > \omega(w)\}$
Category 4 (Quality)	N contains a node w with $p.Q > w.Q$ $R_4 = \{w \in N \mid p.Q > w.Q\}$

Table 5.1.: Comparison categories to determine the rank of LT

Function $replacementCandidate(p, N)$ implements the selection of a replacement candidate for a prospective neighbor p. Let i be the smallest integer such that $R_i \neq \emptyset$. If $|R_i| = 1$, then the single node in R_i is the replacement. Otherwise, the

remaining categories are used to determine the replacement candidate, i.e., to narrow R_i down to a single node. Let $j > i$ be minimal, such that, $R_i \cap R_j \neq \emptyset$. If $|R_i \cap R_j| = 1$, then the single node in $R_i \cap R_j$ is chosen. Otherwise, the process is repeated, i.e., $R_i \cap R_j \cap R_k$ is considered. If the last category still does not give a unique candidate the most recent entry is selected. If $R_i = \emptyset$ for $i = 1,\ldots,4$ the function returns *null* and none of the current neighbors is replaced. Thus, p stays in S, having the opportunity to increase its rank further.

All previous considerations are necessary to achieved the goal of employing SSAs in WSNs. In the evaluation section we try to shed a light on appropriate settings for many of the presented parameters.

5.4. Evaluation

Evaluating the TCA is a difficult task as multiple tuning parameters are entangled and have various impacts depending on the current network parameters. E.g, the optimal value for the size of the neighbor list C_N is directly influenced by the density and the number of network nodes n. Furthermore, the dynamic behavior of constantly changing link qualities and, especially in real world deployments, the shutdown or reappearance of nodes, have colossal effects on any TCA.

This chapter presents the metrics used to analyze NORMAN. Followed by graphs showing advantages and limits of our approach. The first multitude of test is giving a proof of concept for NORMAN as well as a parameter survey. Secondly, the TCA is compared to the static TCA XTC, mainly to confer the memory usage and scalability of NORMAN. Dynamic test with added and removed nodes demonstrate its agility. Moreover, the performance of two SSAs on top of NORMAN similar to the tests conducted in Section 3.5 are compared to their behavior utilizing another TCA.

5.4.1. Methodology

The simulation environment and the real world setup to evaluate our approach is presented in the following. Furthermore, the metrics adopted to assess the results are stated.

i. Simulation Environment

We implemented NORMAN and the algorithms it is compared to, i.e., XTC [WZ04], and LEEP [Gna07] using the operating system CometOS [UWT12] and performed simulations using OMNeT++ with the MiXiM framework. To simulate the wireless channel, slow-fading was applied. The used MiXiM parameters are: Logarithmic-normal shadowing: $mean = 0.5dB$, $standard\ deviation = 0.25dB$; integrated path-loss model : $alpha = 3.5$, and random bit error rate: $lower\ bound = 10^{-8}$.

CometOS enables software designers to compile the written code for simulation (e.g., OMNeT++) as well as for hardware. We used the M3 nodes in the FIT IoT-LAB in France to test NORMAN in real world deployments.

For the simulation, square grids were chosen if not stated otherwise. These grids represent a uniformly distributed network. A communication range of a one, two, and three node diameters is used, to emulate different node densities as stated in Table 5.2.

	Range	min deg	average deg	max deg	average D	max D
$n = 100$	1	2	3.6	4	9	18
	2	5	10	12	4.8	9
	3	10	21.2	28	2.9	5
$n = 225$	1	2	3.7	4	10.2	23
	2	5	10.7	12	9.8	19
	3	10	23.4	28	3.3	6

Table 5.2.: Node degree and diameter depending on communication range in 10×10 and 15×15 square grids

On hardware the node placement on the individual nodes cannot be changed as all used nodes have a fixed position (e.g., under the floor panels at an office in Grenoble). Nevertheless, among roughly 300 nodes a subset was chosen for the conducted experiments. This subset was either chosen randomly, keeping in mind that the network should be connected, or with other restrictions in mind, e.g., forcing a certain average node degree (if possible).

ii. Metrics

An important requirement for a TCA is that each node is reachable from every other node in the created topology, i.e., connectedness. Therefore, the definition of *connectedness* stated in Section 3.2.1: $|CC|/n$ is used.

Instead of the similarity used to describe the topology changes in the previous chapter, in the following the *absolute link changes* in the system are reported. This reflects more clearly on the influence of the TCA, as each link change can trigger a rule execution in a SSA in the layer above.

The *initial setup time* is measured until the system may be operational for the first time. That is the time of starting a simulation or hardware run until the connectedness reaches one for the first time.

In the final evaluation scenario the performance of two SSAs, a MIS as defined in Section 3.2 and a spanning tree algorithm, is compared when utilizing different TCAs. The spanning tree algorithm is an extension of the simple algorithm presented earlier, its details are declared in Section 6.3.1. The important extension is that each node records its direct children, instead of merely storing the parent and the distance to it. As a metric the absolute error is measured in contrary to normalized values, to better show the severity of errors. For instance, when the father and the child variable at adjacent nodes differs a fault is recorded. A second error case that is recorded are *loops* such can happen when two nodes choose each other as parents. These errors can have severe impacts on the network as a whole, e.g., when a message is forwarded, then these two, hypothetical, faulty nodes can clog up the wireless channel due to constant back and forth sending.

5.4.2. Scenario I: Proof of Concept – Parameters of NORMAN

A proof of concept that NORMAN can build a connected topology on top of a given network starts off the evaluation series. The focus of the evaluation lays on the symmetry and the stability of the created topology.

i. Scenario Setup

The parameters defining the setup of NORMAN are variated to get an understanding of their interconnection. The density and the number of network nodes are topology parameters which were variated for the simulations. Furthermore, the size of C_N was modified. All presented plots show the variation over 100 runs with changing random simulation seeds.

Each experiment starts without any connected nodes, i.e., $N = \emptyset$. A round denotes the time it takes for all nodes to broadcast a maintenance message. The round

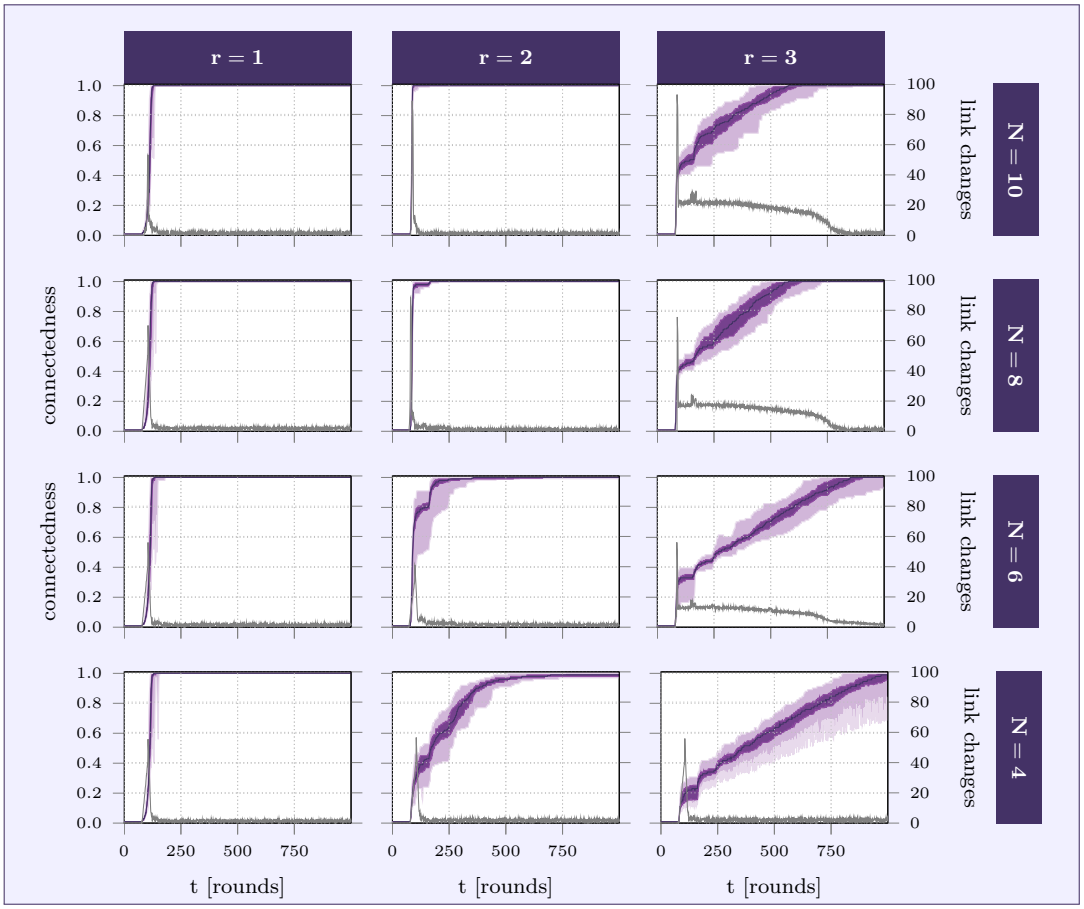


Figure 5.8.: Link change average (solid line; bottom of each plot) and connectedness over time (area plot showing percentiles over 100 experiments each); $n = 100$; variation of communication range and C_N

length depends the maximum node degree in the topology and must be chosen sufficiently large to avoid collisions and retransmissions, here the round length was one second.

ii. Results and Discussion

Figure 5.8 and 5.9 show the median, the lower and, higher quartile, with the top and bottom most graph presenting the maximum and minimum results over all experiments with the same setup. The average link change is depicted on the bottom of each plot as well.

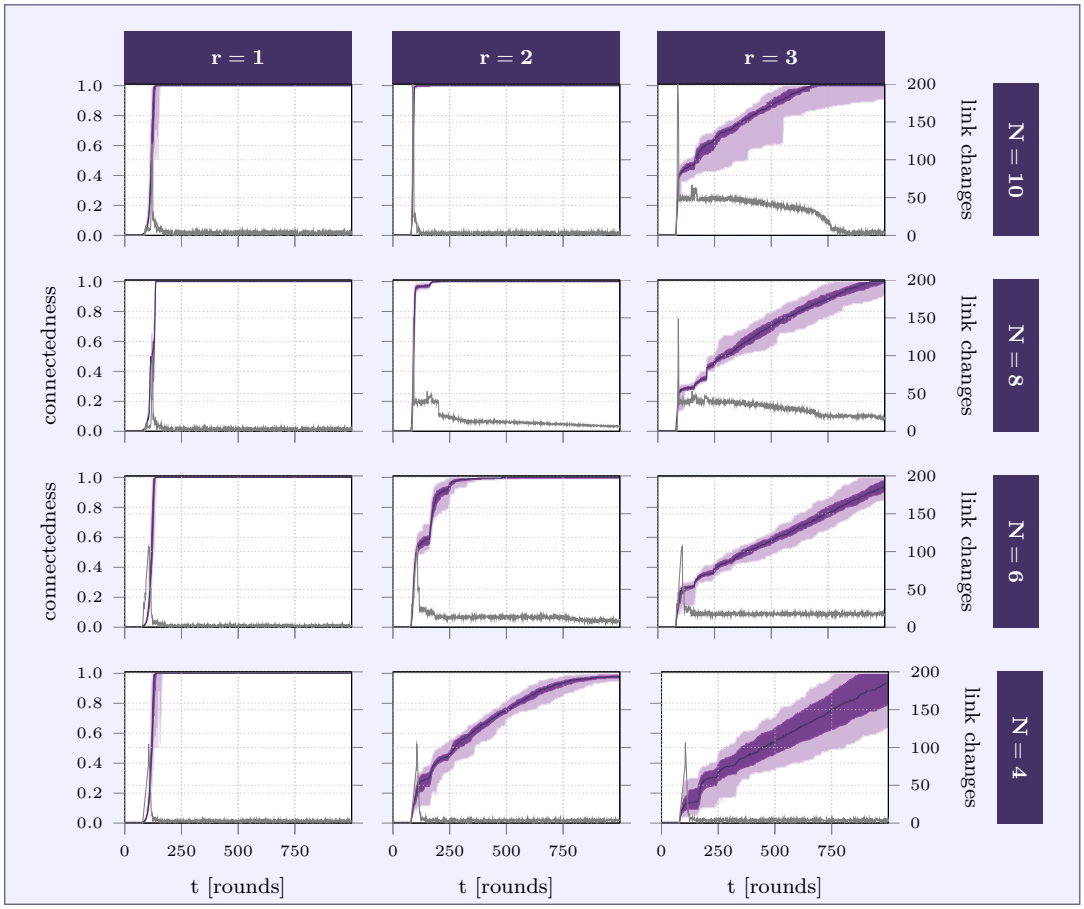


Figure 5.9.: Link change average (solid line; bottom of each plot) and connectedness over time (area plot showing percentiles over 100 experiments each); $n = 225$; variation of communication range and C_N

As expected, the communication range, i.e., the node density, plays an important role on the performance of NORMAN. The more neighbors there are to be chosen from, the harder it is to connect the network, as bi-directional connections are taken into account only. When the set of potential *good* neighbors increases, it takes significantly longer for the neighbors to mutually agree to add the other node. Especially as density and network size grow this fact becomes visible. This can be verified when comparing the size of the area depicted, the more massive the area, the more substantial the result fluctuations. Nevertheless, it can be noticed that even with optimistic settings of $C_N = 6$ the network will be connected eventuality.

Sparse networks on the other hand, e.g., $r = 1$, are easily connected and scarcely vary as their is no choice to be made among the given physical neighbor set. As

long as the physical neighborhood can be connected with a sufficiently high quality of communication links NORMAN builds a connected topology. Even with $C_N = 4$ and $n = 225$.

The results clearly identify the strengths of NORMAN, these are, fast convergence time and strong connectedness. It is evident that $C_N = 10$ is a comfortable choice when considering the size of the neighbor list. It insures that the message size for the maintenance messages stays well in the bounds defined by the wireless domain, while a high level of connectedness is achieved and maintained. The initial setup time with three to five minutes (employing one second maintenance broadcasts) is insignificant when considering the projected life time of WSNs in general.

The link change average is depicted as well. As the network needs to be stable and connected for NORMAN to build a cornerstone for SSAs, increasing the number of nodes and the density puts a strain on the algorithm. Nevertheless, the link change average stays on the bottom of the spectrum. In Scenario IV the fluctuation rate is put into perspective as the performance of selected SSAs is monitored with NORMAN as the underlying TCA.

The presented figures point out why the quality of the communication link between neighbors does not have the highest priority when it comes to choosing members for the neighbor list (according to Table 5.1). Bi-directional positions are favored, even though intuitively the quality value seems to be of utmost importance. Especially considering trading a significantly more stable link with a weaker one (but still above the given threshold Q_{tol}) that is bi-directional may seem odd at first glance. Nevertheless, this approach ensures the high connectedness of NORMAN while keeping the neighbor list size bounded.

5.4.3. Scenario II: Proof of Concept – Physical Deployment

After giving the proof of concept in simulation the algorithms performance was assessed on hardware. Investigating the behavior of the TCA over longer periods of time. Hardware environments are influenced by multiple factors a simulation usually does not account for. While the radio channel, e.g., fading, is represented well, obstructions due to physical changes, e.g, weather conditions, or people using the office space where the nodes are deployed, are not captured. Furthermore, time dependent influences are uncommon in a simulations too, e.g., the day and night rhythm of an office area.

i. Scenario Setup

To measure the behavior of NORMAN over a longer period time, while having the opportunity to be effected by the mentioned turbulences, each real world deployment test was run for 24 hours. C_N was set to ten, as the simulation showed this to be a promising value. Moreover, C_A and C_S where set to five. In Section 5.5 the influence of the assessment and standby list become more clear, for now it suffices to say that the mentioned setting does not put a strain on NORMAN, nor are those settings unrealistic. The network size was $n = 100$. The average node degree was influenced through the choice of nodes in the predefined area. Three 24 hour experiments for an average node degree of roughly $8, 12, 16$ were conducted. Maintenance messages were dispatched every second.

ii. Results and Discussion

The connectedness progresses in a fashion quite similar to the simulated test cases, as Fig. 5.10 shows. The link changes in the data plot represent the cumulative changing actions in all neighbor tables over 60 rounds. Hence, the theoretical worst case link change is $C_N \cdot 60 \cdot n = 60000$, even though this is unrealistic as each node would have to replace every single entry in the neighbor list with a different value. More realistically, after system startup each node's neighbor list is empty, thus, after the initial time needed to reach Q_{tol} each note adds neighbors, yielding, depending on the density, between 1 and 10 changes, i.e., 100 to 1000 changes in the first approximately 30 rounds. This is indicated in the plot by the very first data point. Which has thus an over-proportionally large value.

Compared to simulations the link quality between neighbors fluctuated more. The average link quality of all nodes in the neighbor list was 0.89 compared to 0.95 in simulation. Some nodes fluctuated between Q_{min} and Q_{tol} causing occasional additions and removals from and to some neighbor list, respectively. Hence, worsening the overall link change observation.

Precluding the startup phase (less than 4 minutes in the presented experiments) the link changes over time stay within a relatively strict margin between 0 and about 100 changes for all nodes over the 60 round period, i.e., less than two changes per second. We conclude that this level of change is a strong foundation for higher level (self-stabilizing) algorithms employing NORMAN as their TCA.

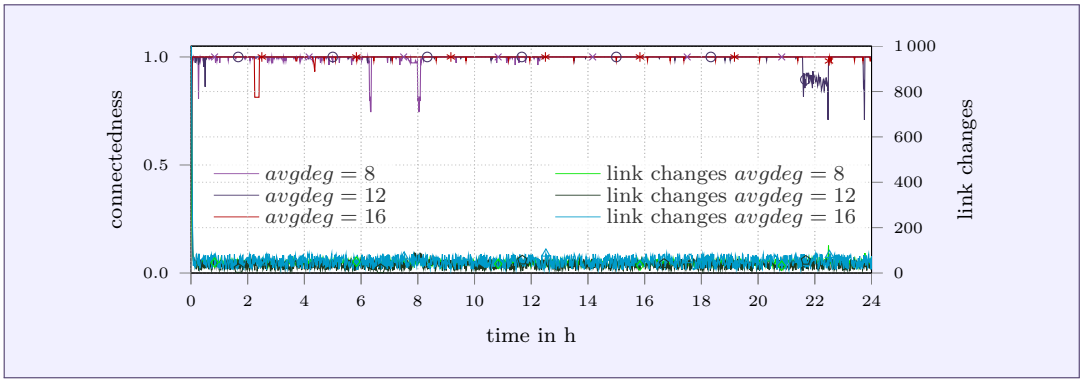


Figure 5.10.: p replaces u_4 because $\omega(u_4)$ is the minimum

Note that the experiments did not start at midnight, with other words, the x-axis of Fig. 5.10 does not point to a precise day time. Having said as much, a typical day and night rhythm for the office area the network was deployed in cannot be observed. The few drops in connectedness can be explained with nodes that where not responsive for a small amount of time, due to unknown reasons, e.g., obstruction. In the second run ($avgdeg = 12$) after 22 hours of run time a small group of nodes got disconnected for about one hour. We were not able to reconstruct what expectantly happened. Nevertheless, after this time NORMAN reconnected the nodes. Reaffirming the dynamic and robust qualities of our approach.

5.4.4. Scenario III: Comparison to XTC – Memory and Scaling

In the following our approach is compared to XTC. It is a static topology control algorithm, i.e., once each node has been assigned a neighbor, this is kept for the complete run time. Obviously, this has the advantage of not needing maintenance messages once a topology has been defined. On the other hand, dynamic behavior can only be achieved if the complete graph is reassessed by XTC which results in significant overhead and strains the wireless channel.

i. Scenario Setup

XTC is a static TCA making a direct comparison to NORMAN with respect to stability and agility impossible. Furthermore, concerning the quality of links, XTC does not make use of a minimum link quality which will cause problems in the context

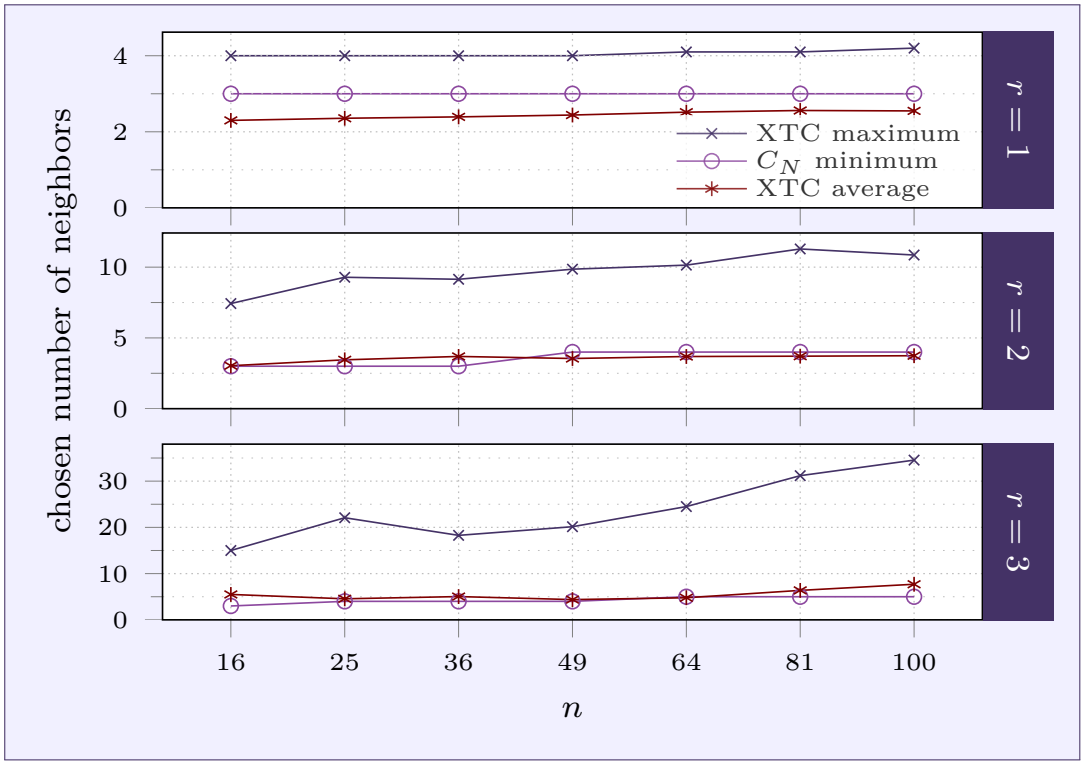


Figure 5.11.: XTC compared to our algorithm

of the usability of this approach in WSNs. On the other hand XTC does ensure a connected network.

For the communication ranges one, two, and three, square grids of 16 to 100 nodes were simulated. Mainly the quality of XTC in respect to the demand bounded degree (topology criterion 4) is addressed in the following. To do so, we measured the number of neighbors chosen by XTC and depicted the maximum and the average number. Furthermore, the minimum C_N for the given network was computed. To compare XTC with NORMAN we computed the minimum value of C_N that leads to a connected topology G_c for each setting. For this purpose we ran ten simulations for each communication range and grid. We continuously decreased the value for C_N as long as all created topologies were stably connected; we excepted this value as minimum.

ii. Results and Discussion

Figure 5.11 presents results for the average as well as the maximum number of neighbors chosen by XTC, for each communication range. The simulations revealed that the average number of neighbors chosen by XTC is about as big as the minimum value for C_N. On the other hand the maximum value of neighbors chosen by XTC grows proportional to the average node degree in the physical topology (recall Table 5.2). This demonstrates a weak point of XTC for a practical usage, its memory footprint is too large. Since the average number of neighbors and the maximum number of neighbors differ severely, it is obvious that some nodes only choose very few neighboring nodes. In fact, on a regular basis nodes only choose a single neighbor. The removal of one node, with only one neighbor, or with a very high number of neighbors, will therefore often disconnect the network.

XTC_{RLS}, an extension of XTC [FVW12], copes with this single point of failure problem by choosing additional neighbors based on their link quality. Nevertheless, the maximum chosen neighbors stay the same, yielding an unmanageable memory consumption for XTC_{RLS} as well.

As pointed out multiple times, setting $C_N = 10$ ensures NORMAN to function correctly. For $r = 2$ both TCA's require about ten neighbors. As Fig. 5.11 shows, XTC uses much higher values for their neighbor lists, e.g., in case of $r = 3$ and $n = 100$ up to 35. Hence, NORMAN is better suited, concerning memory than XTC – not to mention the dynamic properties of NORMAN.

5.4.5. Scenario IV: Providing Stability for Higher Level Algorithms

As Chapter 3 points out, the TCA is not an ends to itself but builds the foundation to the employment of SSAs in WSNs. Hence, in Chapter 3 we compared the performance of SSAs on the network versus these SSAs on top of a TCA. In the following we compare the performance of two SSAs with different TCAs.

i. Scenario Setup

We compared TinyOS's neighbor-based topology control algorithm LEEP [Gna07] to NORMAN. LEEP is widely used and comparable when considering Table 4.1. It uses a fixed neighbor list size of ten as well and adds and removes neighbors dynamically. LEEP quickly removes nodes from its neighbor table when the quality measure drops

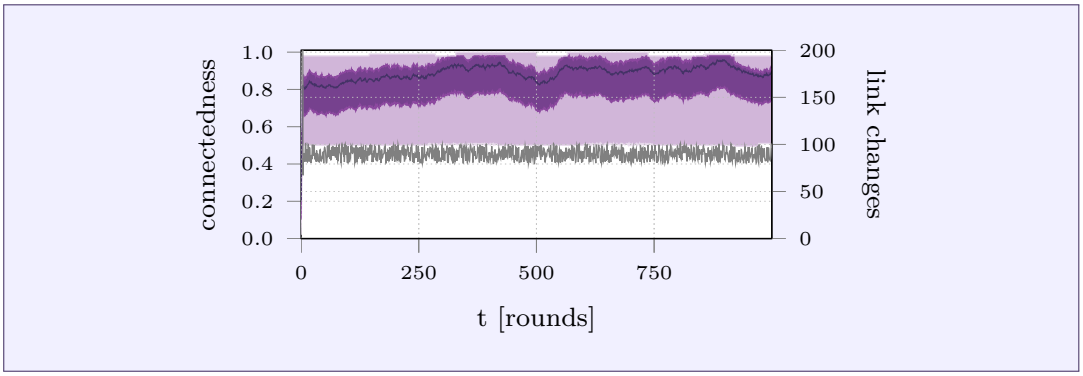


Figure 5.12.: LEEP connectedness for $n = 225$, $r = 3$, area plot and link changes

below the bidirectional ETX value of all other neighbors. The LEEP specification does not state a minimum ETX value to remove unserviceable nodes. Hence, if there are no other nodes to replace an unserviceable node, it will remain in LEEP's neighbor table.

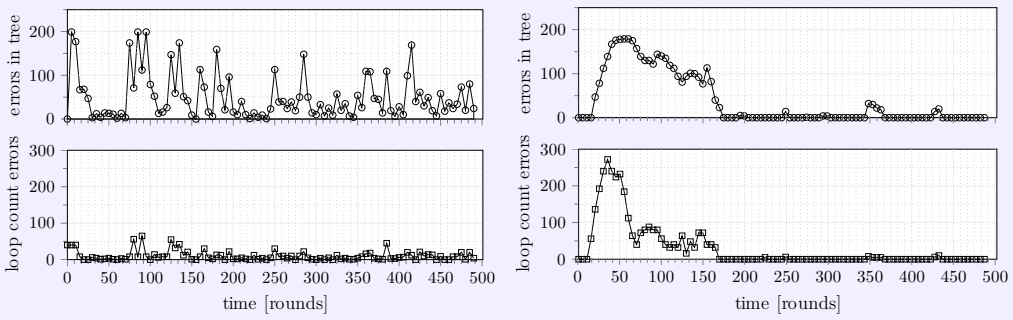
Disconnected or frequently changing topologies lead to problems for applications and self-stabilizing algorithms. Therefore, the influence on other SSAs by LEEP and NORMAN is evaluated. Thus, a MIS algorithm $\mathcal{A}_{MIS}$ and a spanning tree algorithm $\mathcal{A}_{TREE}$ analogous to Section 3.2.2 Algorithm 3.2 and 3.1 have been implemented and their respective performance has been evaluated. For Algorithm $\mathcal{A}_{TREE}$ we considered the number of nodes where the hop count in the constructed tree deviated from that in a statically constructed tree. The number of loops that emerged was used as a second metric.

For NORMAN all lists A , S, and N are initially empty. Therefore, a settling time for NORMAN to achieve a connected topology is to be expected. LEEP maintenance a single neighbor list only. When LEEPs neighbor list is empty it adds new neighbors as soon a they appear.

To reduce traffic, state messages are piggybacked with application messages. To decrease the probability of collisions a node randomly perturbs the time between two broadcasts. The MiXiM parameters are analogous to the already presented scenarios.

ii. Results and Discussion

Firstly, we evaluated the connectedness (criterion 3). In sparse networks, where the choice of neighbors is trivial, LEEP performed comparably to our approach. Dense



(a) Performance of LEEP **(b)** Performance of NORMAN

Figure 5.13.: $\mathcal{A}_{TREE}$, errors and loop errors, LEEP compared to NORMAN

networks (average node degree is larger than the size of the neighbor list) were much more challenging. Simulations with over 100 nodes in dense grids with high link variation showed strengths of NORMAN compared to LEEP. Since LEEP by default uses neighbor tables of size ten a random selection of neighbors generates a connected topology with very high probability. Nevertheless, our approach outperformed LEEP in special setups, e.g., increased density or clustered topologies only connected by bridges.

Figure 5.12 shows the connectedness of the TCA LEEP. LEEP exclusively uses the quality value of links to determine its neighbor set. Hence, continuous optimization of global properties is not ensured. Nevertheless, the network consisting of 225 nodes was occasionally connected and the median of the conducted 100 runs fluctuates in the 90 percent vicinity.

The bigger problem arises from the fact that every other node changes one of its neighbors at every iteration. Causing many link changes, that do not settle, as minor differences lead to evictions. Figure 5.13 shows the performance for a grid topology with $n = 225$ and $r = 2$ for both approaches and algorithm $\mathcal{A}_{TREE}$. The communication range was chosen to ensure that LEEP connects the network, i.e., $r = 2$.

Figure 5.13a shows that during the stabilizing phase algorithm $\mathcal{A}_{TREE}$ is disrupted constantly and errors are inevitable when employing LEEP. Loops showed up throughout the complete simulation. Merely a single typical execution is depicted to show the significant problems arising when neighborhood relations occur to frequently.

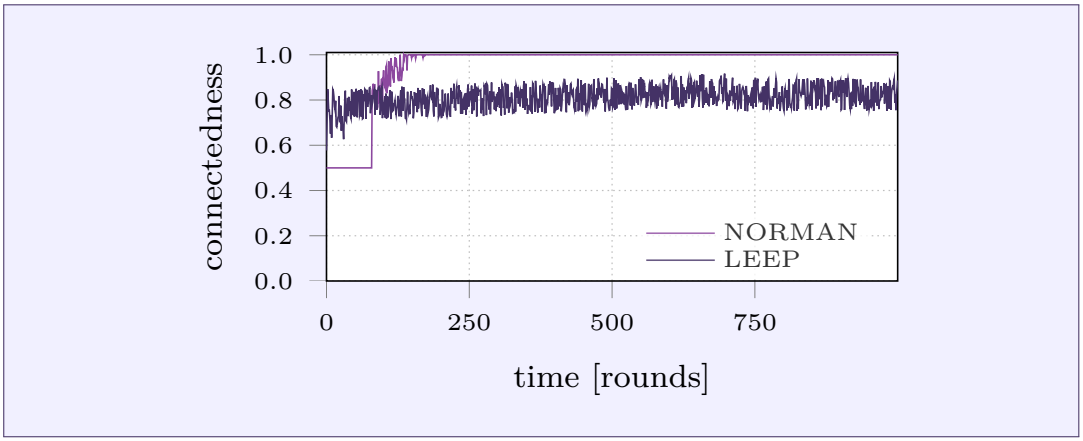


Figure 5.14.: Correctness of Algorithm $\mathcal{A}_{MIS}$ using different TCAs

The quality of the output of $\mathcal{A}_{TREE}$ is considerably improved when executed on top of NORMAN (Fig. 5.13b). During the settling time errors are even increased as the network is not connected. After this time the significantly lower link changes also cause the spanning tree algorithm to settle.

Due to the stability of NORMAN the performance of $\mathcal{A}_{MIS}$ is increased as well. This can be seen in the snap shot presented in Fig. 5.14. Due to LEEPs constant link changing the independent set never stabilizes. The MIS correctness metric is proportional to the link changes in the neighborhood relation. These findings mirror the results presented in Section 3.4 and underline once more the necessity of a stable topology to be able to use SSAs in the wireless domain.

5.5. Algorithm Analysis – Discussion

In the following the major properties defining NORMAN are put into perspective. As timers and space requirements are analyzed bounds of the presented TCA are stated.

5.5.1. Self-organization

We conjecture that NORMAN is probabilistically self-stabilizing. That is, there exist certain executions of the algorithm that could lead to the algorithm behaving as in a loop. But this will only happen if the same execution sequence is repeated over and

over. Hence, with a certain degree of randomness present any such sequence has to end with probability 1.

As we do not proof these claims we state that NORMAN is self-organizing. Without intervention it behaves dynamically to changes in the environment, removes neighbors with disintegrating link qualities and forms a connected topology. The last claim is mainly due to the *cluster rule* presented in Section 5.3.2. As nodes in different connected components are attracted to one another the global behavior of connectedness gradually increases.

The evaluations conducted substantiate this claim. As no cyclic errors occurred nor were topologies encountered that remained unconnected even after extended runtime.

5.5.2. Timings and Timeouts

The size of the assessment list and the standby list have an influence on the addition time for newly encountered neighbors. If C_A and C_S are smaller than three the initial adding time is enlarged. Values between three and five connect grid networks satisfactorily. Values larger than five have only a minor influence on the initial convergence time as the neighbor list (given the case it has a size of ten) can be filled with neighboring node identifiers immediately. Our recommendation is setting $C_A = C_S = C_N/2$ as this ensures quick response times for the setup phase and sufficient alternative neighbors to be promoted to the neighbor list. While keeping the restricted memory of hardware components in mind.

Depending on the parameters used to calculated the quality values (e.g., long term quality) defined by HoPS the eviction of dead nodes or nodes with a strongly declining link quality is influenced heavily. The faster the quality value drops below Q_{tol} the faster the node is evicted. Changing this value may be necessary for certain applications but sticking with the recommended values stated by Renner et al. [REWT11] seems appropriated.

Another parameter with the potential to enlarge the convergence time is K from Algorithm 5.3. It sets a maximal value for the diameter of the network. In a line graph $K = n$, while in a grid usually it holds that $K << n$. If K is set too big the reset time of the algorithm in a faulty case is unnecessary enlarged, while a too small value leads to errors in the algorithm.

5.5.3. Space Requirements and Scaling

The overall memory consumption of NORMAN consists of two bytes for the neighbor identifier, their according 2-hop neighbor identifier ($2C_N$ and $2C_S$), the quality value (4 bytes), the sequence number of the last received message (1 byte), and two bytes for Algorithm 5.3, times $C_N + C_S$. Thus, in total $C_N(9 + 2C_N) + C_S(9 + 2C_S) + 7C_A$. In all experiments described in Section 5.4 we used $C_N = 10$, $C_S = 5$ if not stated otherwise and $C_A = 5$, thus, 420 bytes to store the complete information about the 2-hop neighborhood. If necessary, as shown in Scenario I of Section 5.4 smaller values for C_N are often applicable. $C_A = C_S = 3$ prolong the initial settlement time only slightly, hence, reducing the minimum space requirement for NORMAN with $C_N = 6$ to less than 200 bytes.

5.6. Concluding Remarks

This chapter presented NORMAN a dynamic topology control algorithm which aims to optimize global topology properties, such as connectedness while keeping the number of neighbors of each node below a given value. NORMAN is of high practical relevance for real sensor network hardware, as will be further substantiated in the upcoming chapters. Evaluation was conducted through simulation and real sensor network hardware. Comparing NORMAN to other TCAs. Simulations have shown that the algorithm provides a good compromise between agility and stability. The hardware evaluation revealed its dynamic qualities in real world deployments.

Chapter 6

Virtual Ring

A Straightforward Routing Structure

In the following we show how to construct a virtual ring in a self-stabilizing fashion. With the forced stability of the TCA NORMAN G is considered to be *reasonably stable*. That is, we assume the topology to be stable enough to ensure any of the presented algorithms to converge eventually. Perturbances, node addition, and removals continue to be supported, it is merely assumed that their occurrences rate allows for sufficiently long stable periods. As shown in Chapter 3 such periods exist commonly in WSNs employing a topology control algorithm.

6.1. Overview

This section formally defines the virtual ring. Furthermore, the structure itself is motivated and its properties are described.

6.1.1. Characterization

A virtual ring is a routing structure where a closed path over all nodes in G is specified. Formally a virtual ring is defined as:

Definition 6.1. *A sequence $\mathcal{R} = \langle v_0, v_1, \ldots, v_{l-1} \rangle$ of nodes $v_i \in V$ is called a virtual ring if each node in V appears at least once in $\mathcal{R}$ and if each v_i is a neighbor*

of v_{i+1} (indices are taken modulo l). The value of l is called the length of $\mathcal{R}$. For $v \in V$ each i with $v = v_i$ is called a position of v. The list of positions for a virtual ring $\mathcal{R}$ of a node v is denoted by $Pos_R(v)$.

In the following the index $\mathcal{R}$ is omitted, because *positions* always refer to the same fixed virtual ring. Every connected graph has at least one virtual ring. Note that $l = \sum_{v \in V} |Pos(v)|$. For a virtual ring of short length the sets $Pos(v)$ must be small. Only Hamiltonian graphs have virtual rings with $|Pos(v)| = 1$ for each $v \in V$, this means, $l = n$. Thus, in general there will be nodes v with $|Pos(v)| > 1$.

6.1.2. Motivation and Objectives

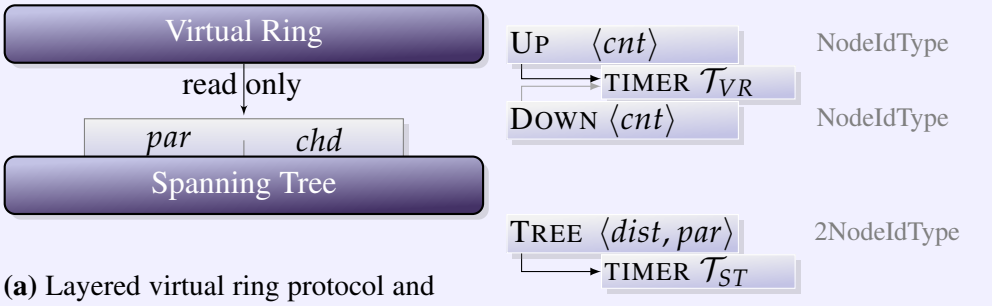
Virtual rings as a routing structure have the advantage that complex forwarding tables can be omitted. Merely the next node has to be specified to route messages to. This suffices to guarantee routing from any node to any other in the network. Paths on the other hand can become quiet long, leading to an increased message loss probability and to prolonged latency.

Finding a virtual ring of small size mitigates this issue. Determining a virtual ring of minimum length is a variant of the Traveling Salesman Problem, thus, is NP-complete. In the following two approaches are presented that bound a virtual ring to a maximum length. The second algorithm may find virtual rings of minimum length, while the first generates rings of constant size.

6.2. Related Approaches

Different distributed algorithms for creating a ring topology exist, e.g., Virtual Ring Routing (VRR) [CCN+06], the Iterative Successor Pointer Rewiring Protocol (ISPRP) [CF+05] and, developed for use in Peer to Peer networks, the Ring Network protocol [SR05]. All three are intended for use in routing. The three algorithms organize nodes in virtual rings in order of increasing node identifiers.

VRR and ISPRP work on top of the link layer and neighboring nodes in the virtual ring are not necessarily neighbors in the underlying graph. For this reason, it is necessary to set up and maintain routes between each virtual neighbor. Also, communication between two neighbors on the virtual ring may require sending messages over multi-hop routes. Maintaining this structure in a WSN is cumbersome.



(a) Layered virtual ring protocol and shared variables

(b) Message types, data types, and timers

Figure 6.1.: Virtual ring protocol overview

For sufficiently dense random networks Levy et al. [LLP04] provide an algorithm that has a high probability to find a Hamiltonian Cycle in random graphs $G(n,p)$ if $p = \omega(\sqrt{logn}/n^{1/4})$. It terminates in $O(n)$ rounds, whereas the expected number of rounds is $O(n3/4+\epsilon)$. A Hamiltonian Cycle forms the shortest possible ring. However, the existence of a Hamiltonian Cycle is not guaranteed and if one exists it is not certain that it is detected by the algorithm. Additionally the algorithm requires a synchronous scheduler and is distributed but not self-stabilizing.

6.3. Virtual Ring Construction – Tree Based Approach

The proposed protocol uses two different algorithms, implemented in a two layer approach. A spanning tree is constructed. It forms the basis for the position algorithm, that ultimately defines the positions of each node in the virtual ring. We use collateral composition to join the self-stabilizing spanning tree algorithm and the position determination algorithm. Figure 6.1 shows the layers and the corresponding Application Programming Interface (API).

6.3.1. Spanning Tree Layer

This layer maintains a spanning tree of the graph defined by the TCA. The implementation in Algorithm 6.1 is based on a self-stabilizing algorithm to construct a breadth-first search (bfs) tree by Huang and Chen [HC92], it was adopted to the mes-

sage passing model . We assume the existence of a dedicated root node. Alternatively, with a leader election algorithm a root node can be determined (e.g., Algorithm 5.3).

i. Alternate Approaches for Spanning Tree Construction

Many different self-stabilizing spanning tree algorithms with various features have been proposed, Gärtner et. al give a compendious survey [Gär03]. Most algorithms try to minimize the memory stored on a node [DJ15], or minimize the theoretical worst case move convergence time [KK06]. From a practical point of view such optimizations carry little weight, as tree algorithms are usually not the bottleneck when it comes to memory consumption or stabilization times.

ii. Employed Approach

A broadcast message has the form:

$$\text{TREE}\langle dist, par\rangle \quad .$$

It contains a node's current distance $dist$ to the root and the identifier par of its parent. Each node broadcasts with period $\mathcal{T}_{ST}$ the message $\text{TREE}\langle dist, par\rangle$. Upon reception of a message $\text{TREE}\langle dist, par\rangle$ not originating from a current child, a node v checks whether it has to update the local variables $nextPar$ and $nextDist$. If the message indicates that the sender considers v its parent, then the node adds the sender to its set of temporary children $NextChd$. Within an interval of length $2\mathcal{T}_{ST}$ the three local objects, $nextDist$, $nextPar$, and $NextChd$ replace the currently stored data.

Proof sketch. The tree T_{par} spun distributively by all par variables is a spanning tree over the connected graph G. T_{par} is acyclic as each node only has one variable par and only chooses nodes with a lower distance or if equal, smaller identifiers. If each node has a unique identifier T_{par} cannot be cyclic. Should an error occur, it is fixed by the periodically distributed information, which is collected for $2\mathcal{T}_{ST}$. That is, the temporary values overwrite all erroneous entries. If the error is in the distance variable it takes round to fix errors at level 1 nodes (distance 1 from the root). Since the graph is connected there exists at least one level 1 node. Considering each level 1 node as the root of its subtree, allows to recursively apply the same reasoning to the previous step, until the leaves are reached. Hence, an error is fixed after $\mathcal{O}(D)$ rounds, while each round is $\mathcal{T}_{ST}$ time steps long. □

iii. Concluding Remarks Concerning Spanning Tree Algorithms

The presented tree algorithm ensures a shortest path from each node to the root. Symmetry breaking is accomplished trough prioritized identifiers, i.e., nodes with smaller identifiers are preferred. Self-stabilization is accomplished trough periodic message dispatch, and temporary reevaluation of the neighborhood.

Algorithm 6.1 Self-stabilizing Spanning Tree

Variables:	*Chd*	set of children of node v
	par	parent of node v
	dist	distance to root
Parameter:	*root*	Boolean indicating that the current node is the root node

loop with period $\mathcal{T}_{ST}$
 if *root* **then**
  $dist := 0$
  $par := 0$
 end if
 broadcast(TREE$\langle dist, par\rangle$)
end loop

---

loop with period $2\mathcal{T}_{ST}$
 par := nextParent
 dist := nextDist
 Chd := NextChd
 nextParent := MAX_VALUE
 nextDist := MAX_VALUE
 NextChd.removeAll()
end loop

Upon reception of: TREE$\langle distS, parS\rangle$ from s by v:
 if $parS \neq v$ **then** ▷ Msg. not from child
  if $distS + 1 < nextDist$ **then**
   $nextDist := distS + 1$
   $nextPar := parS$
  end if
  if $distS + 1 = nextDist \wedge parS < nextPar$ **then**
   $nextPar := parS$
  end if
 else ▷ Msg. from child
  $NextChd.add(s)$
 end if

6.3.2. Virtual Ring Layer

A depth-first search (dfs) traversal of the created tree, where every node visit is recorded with an incremented value, determines the positions on the virtual ring. In this case a node v has as many positions as v has neighbors in the spanning tree, i.e., $l = 2(n-1)$. The maximum number of maintained neighbors is regulated by the constant C_N of NORMAN, hence, it accommodates to the restricted memory

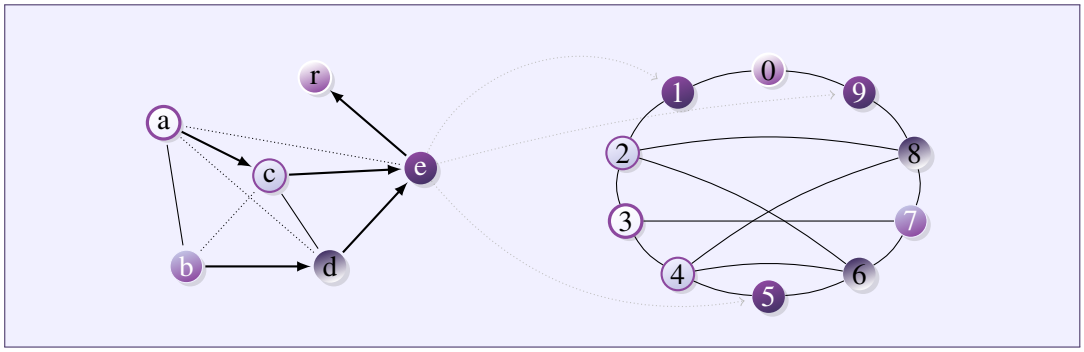


Figure 6.2.: Example topology, bold directed edges define spanning tree (left). Corresponding virtual ring graph (right)

space of the sensor nodes. The length of the resulting virtual ring is independent of the spanning tree. Figure 6.2 shows a spanning tree (bold edges) for a topology with six nodes (left). The basic idea was published in [HR87] and we found an elegant distributed and self-stabilizing way to implement it.

i. Implementation

The positions on the virtual ring are numbered beginning with position 0. Each node maintains a list P with its own positions and a table R with all positions of all its neighbors sorted by positions for efficiency. The smallest position of each node is called the *home position*.

The virtual ring follows the path of a dfs traversal of the tree. Instead of implementing such a traversal we compute for each node the number of nodes in the subtree rooted at each child. With the help of this value it is possible to compute the positions for each node on the virtual ring starting with Position 0, i.e., at the root. For this calculation the nodes do not need to know the total number of nodes. The computation of the positions is realized in two antidromic waves (see Algorithm 6.2). The first of these starts at the leaves. Recursively, each node tells its parent the number of nodes in its subtree sending messages of the form:

$$\text{UP}\langle cnt \rangle \quad .$$

The second wave begins at the root and proceeds towards the leaves. A node that knows its home position and the size of the subtree rooted at each child can compute

its own positions and the home positions of each child. Such messages have the form:

$$\text{DOWN}\langle pos \rangle \quad .$$

All leave nodes send message $\text{UP}\langle 1 \rangle$ with period $\mathcal{T}_{VR}$ to their parents. Similar to the tree algorithm a Boolean identifier (*Fresh*) is used to keep track of recent updates for each child. Upon receiving an UP message a node checks whether all the information of each child is up-to-date. If this is the case, the node forwards an UP message with the accumulated counts to its parent and resets the *Fresh* variable. If the root receives an UP message, then it starts the down wave by sending a DOWN message to all its children with their calculated home position (see Algorithm 6.2 for calculation). Upon the reception of such a message a node calculates the potions for the according subtree and forwards the updated DOWN message to all its children. Fault tolerance is achieved through periodic repetition of this process.

Waiting forever for a dead or removed child is avoid by the neighbor layer and the tree layer. After a neighbor is removed this node will eventually be removed from the tree layer as well. Hence, the virtual ring layer does not wait any longer, as the array *Chd* has changed.

ii. Shortcutting the Virtual Ring

For the virtual ring construction only the links in the previously defined spanning tree are utilized. To benefit from links selected by the TCA that are not used to build the virtual ring, *shortcuts* are introduced.

Definition 6.2. *Let $\mathcal{R}$ be a virtual ring. An edge (v_i, v_j) with $j \neq i+1$ is called a shortcut of $\mathcal{R}$.*

In the following the virtual ring is interpreted as a graph where the nodes correspond to the positions of the original nodes. Thus, if G has n nodes, then the virtual ring graph has $2(n-1)$ nodes. Figure 6.2 (right) shows the virtual ring graph emerging from the given topology, the selection conducted by the TCA, and the spanning tree on the left, i.e., $\mathcal{R} = \langle r, e, c, a, c, e, d, b, d, e \rangle$. The edge between node a and node b (left), for instance, results in a shortcut between positions 3 and 7 (right). A node's appearance (color/ shading/ etc.) in the topology (left) corresponds to the appearance of its position on the virtual ring (right).

Algorithm 6.2 Virtual Ring

Nodes:	v	the current node
Variables:	R	table with positions of all neighbors sorted by positions
	P	list of own positions
	$Fresh$	boolean list to indicate recently refreshed child count
	Cnt	list of number of children in subtree of each child
Interface:	par, Chd	from spanning tree : Algorithm 6.1, Fig. 6.1

loop with period $\mathcal{T}_{VR}$
 if $Chd = \emptyset$ **then**
  send$(par,$ UP $\langle 1 \rangle)$
 end if
end loop

Reception of: UP$\langle cntS \rangle$ from s:
$Cnt[indexOf(s)] := cntS$
$Fresh[indexOf(s)] := true$
if $\forall i \in Chd \mid Fresh[i] = true$ **then**
 for all $i \in Chd$ **do**
  $Fresh[i] := false$
 end for
 if $\neg root$ **then**
  send$(par,$ UP $\langle 1 + \Sigma_i^{|Chd|} cnt[i] \rangle)$
 else
  $P[0] := 0$
  for all $i \in Chd$ **do**
   $P[i] := P[i-1] + 2 \times Cnt[i]$
   send$(Chd[i],$ DOWN$\langle P[i-1] \rangle)$
  end for
 end if
end if

Reception of: DOWN $\langle pos \rangle$ from s:
$P[0] := pos + 1$
for all $i \in Chd$ **do**
 $P[i] := P[i-1] + 2 \times Cnt[i]$
 send$(Chd[i],$ DOWN$\langle P[i-1] \rangle)$
end for

To reduce traffic, the periodically broadcasted messages at the spanning tree layer are used to inform a node's neighbors about the node's position on the virtual ring, as well. For this purpose every node appends its current positions to the broadcasted TREE message and receiving nodes use this information to update their table R. This cross layer approach is only an option. Alternatively, the positions can also be broadcasted on the virtual ring layer, ensuring the separation of layers while increasing the network load.

6.4. Concluding Remarks

The virtual ring builds the main structure for the publish/subscribe system presented in the upcoming chapter. It defines an API to access the stored information which

is presented in the next chapter. The maintained list of neighbor positions including shortcut information may be interesting for other algorithms as well. Nevertheless, routing on the virtual ring along the shortcuts is not straight forward, as it has to be ensured that no duplication or cycling messages are fed into the system.

The presented algorithm builds a virtual ring which is not an overlay, neighbors in the virtual ring are actual neighbors in the underlying topology. The positions are necessary to integrate the shortcut approach, otherwise only a pointer to the *next* position on the ring is necessary to successfully route message to any node in the network.

Chapter 7

Publish/Subscribe Middleware

A Composite Inherently Fault-tolerant Data Dissemination Infrastructure

This chapter presents a middleware that provides a communication and data dissemination infrastructure. It incorporates the concepts presented in the previous chapters, motivates, and verifies them. The middleware realizes the channel-based publish/subscribe paradigm which has been identified as a valid means to asynchronously disseminate data in sensor networks or IoT deployments. The routing algorithm $\mathcal{PSVR}$, that builds the core of the system, reduces the path lengths to deliver publications and it is suitable for scenarios with a high subscriber fluctuation rate.

The middleware is self-stabilizing and eventually provides safety and liveness properties such as the guaranteed delivery of all published messages to all subscribers of the corresponding channel and the correct handling of subscriptions and unsubscriptions, while no error occurs. As throughout this work, transient message and memory corruptions, as well as dynamic network changes such as node and link removals and additions, are respected. The evaluation of the middleware, based on simulations and real deployments, shows that it has an acceptable memory footprint, scales well with the number of nodes, and has advantages with respect to existing comparable publish/subscribe systems.

7.1. Overview

In the following we motivate the middleware, with focus on envisioned technologies like the IoT. Furthermore, requirements on a publish/subscribe system for WSN are stated.

7.1.1. Introduction to Publish/Subscribe in the Wireless Domain

Self-stabilizing algorithms, be it coloring, clustering, tree construction, or independent sets are not an end in itself, they serve a higher purpose. E.g., building the routing structure for an application or forming a suitable setup to enable high level functionality.

A middleware supports developers with multiple interfaces and protocols while keeping certain architectural aspects, e.g., sensor node platforms and operating systems, transparent. For instance, the developer may have access to a function to send messages to all nodes while being oblivious to the fact that a self-stabilizing algorithm builds a tree and a MAC protocol handles message sending and delivery. This transparency also decreases the probability of errors due the abstraction of lower level constraints.

A main feature of publish/subscribe is that the communicating components are decoupled from each other. Thus, new data sources or consumers can be added or removed dynamically. All these considerations lead to the conclusion that publish/subscribe middlewares are a very promising technique to build IoT systems or other Cyber-Physical Systems. The suitability of publish/subscribe systems for the IoT has been described in prior work concerning different areas: Smart cities [SF13, BKS+14], Smart grid [ZLS12, KRS+12, HHM16], Electric vehicles [RJS+16], and Smart home [JJP+10, CSEC+09].

The publish/subscribe paradigm is renowned and commonly used in distributed computing. It as been widely studied for the Internet environment, e.g., [CRW01, CDKR06] and has been proven to be superior to simple flooding schemes where each entity rebroadcasts all received data. Flooding quickly leads to scaling issues, as the frequency of data, or the number of communication partners grow culminating in the *broadcast storm problem* [TNCS02], leading to large delays or even complete system shutdown.

In WANETs, WSNs, or other wireless communication backbones however, publish/subscribe is fundamentally different and more challenging. As previously pointed out, sensor nodes are resource-constrained with respect to memory and computation power. Furthermore, wireless connections are prone to errors, in particular, message loss, corruption, and radio interference. Wireless communication is realized using multi-hop relays introducing more challenges compared to a robust wired infrastructure. IoT systems need to be scalable with respect to message latency, bandwidth usage, and local memory consumption. Moreover, fault tolerance is another important requirement considering unreliable wireless communication. The loss of publications or subscriptions, arbitrary message delay and failing nodes need to be addressed. Thus, Internet based publish/subscribe solutions, such as IBM's MQseries or Java Message Service (JMS) are unsuitable for WSN.

A large body of research is devoted to topic-based publish/subscribe systems using overlay networks (e.g., peer-to-peer networks). These are logical networks on top of real network where links correspond to paths in the underlying network, hence, nodes may be logically connected that are multiple hops apart in the physical world. An example is the Scribe system [CDKR06] which is based on Pastry [RD01]. These overlay networks are mainly designed for connection-oriented transport such as TCP. They are only of limited value for WSNs because of the additional overhead for maintenance. The costs for maintaining the overlays during fluctuations are either not considered or are very high [VJC12, CVJ16]. Most systems rebuild the overlay from scratch after each unsubscription.

In this work we present a publish/subscribe system for WSNs suitable for scenarios with a high subscriber fluctuation rate, a scenario that is anticipated as very important for dynamic applications like the IoT. We propose a distributed data structure that implicitly defines a multicast tree for each node per subscribed topic. All leaf nodes of these trees are subscribers. Publications are forwarded over these trees leading to a high degree of concurrency. The focus is on low maintenance costs for the distributed data structure under new sub- and unsubscriptions at the expense of increased forwarding paths. This is achieved by limiting the changes to an area around the subscribing node. The size of this area shrinks with an increasing number of subscribers.

The presented publish/subscribe system provides non-masking fault tolerance with the concept of self-stabilization. Self-stabilization is achieved using the *leasing* technique [GC89], a fault-tolerant mechanism for distributed data consistency. The re-

newal of leases is triggered by periodically replaced data. In a WSN this data is shared using messages. An expired lease, i.e., after a certain time-out unrenewed data, leads to the removal of the entry. Note that, as messages can get lost, an expired lease can be voluntary, i.e, stopping to subscribe to a channel, or involuntary due to continuously lost messages.

7.1.2. Motivating Examples

This section presents IoT application scenarios taken from the literature to substantiate the suitability of a publish/subscribe system for IoT applications in various fields.

- **Smart Grid**: An important feature of Smart Grids is the ability to operate reliably and recover virtually without manual interventions. This is possible using Phasor Measurement Units (PMUs) that can provide precise measurements from different locations in the grid. These act as publishers; the power vendor companies are subscribers [KRS+12].

- **Virtual Power Plants**: A virtual power plant is a system that integrates several types of power sources, such as wind-turbines, small hydro, photovoltaics, back-up generators, batteries to act as a reliable overall power supply. In this scenario power sources are the publishers and the operators of virtual power plants subscribe to the energy sources [VZK+11].

- **Smart City**: Timetables of smart travel planners for individual passengers are subscribers to status information of public means of transportation, such as buses and trains [SF13].

- **Smart Home**: Energy sources for instance photo-voltaic appliances and power suppliers publish information about current and future energy availability and prices. Devices are the subscribers, they use this information to adapt their consumption patterns [JJP+10].

- **Electric Vehicles**: Charging stations publish their prices and occupancy and electric vehicles are the subscribers [RJS+16].

7.1.3. Additional System Requirements

The task of a publish/subscribe system is to deliver the data d published by a node to a channel c (commonly called publication) to all nodes with a subscription to c. Channels are identified by logical identifiers. The set of channel identifiers is defined prior to system start-up and their existence is known to all nodes. Nodes can, at any time and in any number, subscribe to and later unsubscribe from any number of channels. Thus, each node must support the following interface:

API Declaration 7.1 Interface of Publish/Subscribe System

```
publish(c, d)
subscribe(c)
unsubscribe(c)
```

The publish/subscribe middleware ensures the forwarding of publications to subscribers. Delivery of a publication at subscribers can be realized asynchronously via callbacks or synchronously with a message queue and polling. This aspect is handled transparently by a function *deliver*. After a finite time span following a subscription to a channel c a node must receive all publications to c until it unsubscribes from c again. After a node has unsubscribed from c, no more messages for c must be delivered to it.

The main non-functional requirements are non-masking fault tolerance, small memory footprint, scalability with respect to the number of participating nodes, and their density. As a reminder, non-masking fault tolerance requires that after a transient fault the system will within finite time again satisfy its functional requirements as described above. Transient faults can be caused by hardware or memory errors, and temporary unavailability of network links resulting in message duplication, loss, corruption, or insertion. Permanent faults and Byzantine behavior are still not considered. In addition, a system must eventually adopt to node deletion and insertion. Scalability with respect to the number of network nodes n requires that the memory demand for routing tables must be independent of n. The same holds for the node degree. Multi-hop routing requires a certain amount of knowledge about neighboring nodes. The memory to save the complete neighborhood of a node may be to demanding and needs to be accounted for. Hence, a subset of nodes, independent of the actual node degree must be specified.

Scalability with respect to the number of channels is not a requirement. Further non-functional requirements are low publication message latency and appropriate bandwidth usage. The memory footprint must satisfy the restrictions determined by the sensor node hardware.

Some authors, e.g., Jaeger [Jae08] model a broker overlay network by G where each participating entity is connected to a node of G. We dissociate this work from such assumptions. Hence, the more common model for WSNs is used, were all nodes participate in the routing process, despite their occupation (e.g., publisher, subscriber, or regular node).

7.2. Publish/Subscribe Middleware

This section introduces the publish/subscribe routing algorithm $\mathcal{PSVR}$ ($\mathcal{P}$ublish/$\mathcal{S}$ubscribe on $\mathcal{V}$irtual $\mathcal{R}$ings). After presenting the routing structure, the subscription and publication routing, and lastly the unsubscribing process is elucidated.

7.2.1. Routing Structure of $\mathcal{PSVR}$

Each node v maintains a routing structure $RS(v)$ in form of a $n_c \times n_p$ matrix, n_c denotes the number of channels and n_p the number of positions a node has on the virtual ring. A node v's position on the virtual ring are denoted by $Pos(v)$ and $n_p = |Pos(v)|$ (as described in Chapter 6). RS stores tuples in the form $\langle ns, \delta_U, nsn \rangle$. When a message for the ${c_i}^{th}$ channel is received at the ${p_j}^{th}$ position, then $RS(v)[c_i, p_j].ns$ is the position of the subscriber for channel c_i which is counter clock wise (ccw) closest to the ${p_j}^{th}$ node position (called forwarding position). The components $\delta_U, nstmp$ are used for unsubscriptions (see Section 7.2.4).

Recall that an important feature of the applied TCA NORMAN, as stated in Chapter 5, is that it allows the specification of an upper bound C_N for the number of neighbors of each node. Therefore, C_N generates a trade-off between the average route length of publications and the memory space required for routing-tables. With C_N, the former grows linear, while the later grows quadratically. Furthermore, increasing C_N introduces a higher number of shortcuts, hence, reduces path lengths.

7.2.2. Subscriptions

Subscription messages are used to maintain the routing structures RS at all nodes. Lost subscription messages do not lead to a permanent omission of publications, because the leasing technique guarantees the renewal of a subscription within time $\mathcal{T}_{Sub}$.

i. Subscription Distribution Range

In $RS(v)$ the next ccw subscriber for each position of v is stored. A newly subscribing node w requires that nodes update their routing structure. In particular a node u needs to update $RS(u)$ if and only if there exists a position $p_w \in Pos(w)$ and a position $p_u \in Pos(u)$ such that p_w is ccw in between p_u and p_u^f, where p_u^f is the according forwarding position in $RS(u)$ for p_u. That is, only the positions between a new subscriber w and the *clock wise* closest subscriber u, i.e., all nodes in the interval $[u, w)$, need to receive subscriptions from w. Figure 7.1 shows the stored next subscriber. Positions in the interval $[7, 9)$ record position 9 as next subscriber. All other positions, i.e., the positions in the interval $[9, 7)$, store position 7. Note that in this example only two subscriber exist and that publisher positions are irrelevant.

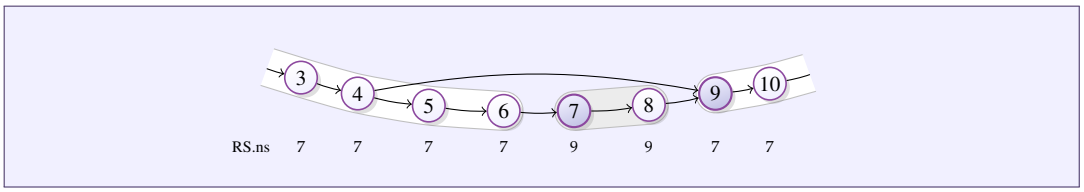


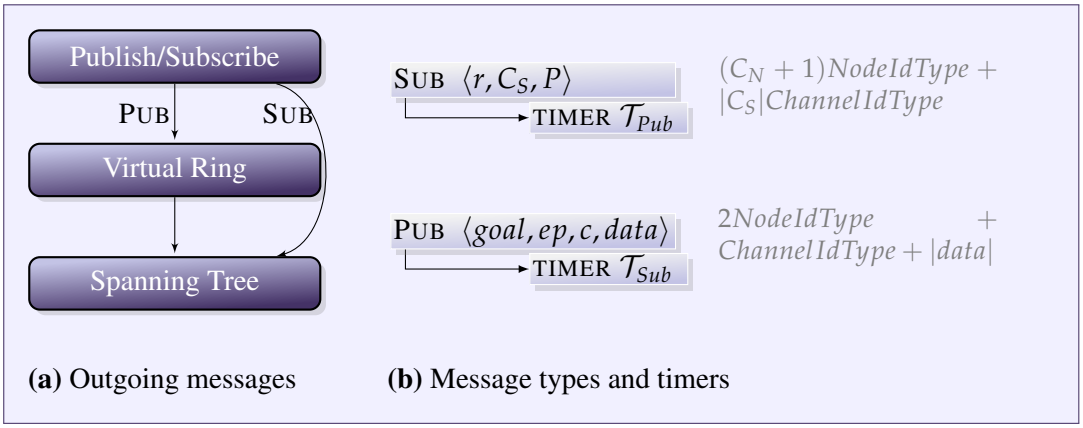
Figure 7.1.: Virtual ring with two subscribers (positions 7 and 9)

ii. Distributed Subscription Routing on the Tree

Subscription messages can be spread faster and with viewer messages if distributed over disjoint paths. Thus, we spread subscription messages (not publications) over the spanning tree built to construct the virtual ring. In the spanning tree layer subscription messages are distributed through broadcasts. For maintenance of RS messages of the form

$$\text{SUB}\langle r, C_S, P\rangle$$

are distributed with period $\mathcal{T}_{Sub}$. Figure 7.2 gives an overview of layers involved in the routing process of subscriptions and publications. Furthermore, the message



(a) Outgoing messages **(b)** Message types and timers

Figure 7.2.: Publish/Subscribe protocol stack and timers

types and timers, and the message size are depicted. The $ChannelIdType$ is considered to be a one byte sized identifier.

The spanning tree layer provides an interface for broadcasts

broadcast(Message msg)

, which is used by to the publish/subscribe layer to send SUB messages. Hence, the virtual ring layer is bypassed. The spanning tree layer acts as a broadcast filter. That is, a subscription message sent by a node in the tree is received by parent and child nodes only. Algorithm 7.1 describes the handling of subscription messages.

To avoid multiple delivery of SUB messages, they contain the previous sender r of the message, initially $r = \perp$. C_S contains the identifiers of the subscribed channels and P all positions of subscriber s, i.e., $P = Pos(s)$. Distributing the set of channels a node has subscribed to in one message instead of sending one message per channel reduces the network load.

If a SUB message from a subscriber s, forwarded by a node u, is received by a node v, then $RS(v)$ is updated using $UpdSn(c, SP)$: If there exists a position $p_i \in P$, which is ccw closer than the currently stored next subscriber ns values in $RS(v)$, then it is replaced by p_i for a given channel c. Details about $UpdSn(c, SP)$ are presented in Algorithm 7.3, because the functionality is extend to fit the needs of unsubscriptions.

An example for the operation of $UpdSn(c, SP)$ is given in the following. If $Pos(v) = \langle 5, 12, 18 \rangle$, $RS(v) = \langle 14, 14, 20 \rangle$, and the new subscriber positions are

Algorithm 7.1 Subscribing – Publish/Subscribe Layer

Constants:	$\mathcal{T}_{Sub}$	resubscribe period (leasing period)
Variables:	C_S	set of subscribed channels
	$reqRenewalC$	set of channels to be broadcasted
Functions:	$UpdSn(c, SP)$	updates table $RS(v)$ with positions P
Spanning tree layer API:		
	*broadcast(*MSG*)*	broadcasts message MSG
	numChildren()	returns number children in the tree

function *subscribe(c)*
 if $(c \notin C_S)$ **then**
  $C_S.add(c)$
  TIMER_SUB.set(0)
 end if
end function

Expiration of timer TIMER_SUB:
TIMER_SUB.set($\mathcal{T}_{Sub}$)
*broadcast(*SUB$\langle \perp, C_S, P \rangle$*)*

Upon v's reception of SUB$\langle r, C, P \rangle$
if $(r = v)$ **then**
 return
end if
for all $c \in C_S$ **do**
 $UpdSn(c, SP)$;
end for
$C := C \setminus C_S$;
if $(C \neq \emptyset \wedge$ *numChildren()* $> 0)$ **then**
 *broadcast(*SUB$\langle u, C, P \rangle$*)*
end if

$Pos(s) = \langle 3, 7 \rangle$, then the updated routing structure is $RS(v) = \langle 7, 14, 20 \rangle$, because position 7 is closer to 5 than 14.

Before forwarding a message from a node u the parameter r is altered by the forwarding node v, i.e., $r := u$. If a node w receives a message with $r = w$, then w discards the message. This ensures that a node does not resend a previously sent SUB message. Leaves of the tree and subscribers do not forward messages, they merely update their routing structure $RS(v)$.

iii. Example for Subscription Routing

Figure 7.3 shows an example, which is kept simple to increase lucidity. It shows the subscription distribution for a single channel in a line topology. The according virtual ring which does not have any shortcuts is depicted as well. When node a subscribes for the first time, node c already is a subscriber. Node a broadcasts the initial SUB$\langle \perp, \langle c \rangle, \langle 3, 9 \rangle \rangle$ message. Node c does not forward it because it is a subscriber itself. Node d and b forward the subscription and change variable r accordingly. As

a leaf, node f updates its routing structure but does not forward the message. All changes of RS induced by the subscription of node a are also depicted.

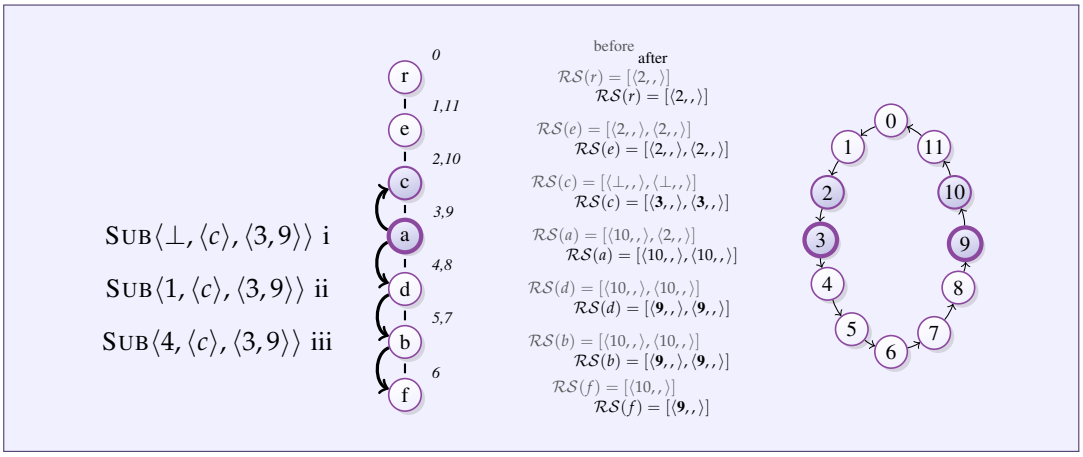


Figure 7.3.: Node a with positions 3 and 9 subscribes for the first time to channel c

iv. Final Remarks Concerning Subscription Routing

This concludes the subscription routing approach. Due to the fact that the virtual ring is based on a tree most SUB messages reach multiple node positions at the same time. Routing subscriptions over the virtual ring would be to costly as multiple nodes would resend messages, increasing the overall number of messages. The routing structure defined by RS can also not be used for SUB message routing as positions prior to a subscriber need to be informed of new subscribers but RS points in the opposite direction on the virtual ring. Finally no routing information is necessary for the applied approach and the messages size stays in the bounds expected for WSN, as long as the number of subscribed channels does not grow to big (e.g., larger than 70 considering a 100 Byte payload and 1 Byte channel identifiers). As this is not stated in the requirements in Section 7.1.3 we conclude that our approach is practical for the defined means.

7.2.3. Publications

Publication messages need to be routed to subscribers only. Hence, shortcuts can be used to skip non-subscribing nodes on the virtual ring. Furthermore, publications of nodes with multiple positions can be distributed concurrently over different paths.

The following propositions are tied to the fact that the virtual ring is built upon a tree. Under a different scheme the routing still works, but some properties, e.g., that each subscriber receives a publication only once, may not be guaranteed anymore.

i. Concurrent Routing

To explain publication routing on the virtual ring and the faced challenges when routing messages concurrently we recap tree-based routing. Each node maintains a routing table to identify branches where at least one subscriber is present. A publisher distributes messages into all such branches concurrently. The same reasoning is conducted by forwarding nodes, while avoiding to send messages back to previous senders. Trees are cycle free, hence, a publication is delivered once per subscriber. In the virtual ring, shortcuts introduce cycles. To avoid message duplication the concept of *routing into a branch* is transferred to the virtual ring. Therefore, the *end of a branch* is defined.

Nodes have multiple positions on the virtual ring, one for each neighbor in the tree. Hence, sending a message from every position in $Pos(v) = \langle p_1, \ldots, p_s \rangle$ to $p_1 + 1, \ldots, p_s + 1$, respectively is the equivalent of a tree node sending into all branches. In the routing structure RS the next subscriber for each position is stored ($RS.ns$), this reflects a node's understanding that a subscriber exists in a certain tree branch. Therefore, if a publisher knows that there is at least one subscriber in an interval $\mathcal{I} = [p_i, p_{i+1})$ for a given channel c, then it sends a publication to a *goal* position in $\mathcal{I}$. The goal position is the ccw closest one-hop reachable position to the next subscriber in $\mathcal{I}$, i.e., goal is either the next position on the virtual ring or a position reachable by a shortcut.

Received publications are delivered to all nodes subscribing to the message's channel. Regardless of the delivery, publications are forwarded to ensure that all subscribers receive it. Forwarding of publications is restricted to the interval they are sent into. To avoid sending messages beyond interval borders the endpoint ep of each $\mathcal{I}$ is attached to publication messages (recall Fig. 7.2)

$$\text{PUB}\langle goal, ep, c, data \rangle \quad .$$

Where ep is the right endpoint of $\mathcal{I} = [p_i, p_{i+1})$, i.e., $ep = p_{i+1}$. A message is neither routed to ep nor to a position beyond it. Parameters $goal$ and ep are updated at every forwarding node. The parameter $data$ represents the payload.

The start position of an interval is the current position of a node and the endpoint position is defined by the ccw next position of the same node. Multiple delivery of a publication to nodes with multiple positions in an interval is avoided.

Lemma 7.2.1. *The positions of nodes on the virtual ring are never interlaced. That is, a node v may have a position on the virtual ring which is followed by a node w's position, once another position of v appears there cannot be a further position of w.*

Proof. The virtual ring is derived from a tree using DFT. A node has multiple positions if and only if it has children in the tree. All positions of a child branch are therefore nested in between two of its parents positions. □

As Lemma 7.2.1 suggests, within a node's interval $\mathcal{I}$ may be further intervals of other nodes. For the routing procedure this means that a node forwarding a publication applies the same reasoning as a publisher to determine how to forward messages. In the tree this corresponds to branching. Each branch containing a subscriber leads to an additional message sent concurrently. The analogy in the virtual ring is as follows: Each subscriber in the interval $\mathcal{I}_f = [p_i, p_{i+1})$ with p_{i+1} ccw in between p_i and ep forwards the PUB message. That is, in the *subsection* of the virtual ring bounded by the current node position and the received endpoint position ep, independent concurrent routing is conducted. Therefore, the parameters of the PUB message are updated. The endpoint becomes p_{i+1} if p_{i+1} is ccw between p_i and ep otherwise it stays unchanged.

Algorithm 7.2 shows the handling of publications and the calculation of associated endpoints. When a node generates a publication with content d, then the *handlePub()* function is called, i.e., message PUB$\langle P[0], P[0], c, d\rangle$ is sent.

Theorem 7.2.2. *In error-free phases subscribers receive* PUB *messages exactly once.*

Proof. Once a position receives a publication message it is distributed over all possible positions with updated ep. This is equivalent to routing messages into branches the of the spanning tree that was built as a support structure. Since parameter ep of a publication is closer or equal to the next position of the same node when the message is forwarded, it is assured that no further position of the same node receives a message again. For a particular position of a node, routing is conducted using a tree edge or a shortcut. A shortcut can only be used if $RS(v)$ ensures that no subscriber is skipped. Hence, in the range of the tree between the position the shortcut leads to

Algorithm 7.2 Handling and forwarding of publications

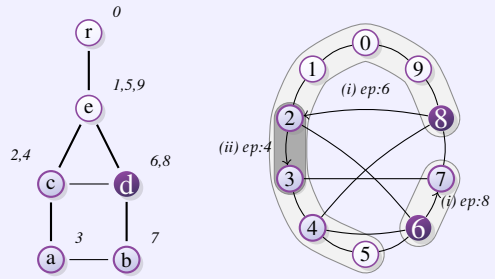
API provided by virtual ring layer (VR):

getPosClosestTo*(p, goal)*	returns largest ccw position beyond p and prior (or equal to) *goal* within neighbor positions
sendOnRing*(p, msg)*	sends message *msg* to position p
isBetween*(test, left, right)*	checks if position *test* is in ccw ring segment bounded by positions *left* and *right*. note: $isBetween(x,y,y) = true$
deliver(PL)	delivers the PL to the application

```
function publish(c, PL)
  handlePub(P[0], P[0], c, PL)
end function

Upon reception of
  PUB⟨curPos, ep, c, PL⟩
if (c ∈ C_S) then
  deliver(PL)
end if
handlePub(curPos, ep, c, PL)

function handlePub(curPos, ep, c, PL)
  for all p ∈ P do
    nextS := RS[indexOf(c)][indexOf(p)].ns
    newEp :=calcNewEP(p, ep)
    if    (isBetween(nextS, curPos, newEp))
then
    goal :=getPosClosestTo(p, nextS)
    sendOnRing(goal,
      PUB⟨goal, newEp, c, PL⟩)
  end if
 end for
end function

function calcNewEp(p, maxEp)
 i := indexOf(p)
 epIndex := i + 1 mod |P|
 if (isBetween(P[epIndex], p, maxEp))

then
  return P[epIndex]
 else
  return maxEp
 end if
end function
```

and the tree position which would be used instead (incremented current position) no subscriber exists. □

To illustrate the advantage of using shortcuts consider the topology and the virtual ring graph in Fig. 7.4a and 7.4b, respectively. In *pure* tree routing a message from node d to c is sent via node e. With $\mathcal{PSVR}$ the direct shortcut between node d and c is taken. The table in Fig. 7.4c shows the *next subscriber* and the *goal* positions. The next subscriber is the according entry in RS for the stated position. The publisher initiates two delivery paths, one for each position, i.e., for each interval. In the virtual ring in Fig. 7.4b these subsections are depicted as light gray areas. One subsection starts at position 6 the other at 8 while ep is the start position of the next



node	r	a	b	c	d	e
position	0	3	7	2 4	6 8	1 5 9
next subscriber	2	4	2	3 7	7 2	2 7 2
goal	1	4	8	3 6	7 2	2 6 0

(a) Graph **(b)** Virtual ring graph **(c)** Forwarding positions

Figure 7.4.: Illustration of the forwarding process (Nodes a,b,c: subscribers; d: publisher)

subsection, respectively. Publisher d sends messages $\text{PUB}\langle 7, 8, c, d\rangle$ from position 6 and $\text{PUB}\langle 2, 6, c, d\rangle$ from position 8. Position 2 forwards the publication in one interval with the borders $[2, 4)$ with the message $\text{PUB}\langle 3, 4, c, d\rangle$. In Fig. 7.4b this is represented by the dark gray area. In the interval $[4, 6)$ no subscriber exists, hence, no message is sent into the respective subsection.

ii. Detailed Example for Parallel Publication Routing

Figure 7.5 shows an execution of Algorithm 7.2. A virtual ring with shortcuts is depicted. Furthermore, the table next to the ring shows the routing table $RS(v)$ for each node and a single channel, with each of its own positions (Pos). A table cell represents one node, e.g., node r has the positions 0, 8, 12, and 14.

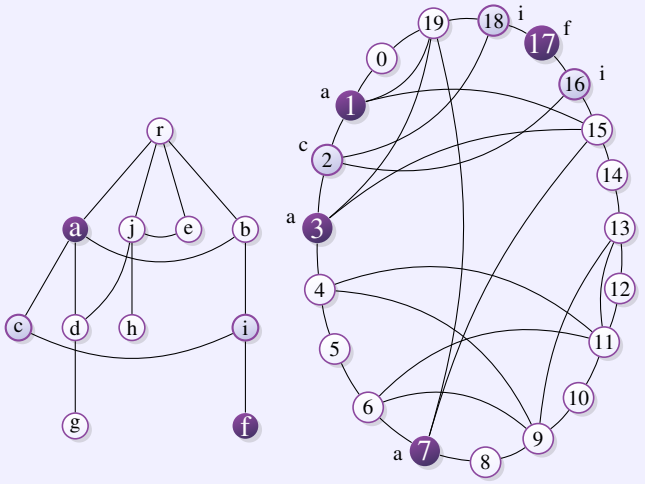
Publishers are nodes a and f and subscribers are nodes c and i. In Fig. 7.5b the two publishers can be found at positions 1, 3, and 7 (node a) and at position 17 (node f). Both publishers send one message for each interval $[p_i, p_{i+1})$ a subscriber is present (see schedule for ⓕ and ⓑ).

	from		to	ep
ⓐ	1	->	2	3
	3	not		
	7	->	15	1

	from		to	ep
ⓑ	15	->	16	18
	19	not		

	from		to	ep
ⓘ	16	not		
	18	not		

When a subscriber receives a publication, it delivers the message, then it evaluates if the message has to be forwarded. If one or more subscribers exist within the



(a) Graph

(b) Virtual ring graph

Pos	0	8	12	14	10
RS.ns	2	16	16	16	16
Pos	1	3	7		13
RS.ns	2	16	16		16
Pos	2				15 19
RS.ns	16				16 2
Pos	4	6			16 18
RS.ns	16	16			2 2
Pos	5				17
RS.ns	16				18
Pos	9	11			
RS.ns	16	16			

(c) Routing overview

Figure 7.5.: Publication routing example on virtual ring. Nodes a and f are publishers and c and i are subscribers

received ep, and if the calculated goal position does not lie beyond any of its other positions or the received ep, then a new ep is calculated and the message is altered before it is sent.

Starting with the publication from node a. Positions 2 and 15 received the publication. 2 delivers the publication but does not forward the message, as the ep forbids it. Node ⓑ, i.e., position 15 forwards the message to position 16, their the message is delivered too, and no further forwarding happens, since the next subscriber lies beyond the ep 18.

	from		to	ep
ⓕ	17	->	18	17

	from		to	ep
ⓘ	18	note		
	18	->	2	16

	from		to	ep
ⓒ	2	not		

Publications from node f are routed differently. Firstly, they are sent to position 18 – the next subscriber. Here, the ep is changed to 16, and the message is routed to position 2. Hence, all subscribers received their a publication from each publisher.

In the presented example, five messages are sent by $\mathcal{PSVR}$, compared to nine, using a common tree approach. The turn out depends highly on the location of short-

cuts, subscribers, and publishers. In the evaluation we will shed some light on the actual performance of $\mathcal{PSVR}$ compared to tree approaches and also discuss shortcomings of our approach.

7.2.4. Implicit Unsubscription Handling

A node v that ends a subscription to a channel removes the respective channel identifier from C_S. If C_S becomes empty, then v ceases to send SUB messages. Either case triggers updates in the routing structure RS at other nodes. If a value in RS has not been renewed after the leasing period $\mathcal{T}_{Sub}$, then it is identified as *stale*. Stale entries are used for routing nonetheless, i.e., for incoming publication forwarding the staleness of the entry is irrelevant. When a SUB message with a ccw closer subscriber position is received, the stale value is replaced and time-stamp δ_U (defined in Section 7.2.1 states c of value) is renewed.

If a stale value is not replaced in this way, then a temporary new next subscriber nsn is stored when the next SUB message is received. This nsn value is treated in the same way as the according stale value in RS. Initially $nsn := \perp$. When a SUB message is received nsn is set to the closest ccw subscribing position stated in the message. For every received SUB message nsn is updated to store the closest ccw subscriber position. While nsn is updated, routing for PUB messages still refers to the stale value. After an update period $T_{w/back}$ nsn replaces the stale value ns. The time-stamp δ_U is set to the current time and nsn resets to $\perp$. Algorithm 7.3 describes the details of the already mentioned $UpdSn()$ function.

Routing is correct during the whole process, that is, no subscriber is skipped. When a node unsubscribes or an error in RS occurs, it takes at most $T_{w/back}$ periods of time until RS is consistent again. The burden on memory for the presented unsubscribing scheme is manageable. For each node position the temp value and δ_U has to be accounted for, typically for each node that means $(2+4)C_N$ Bytes.

Note that $\mathcal{T}_{Sub}$ can be constant or determined during runtime, as it has a strong correlation to the length of the virtual ring. When the virtual ring is constructed the root node sends a DOWN message including starting positions of each node to its children in the tree (as explained in detail in Chapter 6 Section 6.3.2i). The root node has knowledge of tree and ring size. Hence, attaching this value to the DOWN message of the virtual ring setup algorithm can be realized conveniently. The number of nodes, i.e., the length of the ring can then be used to determine $\mathcal{T}_{Sub}$. Thereby

Algorithm 7.3 Unsubscriptions

Constants:	T_{clean}	clean timer expiration time
	$T_{w/back}$	write back period → new replaces stale
Functions:	$isStale(t_s, expT)$	return $currentLocalTime() - t_s > expT$

function $UpdSn(c, SP)$
 for all $sp \in SP$ **do**
  for all $rs_j \in RS[c]$ **do**
   if (***isBetween***$(sp, P[j], rs_j.ns)$) **then**
    $rs_j.ns := sp$
    $rs_j.\delta_U := currentLocalTime()$
   else if $(isStale(rs_j.t_s, \mathcal{T}_{Sub}))$ **then**
    if (***isBetween***$(sp, P[j], rs_j.nsn)$) **then**
     $rs_j.nsn := sp$
    end if
   end if
  end for
 end for
end function
//TIMER_CLEAN initialized on system startup
Expiration of timer TIMER_CLEAN:
TIMER_CLEAN.set(T_{clean})
for all $rs \in RS$ **do**
 if $(isStale(rs.t_s, T_{w/back}))$ **then**
  $rs.ns := rs.nsn$
  $rs.\delta_U := currentLocalTime()$
  $rs.nsn := \perp$
 end if
end for

also ensuring that all nodes use (eventually) the same $\mathcal{T}_{Sub}$ avoiding asynchronous fluctuation effects that otherwise may arise.

7.3. Evaluation

This section introduces the methodology and the metrics used to put $\mathcal{PSVR}$'s performance into perspective. Numerical simulation, simulation employing a realistic radio model to mimic wireless communication, and real world deployments have been conducted to substantiate the usability of our approach.

7.3.1. Methodology and Metrics

Before presenting the acquired results each particular algorithm used for comparison is declared. Furthermore, the metrics used to analyze the results are stated.

i. Methodology

Scenario type	Algorithm for comparison		Publication delivery	Conserved messages	Node addition
Calculation	Single bfs-tree	$\mathcal{A}_O$		IV	
	Bfs-tree each	$\mathcal{A}_M$		V	
Simulation	Flooding	$\mathcal{A}_F$	I		
	Shen's algorithm	$\mathcal{A}_S$	II	VI	
Real world			III		VII

Table 7.1.: Evaluation scenario overview

We propose seven different scenarios (I-VII) to ensure the usability of $\mathcal{PSVR}$, that the trade-off between path length and routing structure size is beneficial, and that the system is productive in real world deployments. *Publication delivery* (I-III), i.e., throughput, gives a proof of concept. The scenarios concerning *conserved messages*(IV-VI) show how the virtual ring structure with shortcuts benefits or hinders the length of routing paths compared to different tree approaches. Finally for the real world scenario we analyzed the time it takes to actually receive publication messages after a node was added to an already deployed and running system.

$\mathcal{PSVR}$ presents a compromise of size and maintenance effort for routing tables and routing path lengths. In order to assess the increase of the path lengths a comparison with two ($\mathcal{A}_O$ & $\mathcal{A}_M$) routing strategies was done algorithmically. The algorithms are executed on connected graphs $G(n,p)$ using the Erdős-Rényi model.

Single bfs-tree. $\mathcal{A}_O$ follows the common approach of a single routing tree. We chose a bfs-tree rooted at a central node.

Multiple bfs-trees. The second algorithm $\mathcal{A}_M$ computes a breadth-first tree for each node and recursively prunes leaves not corresponding to subscribers. Publications made by a node are forwarded via the corresponding bfs-tree. This strategy comes close to the optimal structure, i.e., a Steiner tree.

Flooding. $\mathcal{A}_F$ is an intuitive approach which does not need any routing structure, but lacks performance. Even though a last-received-message cache is introduced to mitigate the *broadcast storm problem*, flooding is an insufficient choice for the described system, mainly because robustness suffers with growing networks, due to interference and duplicate messages.

Shen's algorithm $\mathcal{A}_S$. The system is compared to a self-stabilizing algorithm for publish/subscribe by Shen et al. [She07]. $\mathcal{A}_S$ uses a tree to route publications and exchanges routing tables to fix errors in these tables (first introduced in Section 4.3.2).

The dynamic simulations ($\mathcal{A}_F$ & $\mathcal{A}_S$) are conducted with the OMNeT++ simulation environment. The MiXiM framework is used to simulate the wireless channel behavior. Therein, log-normal shadowing and a simple path-loss model are applied (MiXiM's setup is equivalent to the experiments in Section i). In the simulation, Shen's publish/subscribe algorithm and $\mathcal{PSVR}$ use the same MAC layer (CSMA/CA), the same topology control algorithm, i.e., NORMAN, and the same spanning tree (as defined in Section 6.3). Hence, errors due to topology changes or lost messages interfere with both approaches in a similar matter. For the simulation square grids, with the communication range of one, two, and three were chosen, following the methodology described in Section 5.4.1. Maximizing the number of symmetric neighbors and ensuring a connected network becomes more difficult as the set of suitable neighbors increases. In all upcoming tests C_N is set to 10 to ensure a connected network, as proposed in Chapter 5.

ii. Metrics

To asses the robustness of each system the publication *delivery ratio*, i.e., the percentage of received publications is measured (scenarios I-III). Furthermore, the number of sent messages necessary to deliver a publication is put into perspective. The potential message overhead in percent is calculated by $^{100B}/_{A_X} - 100$, where B is the number of messages needed by $\mathcal{PSVR}$ and A_X is the number of message needed by the approach it is compared to. Lastly, we measure the time t_{sub} it takes for a publication to be received by a new subscribing node after being turned on.

7.3.2. Results for Scenarios I & II: Publication Delivery in Simulation

First the results for the throughput of $\mathcal{PSVR}$ compared to $\mathcal{A}_F$ the most basic approach and the comparable self-stabilizing algorithm $\mathcal{A}_S$ are presented. The amount and location of publishers and subscribers is uniformly distributed over the network,

different for each run. The average delivery rate for 10000 messages sent per publisher is depicted in Fig. 7.6. Each simulation is repeated 100 times.

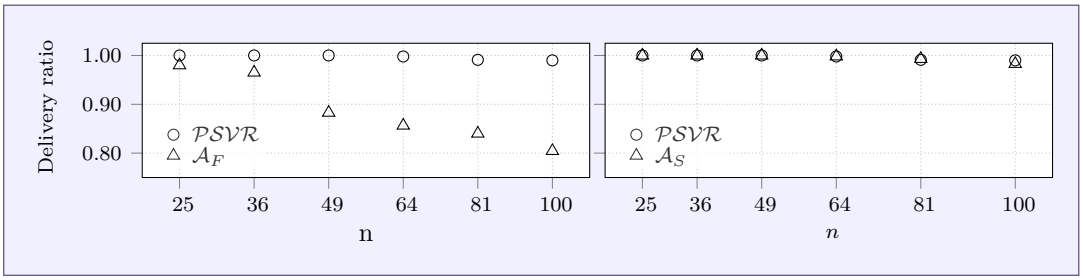


Figure 7.6.: Publications delivery, $\mathcal{A}_F$ and $\mathcal{A}_S$ compared to $\mathcal{PSVR}$

$\mathcal{A}_F$ suffers substantially with increasing density, to the point that messages get lost. Hence the extended comparison between $\mathcal{A}_S$ and $\mathcal{PSVR}$ gives more insight. The decrease of the delivery ratio of less than five percent with increasing density is due to intensification of message collisions. Furthermore, the TCA has a longer stabilization time with growing density, and more changes in the topology can be expected. This leads to errors in the spanning tree, thus, also in the virtual ring and therefore to stabilization phases where the functionality of $\mathcal{A}_S$ and $\mathcal{PSVR}$ cannot be guaranteed. Due to the TCA only links with sufficiently high quality are maintained. Therefore, the message loss probability on each link is low. If one of the approaches generates a significantly shorter route between publisher and subscriber, then the overall link quality is increased. Lost publications appear when the TCA, the tree, or in the case of $\mathcal{PSVR}$ the virtual ring are interrupted and have to be rebuilt. During this process the system behavior is arbitrary and publication loss is possible. Since self-stabilizing algorithms provide non-masking fault tolerance global knowledge of their convergence is not available [GGHP07]. Since faults in the underlying layers influence both approaches in a similar way, no substantial increase or decrease concerning the delivery rate could be found with either approach.

7.3.3. Results for Scenarios IV–VI: Overhead

The following results give an incentive to actually use $\mathcal{PSVR}$ for publish/subscribe systems. The results indicate that the difference between average path lengths decreases with increasing density and with an increase of the number s of subscribers. In Fig. 7.7 the overhead for both approaches compared to $\mathcal{PSVR}$ are depicted. For

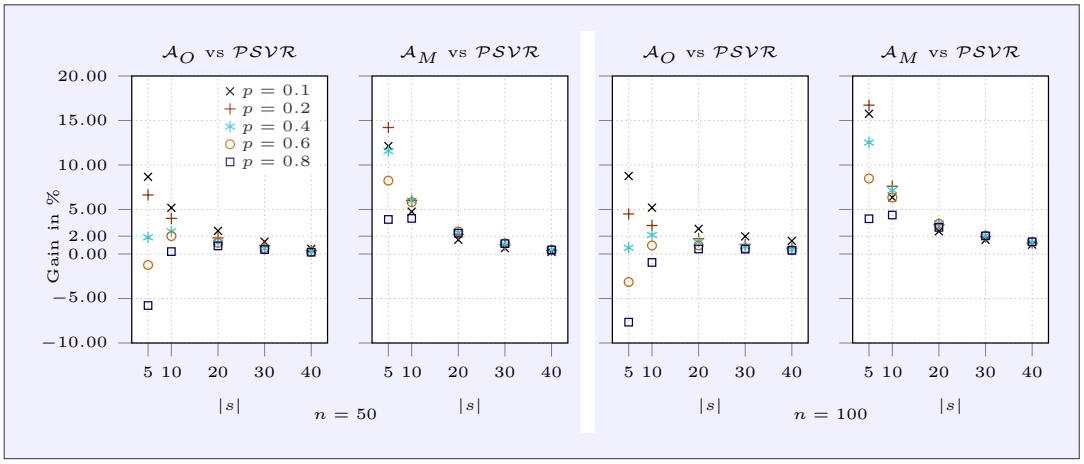
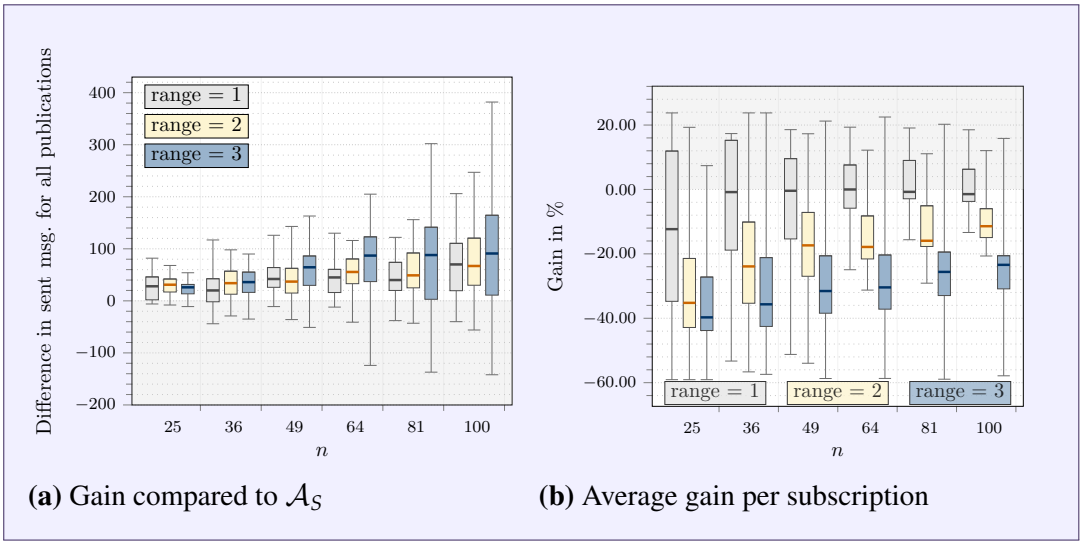


Figure 7.7.: Calculation of message overhead $\mathcal{PSVR}$ compared to $\mathcal{A}_O$ and $\mathcal{A}_M$

example for $n \leq 100$ and $s \geq 10$ the overhead of $\mathcal{PSVR}$ is less than 8 percent. We conclude that except for very small numbers of subscribers the overhead of $\mathcal{PSVR}$ with respect to path lengths is surprisingly low. With increasing density the number of shortcuts increases, allowing for shorter routing paths. Furthermore, with a growing number of nodes the overhead follows the same distribution.

Even though $\mathcal{A}_S$ also builds a fixed tree, infect, in each simulation with a radio model the exact same tree that $\mathcal{PSVR}$ is using, the message overhead varies substantially. Figure 7.8a shows the actual difference in sent messages in a dynamic environment. The route length between publishers and subscribers differs between both approaches due to the shortcuts in the virtual ring. Shorter routes lead to less sent messages, less contention, fewer sending attempts, as well as faster delivery of publications. In the following, the virtual ring, spanning tree, and neighborhood maintenance messages do not add to the total amount of sent messages. It has to be ensured that the overhead for maintaining these underlying structures is considerably smaller than the number of publications sent, otherwise the whole approach is counter productive. As mentioned in Section 3.3.3 in certain cases data can be sent piggybacked or data from multiple layers can be sent in a single packet, mitigating this problem. Furthermore, the rate of resubscriptions is set smaller than the rate of publication, otherwise maintenance is clearly more resource consuming than the actual routing process.

Reflected by Fig. 7.8a, *on average* $\mathcal{PSVR}$ decreases the number of necessary messages to deliver a publication. Negative values indicate that $\mathcal{PSVR}$ needed more



(a) Gain compared to $\mathcal{A}_S$ **(b)** Average gain per subscription

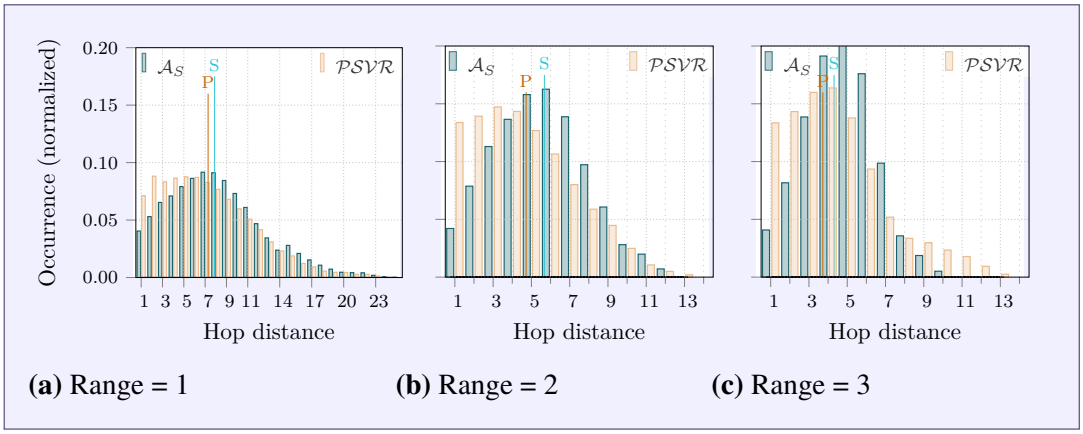
Figure 7.8.: Overall and quantitative gain in simulation with radio model, $\mathcal{A}_S$ vs $\mathcal{PSVR}$

messages than $\mathcal{A}_S$. Since with NORMAN only a fixed number of neighbors is stored, a higher number of nodes in the network leads to longer routing paths.

The quantitative overhead in percent was also calculated and is represented in Fig. 7.8b. Here a negative overhead indicates that fewer messages where sent by $\mathcal{PSVR}$ compared to $\mathcal{A}_S$. As can be seen, with increasing density a higher message saving ratio can be achieved. While with a range of 1 the saving stagnates in larger networks, with a range of 3 it stays above 20 percent. With increasing network density the number of links that can serve as shortcuts increases. The more shortcuts exist the higher the chance to find a *useful* shortcut that helps to reach subscribers quickly (Fig. 7.8b). Hindering shortcut, i.e., shortcuts that lead to longer routes are explained in Section 7.4.4.

To substantiate the claim that $\mathcal{PSVR}$ finds shorter paths between publishers and subscribers the generated routes have been analyzed. In Fig. 7.9 the hop distance for the message delivery of publications, for the three communication ranges, averaged over the previously (Fig. 7.8a, 7.8b) used topologies is depicted. $\mathcal{A}_S$ generates less one to three hop but more five to seven hop routes than $\mathcal{PSVR}$. This is also indicated by the depicted average values.

Especially direct connections ensure the usability of the publish/subscribe system, because they are relieving other nodes from forwarding messages, hence, decreasing



(a) Range = 1 **(b)** Range = 2 **(c)** Range = 3

Figure 7.9.: Hop distances of publication delivery paths. Average distance depicted by *S* for $\mathcal{A}_S$ and *P* for $\mathcal{PSVR}$; $n = 100$, $C_N = 10$

the load on the network as a whole. Interestingly, $\mathcal{PSVR}$ occasionally creates routes which are up to four hops longer compared to the routes created by $\mathcal{A}_S$. On average, this shortcoming is outweighed by the number of shorter routes created by $\mathcal{PSVR}$.

7.3.4. Results for Scenarios III & VII: Real World Deployment

Taking the theoretical concept of self-stabilization and evaluating the results of SSA in real world deployments ensures feasibility of the approach. In real world deployments unforeseeable problems, for instance memory problems that were not modeled by the simulation environment can be identified. Furthermore, the actual timing constrains can be evaluated.

i. Throughput and Robustness

$\mathcal{PSVR}$ delivers all publications as long as no error in the underlying routing structure occurs. Figure 7.10 (left) shows the delivery ratio in percent for multiple tests on a real sensor network, consisting of M3 nodes deployed at the FIT IoT-LAB (recall Chapter 3). For each number of nodes 20 tests are conducted each lasting two hours. An initial setup phase of 10 minutes is granted until publication delivery starts. Publications were dispatched every 20 s. In Fig. 7.10 (right) the same experiment is run for ten hours. Whenever an error occurs in the network the publication delivery ratio decreases, in error free phases the value can recover. The figure shows a single representative example for 10, 20, and 50 nodes each.

As the FIT IoT-LAB can be used at the same time by other people, possibly running bandwidth demanding experiments, a long term test shows the recovery strength of our approach. Using the provided monitoring tools we were able to verify that during each of our experiments surrounding nodes were occupied, even though it is not possible to identify their current tasks.

As can be seen by Fig. 7.10 (left) unsurprisingly an increasing number of nodes is more demanding on the publish/subscribe system. In short periods of time and mostly when the wireless channel is in use by other experiments the delivery ratio decreases. As can be seen in the figure on the right, the 10 minute setup period was occasionally to short for the 50 nodes experiments also causing a drop in the delivery turnout. The median shows a delivery rate of about 90 percent, which we find satisfying considering the benefits of inherent fault tolerance and the dynamic adaptability.

ii. Node Additions

Depending on timers and timeouts nodes are added to the neighborhood relation, to the tree, and virtual ring. Adding a node causes rule executions at other nodes. The current virtual ring in a network will be wrong as soon as a new node is added to the bidirectional neighborhood of another node and to the tree. Hence, routing will be disturbed. The same holds for removals of nodes. Therefore, the stabilization time after node additions and removals can be high.

After turning on a node (a subscriber in this scenario) a certain amount of time passes until this node receives its first publication. This time (t_{sub}) was repeatedly measured in the FIT IoT-LAB for 10, 20, and 50 nodes repeating each experiment 50

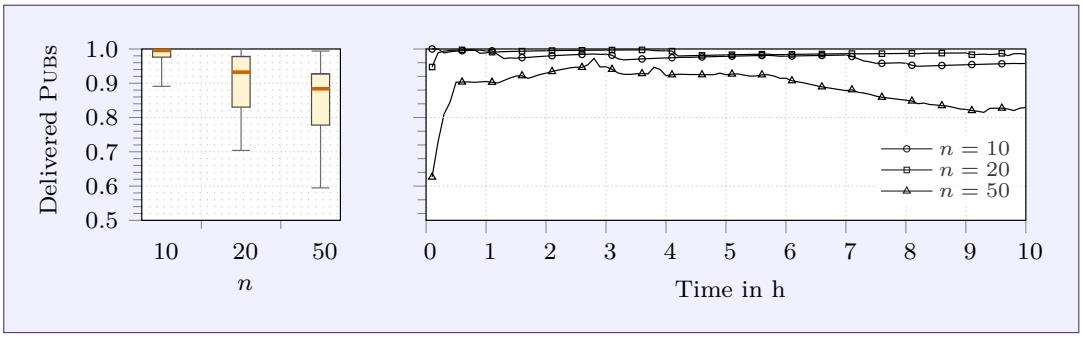


Figure 7.10.: Delivered publications average and long term test snap shot

times. We let the system run for an appropriated amount of setup-time (e.g., $n = 20$ for 10 minutes), then a node (randomly chosen) was turned on and it measured the time it took until the first publication was received.

Figure 7.11 shows the results for different values of the timers used to send maintenance messages. The FIT IoT-LAB has a bounded physical size, hence, the more nodes are used the higher the density. Therefore, with an increasing number of nodes there are nodes with a completely filled neighbor list. In such cases it takes longer to add new nodes as can be witnessed in Fig. 7.11. Interestingly, when decreasing the maintenance timers an increase of t_{sub} can be determined. This is due to lost messages as the channel gets jammed by the increased load of messages. For the 5 seconds and 8 seconds values a natural increase of the adding time is noticeable as the TCA waits to ensure the quality of its new neighbor. The TCA needs approx. 30 consecutively received message from a node to consider it a *good* neighbor. Tree and virtual ring have a settling time of $\mathcal{O}(D)$ (D is the diameter of the network).

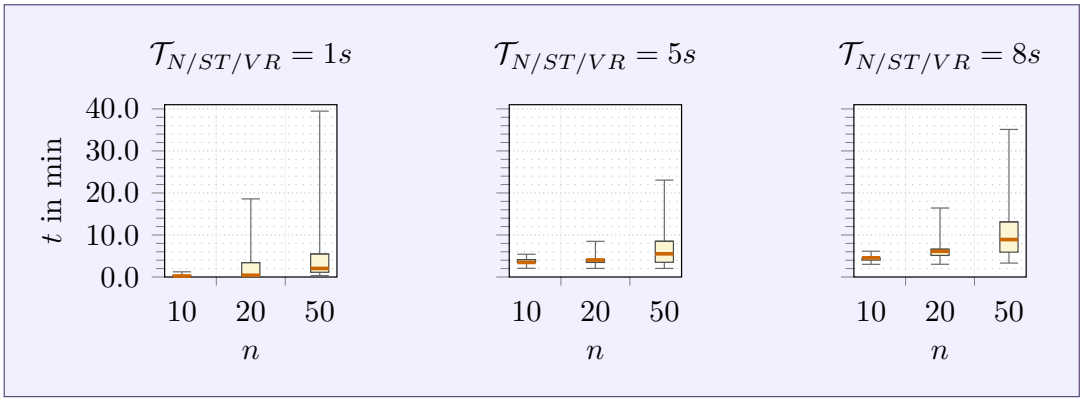


Figure 7.11.: Adding subscribers to running system; publication reception delay

A TCA and its LQE are the main bottlenecks concerning t_{sub}. The time it takes to add a node to the bidirectional neighborhood depends on the network density, the size of the TCA's *standby* and *neighbor list*, the way the quality value is determined by the LQE, and the threshold (Q_{min}) defined until a neighbor is considered *goods*. We measured the time it took to receive a publication for a node that was already part of the tree and the virtual ring. Here it depended mainly on the subscription leasing period and the publication time of node. If a subscription was lost and if the node was added shortly after the last publication, the time was maximal: In our experiments no longer than three times $\mathcal{T}_{Sub}$ plus the period of publication.

7.4. Algorithm Analysis – Discussion

We end this chapter with a discussion of the presented routing system, emphasizing what can be expected when using the discussed approaches, how space demanding the approach is, and how timeouts need to be chosen for a real world distribution to operate reliably.

7.4.1. Self-stabilizing Properties

Self-stabilization is ensured by the leasing technique. Through the periodic renewal of subscriptions the routing structure at each node is continually updated and errors are fixed. Storing a time-stamp of the last update δ_U in the routing structure $RS(v)$ ensures that stale values can be recognized. Hence, inconsistencies due to message errors, loss, or obstruction are corrected. Proper publication routing is ensured by the correctness of $RS(v)$. To unsubscribe from a channel a node removes the channel identifier from C_S, this ceases sending SUB messages. Virtual ring and spanning tree are built using self-stabilizing algorithms. They are tied together using collateral composition where no layer influences a layer below. Self-stabilizing algorithms inherently cannot locally decide if the system is in a globally correct state. Thus, in a faulty case no guarantees can be given, but that eventually the system will recover.

7.4.2. Space Requirements and Scaling

Memory Each node has at most C_N neighbors with each occupying at most C_N positions on the ring. Thus, the tables for the virtual ring R (all positions of all neighbors in C_N) and P (own positions), together require $\mathcal{O}(C_N^2)$ memory. Per channel, RS requires $\mathcal{O}(C_N)$ memory. All other variables together require $\mathcal{O}(C_N)$ memory. This leads to a total memory requirement of $\mathcal{O}(C_N^2 + cC_N)$ (c denotes the number of channels). Thus, our approach scales with the network size, but less well with c.

Subscription messages Even though there are $2(n-1)$ positions on the virtual ring, a subscription periodically generates $n - \tilde{l}$ messages, where $\tilde{l}$ are the leave nodes in the tree that are not subscribers. As subscribers and leaves do not forward messages.

Publication messages The quantity of messages per publication depends on the number of subscribers, this makes an analysis rather difficult. No more than $n-1$ PUB messages are necessary if all nodes are subscribers, analogous to the amount when applying tree routing. With a lower number of subscribers the underlying topology and the neighborhood relation defined by NORMAN can make a tremendous difference in the length of routing paths.

7.4.3. Timings and Timeouts

The self-stabilization property of the presented middleware heavily relies on the leasing technique. A critical issue with this method are the values for the timeouts. With large values a corrupted or lost entry (e.g., a subscription) may remain undetected longer than can be tolerated by an application. For example a corrupted entry in RS may lead to a situation where a subscriber will miss some publications. Increased timeouts will also raise the probability for this case. Smaller timeouts may generate unnecessary overhead, in particular a high number of messages. In case of a fault arising from lost messages, short timeouts may aggravate the situation, because they increase the load on a potentially already overloaded channel.

The middleware uses timeouts on each layer: $\mathcal{T}_{Sub}$ to renew subscriptions, $\mathcal{T}_{VR}$ to renew the positions on the virtual ring, and $\mathcal{T}_{ST}$ to renew the spanning tree. The actual values leading to a stable behavior strongly depend on the characteristics of the network and the frequency of faults. Expressing constraints about the relation between timeout values turns out to be difficult. The fact that the virtual ring is completely determined by the spanning tree suggests that the value of $\mathcal{T}_{VR}$ should be larger than $\mathcal{T}_{ST}$. This is because faults in the tree are only repaired after a period of length $\mathcal{T}_{ST}$. On the other hand, repairing corruptions in table R is independent of the spanning tree. Thus, it makes sense to choose $\mathcal{T}_{VR}$ smaller than $\mathcal{T}_{ST}$. Values of timeouts can be determined adaptively at runtime, possibly introducing new challenges as timers influence each other.

7.4.4. Negative Gain

The evaluation revealed that $\mathcal{PSVR}$ occasionally creates longer routes than $\mathcal{A}_O$, $\mathcal{A}_M$, and even $\mathcal{A}_S$, but that on average $\mathcal{PSVR}$ supersedes $\mathcal{A}_S$. The question *when do shortcuts decrease the usability of* $\mathcal{PSVR}$ is answered in the following.

Figure 7.12 (left) depicts a situation where the publisher and the subscriber placement leads to individual gains. Node 5 (position 2) is the publisher and node 3 (position 9) is the subscriber. Only a single shortcut exists between position 2 and position 6. When position 2 sends its publication it will send it to the closest known position to reach 9, i.e., over the shortcut to position 6. This will result in an overall path length of 4, whereas routing over the tree only takes 3 hops.

Figure 7.12.: Spanning tree, virtual ring positions are depicted outside of the node, one shortcut exists between position 2 and 6. Publisher and subscriber placement influence gain of $\mathcal{PSVR}$.
Left: Shen needs 3 hops $\mathcal{PSVR}$ needs 4 hops (one publisher one subscriber)
Right: Shen needs 8 hops $\mathcal{PSVR}$ needs 2 hops (two publishers two subscribers)

For the topology in Fig. 7.12 every possible subscriber and publisher deployment (4000 different placements) and the used messages were calculated. On average both approaches perform equally. In the worst case $\mathcal{PSVR}$ needs two more messages than Shen's algorithm, while in the best case $\mathcal{PSVR}$ needs six messages less. One of these cases is depicted in Fig. 7.12 (right). Position 2 and 6 are both, publisher and subscriber as indicated by the dual color circle. Sending publications over the tree leads to 8 messages that need to be sent. While $\mathcal{PSVR}$ only needs 2 messages, i.e., one from each publisher directly to the subscriber.

If the number of subscribers is close to n, then the expected gain of $\mathcal{PSVR}$ is marginal. Because for each subscriber one message needs to be sent and the amount of messages necessary converges to n, the same holds for Shen's algorithm. In scenarios where the number of subscribers is smaller than n, the gain can vary greatly. That is, as Fig. 7.8b shows, on average $\mathcal{PSVR}$ needs about one fourth less messages than Shen's algorithm. Furthermore, if the density increases, i.e., if more shortcuts exists, then $\mathcal{PSVR}$'s performance is increased too.

The worst case for both algorithms, $\mathcal{PSVR}$ and $\mathcal{A}_S$ seems to be equal. For $\mathcal{A}_S$ that is when subscriber and publisher are leaves in the maximum level of the tree.

Whereas, for $\mathcal{PSVR}$ very wide trees with shortcuts connecting the leaves are of hindrance, as the route over the root node is shorter in such cases. Nevertheless, as the virtual ring is build upon a tree, and because the maximum number of children in the tree is bounded, e.g., for NORMAN that is C_N. For any other algorithm this value is bounded as well as memory is not arbitrarily large. It is impossible that the tree grows only in width, as this would mean an an infinite number of children. But the tree grows rather by level, and this theoretically arbitrarily deep. Hence, the trees build in the presented setup favor $\mathcal{PSVR}$, as indicated in the evaluation as well.

7.5. Concluding Remarks

The presented publish/subscribe system $\mathcal{PSVR}$ is optimized for scenarios where communications links are unstable and nodes frequently change subscriptions. It is a compromise of size and maintenance effort for routing tables due to sub- and unsubscriptions and the length of routing paths. Fault tolerance is insured through the construction of self-stabilizing support structures, i.e., a spanning tree and a virtual ring. Simulations with a common radio model and verification against theoretical, closer to optimal solutions revealed that our approach gives a fair trade-off between the scalability of the support structure and the message forwarding overhead. A real world deployment confirmed its usability. The approach scales with the number of nodes and is suitable as a backbone for IoT systems.

Chapter 8

Conclusion and Outlook

8.1. Conclusion

This thesis lays the foundation for practical use of self-stabilizing algorithms in wireless error prone infrastructures. Firstly, in Chapter 3 substantial evidence is provided that directly applied self-stabilization is unserviceable in wireless sensor networks. Simulations employing a hybrid approach of simulation and hardware-evaluation revealed the necessity for an infrastructure with sparsely fluctuating neighborhood relations.

For this reason a novel self-organizing topology control algorithm NORMAN is presented in Chapter 5. This topology control algorithm creates a dynamic yet stable overlay that filters out neighbors with erratic communication links. Thus, creating a relatively stable communication infrastructure masking the existence of poor, deteriorating, or oscillating communication links and enabling the use of self-stabilizing algorithms in wireless sensor networks.

The memory consumption and the message size of NORMAN scale well with the network size and density as only a restricted set of neighbors is stored at each node. Thorough evaluation in simulation and on relevant sensor network hardware substantiates the usability of our approach. Additionally, the presented topology control algorithm aims at optimizing multiple global properties of the topology such as connectedness.

Moreover, Chapters 3 and 5 present results comparing the performance of typical self-stabilizing algorithms with NORMAN as a filter layer between MAC-layer

and self-stabilizing algorithm. Compared to other topology control algorithms NORMAN's performance is unmatched and the presented self-stabilizing algorithms perform adequately. Hence, giving an incentive to actually employ self-stabilizing algorithms in wireless sensor network deployments.

Chapter 7 presents a possible use case for wireless sensor networks and gradually extents the modeled self-stabilizing algorithms to a complete system. Presented is a publish/subscribe system $\mathcal{PSVR}$ that builds upon NORMAN and utilizes self-stabilizing algorithms exclusively. The layer structure of the defined system builds a virtual ring on top of a spanning tree. This virtual ring is used to route publications over the network, while introducing the concept of shortcuts. Thereby, reliving the channel through routing decisions, hence, skipping nodes that have not subscribed to a given channel.

$\mathcal{PSVR}$ is optimized for scenarios where communications links are unstable and nodes frequently change subscriptions. It is a compromise of size and maintenance effort for routing tables due to sub- and unsubscriptions and the length of routing paths. Fault tolerance is ensured through the construction of self-stabilizing support structures (spanning tree and a virtual ring). Simulations with a common radio model and verification against theoretical, closer to optimal solutions revealed that our approach gives a fair trade-off between the scalability of the support structure and the message forwarding overhead. $\mathcal{PSVR}$ scales with the number of nodes and uses memory conservatively enabling real world deployments that confirmed its usability further.

All presented algorithms have been implemented keeping in mind the applicability on real sensor node hardware. Especially the publish/subscribe middleware shows the usability of self-stabilization in physical deployments, hence, invalidating strong objections against the concept of self-stabilization.

8.2. Outlook

Nevertheless, some of these objections are still valid. For instance, to utilize self-stabilizing algorithms in wireless sensor networks state variables of neighboring nodes are shared constantly, i.e., messages are exchanged permanently. This usually happens in every cycle, regardless of whether the state has changed or not. As a consequence, lost or corrupted data can be repaired conveniently. However, this re-

dundancy, i.e., increased fault tolerance comes at a price: elevated energy consumption and high bandwidth usage. Since energy and bandwidth are valuable and limited resources, their waste has to be mitigated, otherwise the deployment of nodes running self-stabilizing algorithms while having limited energy resources is questionable.

One possible solution to this challenge is *prolonged cyclic data exchange* of maintenance data. The idea is to introduce an inactive phase after each state exchange of a given self-stabilizing algorithm. If the error rate is low, then with high probability no changes are missed. Possible errors are fixed as soon as the active phase begins. This frequency change can not be introduced globally, as self-stabilizing algorithms are distributed algorithms, and central administration is counterproductive. In practice, a local and dynamic adaptation of the exchange frequency is necessary. For instance by adapting the observation rate based on current conditions, e.g., the number of rule executions in a defined interval. Alternatively, sleep phase for the complete network are thinkable. It is an open question how all nodes can agree on the timing for the sleep phases or the dynamic frequency. As this could create new fluctuations and possibly processes with their own internal dynamics.

The other mayor hurdle self-stabilizing algorithms in wireless sensor networks need to overcome to be of high practical value are the confidence about temporary results. A node running a self-stabilizing algorithm cannot determine if the algorithm has stabilized globally. Hence, from a conventional application point of view self-stabilizing algorithms *cannot be trusted*. The same holds for the composition of multiple self-stabilizing algorithms, if a higher level self-stabilizing algorithm could be sure that the lower level self-stabilizing algorithm is not stable and globally incorrect it could postpone the execution of rules until the lower level algorithm is temporarily finished changing its state. This knowledge could potentially help decrease convergence time and hence, increase the capability of self-stabilizing algorithms beyond the scope presented in this work.

We conjecture that it is still a long way until self-stabilizing algorithms are commonly employed in sensor network middlewares or applications as the number of challenges that is still large. Nevertheless, the great potential in the scope of fault tolerance and the highly practical examples given in this work are hopefully encouraging further research in this area.

Bibliography

[ABM+16] S. Akkermans, R. Bachiller, N. Matthys, W. Joosen, D. Hughes, and M. Vučinić. Towards efficient publish-subscribe middleware in the iot with ipv6 multicast. In *2016 IEEE Int. Conf. on Communications (ICC)*, pages 1–6, May 2016.

[ADB+04] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Comput. Netw.*, 46(5):605–634, December 2004.

[AG93] A. Arora and M. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *Software Engineering, IEEE Transactions on*, 19(11):1015–1027, 1993.

[BFH+13] M. Ba, O. Flauzac, B. S. Haggar, R. Makhloufi, F. Nolot, and I. Niang. Energy-aware self-stabilizing distributed clustering protocol for ad hoc networks: the case of wsns. *TIIS*, 7(11):2577–2596, 2013.

[BKJ+09] N. Baccour, A. Koubaa, M. B. Jamaa, H. Youssef, M. Zuniga, and M. Alves. A Comparative Simulation Study of Link Quality Estimators in Wireless Sensor Networks. In *MASCOTS*, London, England, September 2009.

[BKM+12] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Trans. Sen. Netw.*, 8(4):34:1–34:33, September 2012.

[BKS+14] S. Bischof, A. Karapantelakis, A. Sheth, A. Mileo, and P. Barnaghi. Semantic modelling of smart city data. In *Proc. W3C Workshop on the Web of Things Enablers & Services for an open Web of Devices*, pages 1–5, 2014.

[BKY+10] N. Baccour, A. Koubâa, H. Youssef, M. B. Jamâa, D. do Rosário, M. Alves, and L. B. Becker. F-lqe: A fuzzy link quality estimator for wireless sensor networks. In *Wireless Sensor Networks*, pages 240–255. Springer, 2010.

[BLN+15] S. Beyer, S. Lohs, J. Nolte, R. Karnapke, and G. Siegemund. Self-stabilizing structures for data gathering in wireless sensor networks. In *Proceedings of the International Conference on Sensor Technologies and Applications (Sensorcomm)*, pages 1–8, Venice, Italy, August 2015.

[BLRS06] D. M. Blough, M. Leoncini, G. Resta, and P. Santi. The k-neighbors approach to interference bounded and symmetric topology control in ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(9):1267–1282, 2006.

[BOBBP13] J. Ben-Othman, K. Bessaoud, A. Bui, and L. Pilard. Self-stabilizing algorithm for efficient topology control in wireless sensor networks. *Journal of Computational Science*, 4(4):199–208, 2013.

[BTT+15] R. Banno, S. Takeuchi, M. Takemoto, T. Kawano, T. Kambayashi, and M. Matsuo. Designing overlay networks for handling exhaust data in a distributed topic-based pub/sub architecture. *Journal of Information Processing*, 23(2):105–116, 2015.

[BZZL11] B. Birand, M. Zafer, G. Zussman, and K.-W. Lee. Dynamic graph properties of mobile networks under levy walk mobility. In *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, pages 292–301. IEEE, 2011.

[CABM03] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *MobiCom*, San Diego, CA, USA, September 2003.

[CAR05] N. Carvalho, F. Araújo, and L. Rodrigues. Scalable qos-based event routing in publish-subscribe systems. In *NCA*, pages 101–108, 2005.

[CCN+06] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by dhts. *ACM SIGCOMM Computer Communication Review*, 36(4):351–362, 2006.

[CDKR06] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE J. Sel. Areas in Com.*, 20(8):1489–1499, September 2006.

[CDRT13] J. Chen, M. Díaz, B. Rubio, and J. M. Troya. PS-QUASAR: A publish/subscribe QoS aware middleware for Wireless Sensor and Actor Networks. *J. of Sys. & Softw.*, 86(6):1650–1662, 2013.

[CF+05] C. Cramer, T. Fuhrmann, et al. *Self-stabilizing ring networks on connected graphs*. Universität Karlsruhe, Fakultät für Informatik, 2005.

[CGZA06] Y.-R. Choi, M. G. Gouda, H. Zhang, and A. Arora. Stabilization of Grid Routing in Sensor Networks. *Journal of Aerospace Computing, Information, and Communication*, 3(5):214–233, 2006.

[CJV16] C. Chen, H. A. Jacobsen, and R. Vitenberg. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *IEEE/ACM Trans. on Networking*, 24(1):422–436, Feb 2016.

[CL10] A. Cornejo and N. A. Lynch. Reliably detecting connectivity using local graph traits. In *OPODIS*, pages 87–102, 2010.

[CLKB04] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 449–460. IEEE, 2004.

[CMMP08] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 26(5):748–760, 2008.

[CMTV07] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proc. 26th Annual ACM Symp. on Prin. of Distributed Computing*, pages 109–118, Portland, Oregon, USA, 2007.

[CRW01] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[CSEC+09] D. Cook, M. Schmitter-Edgecombe, A. Crandall, C. Sanders, and B. Thomas. Collecting and disseminating smart home sensor data in the casas project. In *Proceedings of the CHI Workshop on Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research*, pages 1–7, 2009.

[CVJ16] C. Chen, R. Vitenberg, and H. Jacobsen. OMen: Overlay Mending for Topic-based Publish/Subscribe Systems Under Churn. In *Proc. 10th ACM Int. Conf. on Distributed and Event-Based Systems (DEBS)*, June 2016.

[CWPE05] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin. Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing. In *MobiHoc*, Urbana-Champaign, IL, USA, May 2005.

[DGRS03] A. K. Datta, M. Gradinariu, M. Raynal, and G. Simon. Anonymous publish/-subscribe in p2p networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 8–pp. IEEE, 2003.

[Dij74] E. Dijkstra. Self stabilization in spite of distributed control, comm. *Assoc. Comput. Mach*, 17:643–644, 1974.

[DJ15] S. Devismes and C. Johnen. Silent self-stabilizing bfs tree algorithms revised. *arXiv preprint arXiv:1509.03815*, 2015.

[DMT15] S. Dubois, T. Masuzawa, and S. Tixeuil. Maximum metric spanning tree made byzantine tolerant. *Algorithmica*, 73(1):166–201, 2015.

[DÖH07] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proc. of the 5th Int. Conf. on Embedded Netw. Sensor Systems*, pages 335–349. ACM, 2007.

[Dol00] S. Dolev. *Self-stabilization*. MIT press, 2000.

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270. ACM, 1999.

[FFH+14] O. Fambon, E. Fleury, G. Harter, R. Pissard-Gibollet, and F. Saint-Marcel. "FIT IoT-LAB Tutorial: Hands-on Practice With a Very Large Scale Testbed Tool for the Internet of Things". *10èmes journées francophones Mobilité et Ubiquité, UbiMob*, 2014.

[FGJL07] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-Bit Wireless Link Estimation. In *HotNets VI*, Atlanta, GA, USA, November 2007.

[Fou16] W. Foundation. List of wireless sensor nodes, September 2016.

[FVW12] F. Fuchs, M. Völker, and D. Wagner. Simulation-based analysis of topology control algorithms for wireless ad hoc networks. In *Design and Analysis of Algorithms*, volume 7659 of *LNCS*, pages 188–202. Springer Heidelberg, 2012.

[Gär03] F. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical report, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, Distributed Programming Laboratory, CH-1015 Lausanne, Switzerland, 2003.

[GC89] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 23(5):202–210, 1989.

[GGHP07] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing distributed protocols. *Distributed Computing*, 20(1):53–73, 2007.

[GHJS03] W. Goddard, S. Hedetniemi, D. Jacobs, and P. Srimani. Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 14 pp.–, April 2003.

[Gna07] O. Gnawali. The link estimation exchange protocol (LEEP), 2007. TinyOS Extension Proposal (TEP).

[HC92] S.-T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters*, 41(2):109–117, 1992.

[Her92] T. R. Herman. Adaptivity through distributed convergence. 1992.

[Her03] T. Herman. Models of self-stabilization and sensor networks. In *Distributed Computing-IWDC 2003*, pages 205–214. Springer, 2003.

[HGM03] Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *Mobile Data Management*, volume 2574 of *LNCS*, pages 122–140. Springer, 2003.

[HHI+15] J. Hosoda, J. Hromkovič, T. Izumi, H. Ono, M. Steinová, and K. Wada. Corrigendum to: On the approximability and hardness of minimum topic connected overlay and its special instances. *Theoretical Computer Science*, 562:660 – 661, 2015.

[HHM16] M. Hoefling, F. Heimgaertner, and M. Menth. Advanced communication modes for the publish/subscribe c-dax middleware. In *IFIP/IEEE Workshop on the Management of Fog Computing and the Internet of Things (ManFIoT)*, 2016.

[HM06] S. Hadim and N. Mohamed. Middleware for wireless sensor networks: A survey. In *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, pages 1–7. IEEE, 2006.

[HR87] J.-M. Hélary and M. Raynal. *Depth-first traversal and virtual ring construction in distributed systems*. INRIA, 1987.

[HR06] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: state-of-the-art and future directions. In *Proceedings of the international workshop on Middleware for sensor networks*, pages 60–65. ACM, 2006.

[HT04] T. Herman and S. Tixeuil. A distributed tdma slot assignment algorithm for wireless sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 45–58. Springer, 2004.

[HTSC08] U. Hunkeler, H. L. Truong, and A. Stanford-Clark. MQTT-S - A publish/-subscribe protocol for Wireless Sensor Networks. In *3rd Int. Conf. on Com. Systems Soft. & Middleware*, pages 791–798, Jan 2008.

[IGE00] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. 6th Int. Conf. on Mobile Comp. & Netw.*, pages 56–67, Boston, Massachusetts, USA, 2000.

[Jae08] M. A. Jaeger. *Self-Managing Publish/Subscribe Systems*. PhD thesis, TU Berlin, 2008.

[JF09] Z. Jerzak and C. Fetzer. Soft state in publish/subscribe. In *Proc. 3rd ACM Int. Conf. on Distributed Event-Based Systems*, DEBS '09, pages 17:1–17:12, Nashville, Tennessee, New York, NY, USA, 2009. ACM.

[JJP+10] M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners. The energy aware smart home. In *5th International Conference on Future Information Technology*, pages 1–8, May 2010.

[KA06] S. S. Kulkarni and M. Arumugam. SS-TDMA: A self-stabilizing MAC for sensor networks. *Sensor network operations*, pages 186–218, 2006.

[KK06] A. Kosowski and Ł. Kuszner. A self-stabilizing algorithm for finding a spanning tree in a polynomial number of moves. In *Parallel Processing and Applied Mathematics*, pages 75–82. Springer, 2006.

[KKP99] J. M. Kahn, R. H. Katz, and K. S. Pister. Next century challenges: mobile networking for smart dust. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278. ACM, 1999.

[KRS+12] K. Kumar, M. Radhakrishnan, K. M. Sivalingam, D. P. Seetharam, and M. Karthick. Comparison of publish-subscribe network architectures for smart grid wide area monitoring. In *IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pages 611–616, Nov 2012.

[KY02] H. Kakugawa and M. Yamashita. Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. In *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*, pages 202–211. IEEE, 2002.

[Lam85] L. Lamport. Solved problems, unsolved problems and non-problems in concurrency. *ACM SIGOPS Operating Systems Review*, 19(4):34–44, 1985.

[LGCJ04] B. H. Liu, Y. Gao, C. T. Chou, and S. Jha. An energy efficient select optimal neighbor protocol for wireless ad hoc networks. In *29th Annual IEEE Int. Conf. on Local Comp. Netw.*, pages 626–633, Nov 2004.

[LKF05] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[LKNL12] S. Lohs, R. Karnapke, J. Nolte, and A. Lagemann. Self-stabilizing sensor networks for emergency management. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 715–720. IEEE, 2012.

[LLP04] E. Levy, G. Louchard, and J. Petit. A distributed algorithm to find hamiltonian cycles in\ mathcal {G}(n, p) random graphs. In *Combinatorial and algorithmic aspects of networking*, pages 63–74. Springer, 2004.

[LMH+03] D. Lal, A. Manjeshwar, F. Herrmann, E. Uysal-Biyikoglu, and A. Keshavarzian. Measurement and Characterization of Link Quality Metrics in Energy Constrained Wireless Sensor Networks. In *GlobeCom*, San Francisco, CA, USA, December 2003.

[LNST16a] S. Lohs, J. Nolte, G. Siegemund, and V. Turau. Influence of topology-fluctuations on self-stabilizing algorithms. In *2016 International Conference on Distributed Computing in Sensor Systems DCOSS 2016 Poster Abstract*, Washington, DC, USA, May 2016.

[LNST16b] S. Lohs, J. Nolte, G. Siegemund, and V. Turau. Self-stabilization - a mechanism to make networked embedded systems more reliable? In *35th Symposium on Reliable Distributed Systems (SRDS)*, Budapest, Hungary, September 2016.

[LS13] P. Leone and E. M. Schiller. Self-stabilizing tdma algorithms for dynamic wireless ad hoc networks. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[LSLY13] C.-Y. Lee, L.-C. Shiu, F.-T. Lin, and C.-S. Yang. Distributed topology control algorithm on broadcasting in wsn. *J. of Netw. and Comp. App.*, 36(4):1186–1195, 2013.

[LSNT12] S. Lohs, G. Siegemund, J. Nolte, and V. Turau. Mission statement: Tolerancezone a self-stabilizing middleware for wireless sensor netzworks. In *Proceedings of the 11th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'12)*, Darmstadt, Germany, September 2012.

[LZN10] Y. Liu, Q. Zhang, and L. M. Ni. Opportunity-based topology control in wireless sensor networks. *Parallel and Distributed Systems, IEEE Trans. on*, 21(3):405–416, 2010.

[LZZ+06] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He. ATPC: adaptive transmission power control for wireless sensor networks. In *Proc. 4th Int. Conf. on Embedded Networked Sensor Systems (SenSys 06)*, pages 223–236, 2006.

[MA06] M. M. Molla and S. I. Ahamed. A survey of middleware for sensor network and challenges. In *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pages 6–pp. IEEE, 2006.

[Mar05] P. J. Marron. Middleware approaches for sensor networks. *University of Stuttgart, Summer School on WSNs and Smart Objects Schloss Dagstuhl*, 2005.

[MBC+04] A. F. Molisch, K. Balakrishnan, D. Cassioli, C.-C. Chong, S. Emami, A. Fort, J. Karedal, J. Kunisch, H. Schantz, U. Schuster, et al. Ieee 802.15. 4a channel model-final report. *IEEE P802*, 15(04):0662, 2004.

[MFT+05] N. Mitton, E. Fleury, S. Tixeuil, et al. Self-stabilization in self-organized multihop wireless networks. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 909–915. IEEE, 2005.

[MKP10] Y. Manolopoulos, D. Katsaros, and A. Papadimitriou. Topology control algorithms for wireless sensor networks: A critical survey. In *Proc. 11th Int. Conf. on Comp. Systems and Technologies*, pages 1–10, Sofia, Bulgaria, New York, 2010.

[MM07] W. Masri and Z. Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*, pages 349–356. IEEE, 2007.

[MMPT09] F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Comp. Science*, 410(14):1336–1345, 2009.

[MT07] T. Masuzawa and S. Tixeuil. Stabilizing link-coloration of arbitrary networks with unbounded byzantine faults. *International Journal of Principles and Applications of Information Science and Technology (PAIST)*, 1(1):1–13, 2007.

[MT10] T. Masuzawa and S. Tixeuil. Stabilizing locally maximizable tasks in unidirectional networks is hard. In *2010 International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, June 21-25, 2010*, pages 718–727. IEEE Computer Society, 2010.

[NKAA14] R. Nasim, A. Kassler, I. Arko, and A. Antonic. Mobile publish/subscribe system for intelligent transport systems over a cloud environment. In *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on*, pages 187–195, Sept 2014.

[OC08] J. Ortiz and D. Culler. soda, 2008. `http://wsn.eecs.berkeley.edu/connectivity`.

[Ond14] M. Onderwater. An overview of centralised middleware components for sensor networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 2014.

[OR16] M. Onus and A. Richa. Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. *Computer Networks*, 94:307 – 317, 2016.

[PD02] V. Pandit and D. Dhamdhere. Self-stabilizing maxima finding on general graphs, 2002. IBM T.J. Watson Research Center.

[PGY+11] D. Puccinelli, O. Gnawali, S. Yoon, S. Santini, U. Colesanti, S. Giordano, and L. Guibas. The impact of network topology on collection performance. In *Wireless Sensor Networks*, volume 6567 of *LNCS*, pages 17–32. Springer, 2011.

[PS05] J. Paradiso and T. Starner. Energy Scavenging for Mobile and Wireless Electronics. *Pervasive Computing*, 4(1), January 2005.

[PST14] T. Petig, E. M. Schiller, and P. Tsigas. Self-stabilizing tdma algorithms for wireless ad-hoc networks without external reference. In *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, pages 87–94. IEEE, 2014.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[REWT11] C. Renner, S. Ernst, C. Weyer, and V. Turau. Prediction accuracy of link-quality estimators. In *Proc. 8th Europ. Conf. on Wireless Sensor Networks (EWSN)*, pages 1–16, Bonn, Germany, 2011.

[RGNH13] R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM, 2013.

[RJS+16] J. Rivera, M. Jergler, A. Stoimenov, C. Goebel, and H. Jacobsen. Using publish/subscribe middleware for distributed ev charging optimization. *Comput. Sci.*, 31(1-2):41–48, May 2016.

[RKM02] K. Römer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002.

[San05] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, June 2005.

[SCL+07] M. Senel, K. Chintalapudi, D. Lal, A. Keshavarzian, and E. J. Coyle. A Kalman Filter Based Link Quality Estimation Scheme for Wireless Sensor Networks. In *GlobeCom*, Washington, DC, USA, November 2007.

[SF13] R. Szabó and K. Farkas. Publish/subscribe communication for crowd-sourcing based smart city applications. In *Information and Communication Technologies - International Conference*, volume 2, pages 314–318. EDIS - Publishing Institution of the Univ. Zilina, 2013.

[SG10] H. Sethu and T. Gerety. A new distributed topology control algorithm for wireless environments with non-uniform path loss and multipath propagation. *Ad Hoc Networks*, 8(3):280–294, 2010.

[SGV+05] E. Souto, G. Guimares, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner. Mires: A publish/subscribe middleware for sensor networks. *Personal Ubi. Comput.*, 10(1):37–44, 2005.

[She07] Z. Shen. *Techniques for building a scalable and reliable distributed content-based publish/subscribe system*. PhD thesis, Iowa State University, 2007.

[SJS00] C. Srisathapornphat, C. Jaikaeo, and C.-C. Shen. Sensor information networking architecture. In *Parallel Processing, 2000. Proceedings. 2000 International Workshops on*, pages 23–30. IEEE, 2000.

[SL13] G. Siegemund and S. Lohs. Glueapi - joining reflex and cometos. Technical Report urn:nbn:de:gbv:830-tubdok-12285, Hamburg University of Technology, Hamburg, Germany, 2013.

[SOM04] Y. Sakurai, F. Ooshita, and T. Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *Principles of Distributed Systems*, pages 283–298. Springer, 2004.

[SR05] A. Shaker and D. S. Reeves. Self-stabilizing structured ring topology p2p systems. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 39–46. IEEE, 2005.

[SRR95] S. K. Shukla, D. J. Rosenkrantz, and S. S. Ravi. Observations on self-stabilizing graph algorithms for anonymous networks (extended abstract). In *Proc. of the 2nd Wksp on Self-Stabilizing Systems*, 1995.

[ST17] G. Siegemund and V. Turau. A self-stabilizing publish/subscribe middleware for iot applications. *ACM Transactions on Cyber-Physical Systems (TCPS) – Special Issue on Internet of Things*, 0:1 – 25, July 2017.

[STLN13] G. Siegemund, V. Turau, S. Lohs, and J. Nolte. Directed link utilization with mahalle+. In *Proceedings of the 12th GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze" (FGSN'13)*, Cottbus, Germany, September 2013.

[STM14] G. Siegemund, V. Turau, and K. Maâmra. Brief announcement: Publish/subscribe on virtual rings. In *Proceedings of the 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'14)*, pages 343–345, Paderborn, Germany, September 2014.

[STM15] G. Siegemund, V. Turau, and K. Maâmra. A self-stabilizing publish/subscribe middleware for wireless sensor networks. In *Proceedings of the International Conference on Networked Systems (NetSys)*, pages 1–8, Cottbus, Germany, March 2015.

[STT06] M. Sharifi, M. Taleghan, and A. Taherkordi. A middleware layer mechanism for qos support in wsn. In *5th International Conference on Networking*, 2006.

[STW+13] G. Siegemund, V. Turau, C. Weyer, S. Lobs, and J. 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'13)*, pages 376–378, Osaka, Japan, November 2013.

[STW15] G. Siegemund, V. Turau, and C. Weyer. A dynamic topology control algorithm for wireless sensor networks. In *Proceedings of the International Conference on Ad-hoc, Mobile and Wireless Networks, ADHOC-NOW 2015*, pages 3–18, Athens, Greece, June 2015.

[THHS07] F. Tashtarian, A. Haghighat, M. Honary, and H. Shokrzadeh. A new energy-efficient clustering algorithm for wireless sensor networks. In *Softw., Telec. and Comp. Netw.*, pages 1–6. IEEE, 2007.

[TNCS02] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2-3):153–167, 2002.

[TP09] D. A. Tran and C. Pham. PUB-2-SUB: A content-based publish/subscribe framework for cooperative P2P networks. In L. Fratta, H. Schulzrinne, Y. T.

0001, and O. Spaniol, editors, *NETWORKING 2009, 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings*, volume 5550 of *Lecture Notes in Computer Science*, pages 770–781. Springer, 2009.

[TP10] D. A. Tran and C. Pham. A content-guided publish/subscribe mechanism for sensor networks without location information. *Computer Communications*, 33(13):1515–1523, 2010.

[TS17] V. Turau and G. Siegemund. Scalable routing for topic-based publish/subscribe systems under fluctuations. In *Proceedings of International Conference on Distributed Computing Systems - ICDCS 2017*, pages 1608–1617, Atlanta, USA, June 2017.

[Tur07] V. Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Information Processing Letters.*, 103(3):88–93, July 2007.

[TW09] V. Turau and C. Weyer. Fault tolerance in wireless sensor networks through self-stabilization. *International Journal of Communication Networks and Distributed Systems*, 2(1):78–98, 2009.

[UT11] S. Unterschütz and V. Turau. Construction of connected dominating sets in large-scale manets exploiting self-stabilization. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[UWT12] S. Unterschütz, A. Weigel, and V. Turau. Cross-Platform Protocol Development Based on OMNeT++. In *Proc. 5th Int. Workshop on OMNeT++*, Desenzano, Italy, March 2012.

[VJC12] R. Vitenberg, H. Jacobsen, and C. Chen. Reinforce Your Overlay with Shadows: Efficient Dynamic Maintenance of Robust Low Fan-out Overlays for Topic-based Publish/Subscribe Under Churn. Technical report, University of Toronto, 2012.

[VZK+11] T. Vandoorn, B. Zwaenepoel, J. D. Kooning, B. Meersman, and L. Vandevelde. Smart microgrids and virtual power plants in a hierarchical control structure. In *Innovative Smart Grid Technologies (ISGT Europe), 2011 2nd IEEE PES International Conference and Exhibition on*, pages 1–7, Dec 2011.

[WTC03] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. First Int. Conf. Embedded Networked Sensor Systems*, pages 14–27, Los Angeles, California, USA, New York, 2003.

[WTLN09] C. Weyer, V. Turau, A. Lagemann, and J. Nolte. Programming wireless sensor networks in a self-stabilizing style. In *3rd Int. Conf. on Sensor Technologies and Applications*, pages 610 – 616, Athens, Greece, 2009.

[WUT08] C. Weyer, S. Unterschütz, and V. Turau. Connectivity-aware neighborhood management protocol in wireless sensor networks. *Drahtlose Sensornetze am 25. und 26. September 2008 in Berlin*, page 23, 2008.

[WZ04] R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad-hoc networks. In *Proc. 18th Int. Parallel & Distr. Processing Symp.*, page 216. IEEE, 2004.

[XFJ03] B. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

[YKM08] K. Yoshida, H. Kakugawa, and T. Masuzawa. Observation on light weight implementation of self-stabilizing node clustering algorithms in sensor networks. In *Proc. International Association of Science and Technology for Development International Conference on Sensor Networks*, pages 1–8, 2008.

[YKP04] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *Network, IEEE*, 18(1):15–21, 2004.

[ZCA08] M. Zhang, M. C. Chan, and A. Ananda. Location-aided topology discovery for wireless sensor networks. In *IEEE Int. Conf. on Communications (ICC '08)*, pages 2717–2722, May 2008.

[ZK04] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 517–526. IEEE, 2004.

[ZLS12] J. Zhang, Q. Li, and E. M. Schooler. ihems: An information-centric approach to secure home energy management. In *IEEE Third Int. Conf. on Smart Grid Communications (SmartGridComm)*, pages 217–222, Nov 2012.

Index

List of Symbols

Graph Model

Graph

G	graph
$\mathcal{S}_G$	ordered sequence of subgraphs on G
$\mathcal{G}$	evolving graph sequence
D	graph diameter
V	vertices of G
E	edges of G
n	number of vertices ($\|V\|$)
m	number of edges ($\|E\|$)
$N(v)$	neighborhood of node v excluding v
$N[v]$	neighborhood of node v including v
$deg(v)$	node degree of v
Δ	maximum node degree in G

Time notion

t	time
$G(t)$	time dependent graph
$G_c(t)$	time dependent subgraph of G
$CC(t)$	connected component

Topology Control Algorithm

Graph

G_c	controlled subgraph of topology
E_c	controlled edge subset of topology
$LT[v]$	local topology of node v including v

Quality

$Q(e)$	quality of an edge e
Q_{min}	minimum quality
Q_{tol}	minimal tolerated quality

LQE - HoPS

Q_S	short term quality value
Q_L	long term quality value
Q_V	stability variation
Q_R	trend estimation
α, β, γ	tuning parameters of HoPS

Lists

C_N	maximum neighbor list size
C_A	maximum assessment list size
C_S	maximum standby list size

Timer

$\mathcal{T}_A$	Assessment list promotion timer
$\mathcal{T}_N$	Neighborhood list promotion timer
$\mathcal{T}_S$	Standby list determination time out

Connected component determination

CC_d	distance to largest node in CC
CC_{id}	largest known node in CC
K	constant to determine error in CC_d

Rank of local topology

$\omega(v)$	weight of a node in $LT[v]$
$dep(u)$	number of dependent neighbors u in $LT[v]$
$Len(LT[v])$	distance metric within $LT[v]$
R_i	set of replacement candidates, category i

Applied Self-stabilizing Algorithms

Algorithms

$\mathcal{A}_{TREE}$	Self-stabilizing tree algorithm
$\mathcal{A}_{MIS}$	Self-stabilizing MIS algorithm
$\mathcal{A}_{MATCH}$	Self-stabilizing matching algorithm

Metrics

$m^{\text{TREE}}_{journey}$	Correctness of tree algorithm
$m^{\text{MIS}}_{correct}$	Correctness of MIS algorithm
$m^{\text{MATCH}}_{correct}$	Correctness of matching algorithm
s^{TREE}	Stability of tree algorithm
s^{MIS}	Stability of MIS algorithm
s^{MATCH}	Stability of matching algorithm

Dispatch frequencies

f_{SSA}	State exchange frequency
f_{OBS}	Communication frequency
f_{TCA}	Topology control frequency

Virtual Ring Algorithm

Definition

$\mathcal{R}$	virtual ring
$Pos(v)$	list of positions in $\mathcal{R}$
l	length of virtual ring

Timer

$\mathcal{T}_{VR}$	maintenance message timer

Spanning Tree Algorithm

$\mathcal{T}_{ST}$	maintenance message broadcast timer

Publish/Subscribe Algorithm

Routing Schema

$\mathcal{I}$	routing interval
$RS(v)v$	routing structure of node v
c	channel identifier
ep	end position

Timer

$\mathcal{T}_{Sub}$	subscription renewal timer
$\mathcal{T}_{Pub}$	publication dispatch timer

Evaluation Algorithms

$\mathcal{A}_S$	Shen's algorithm [She07]
$\mathcal{A}_F$	typical flooding algorithm
$\mathcal{A}_O$	single bfs-tree algorithm
$\mathcal{A}_M$	multi bfs-tree algorithm

List of Acronyms

ACK	Acknowledgment
API	Application Programming Interface
ASL	ASymmetry Level
ATPC	Adaptive Transmission Power Control
bfs	breadth-first search
CCA	Clear Channel Assessment
CPS	Cyber-Physical Systems
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CST	Cached Sensornet Transformation
dfs	depth-first search
DFT	depth-first traversal
ETX	Expected Transmission Count
EWMA	Exponentially Weighted Moving Average
HoPS	Holistic Packet Statistics
IoT	Internet of Things
ISPRP	Iterative Successor Pointer Rewiring Protocol
LEEP	Link Estimation Exchange Protocol
LGRP	Logical Grid Routing Protocol
LQE	Link Quality Estimator
LQI	Link Quality Indication
MAC	media access control
MIS	Maximal Independent Set
MWCDS	Minimum Weighted Connected Dominating Set
NORMAN	NeighbORhood Management for Ad-hoc Networks
PMU	Phasor Measurement Unit
PRR	Packet Reception Rate
PSR	Packet Success Rate
RAM	Random Access Memory
RETC	Reliable Energy-efficient Topology Control
RNP	Required Number of Packets
ROM	Read-Only Memory
RSSI	Received Signal Strength Indication
RTS/CTS	ready-to-send/clear-to-send
QoS	Quality of Service

SNR	Signal-to-noise ratio
SSA	self-stabilizing algorithm
TCA	Topology Control Algorithm
TDMA	Time Division Multiple Access
VRR	Virtual Ring Routing
WANET	Wireless Ad hoc NETwork
WMEWMA	Window Mean Exponentially Weighted Moving Average
WSN	Wireless Sensor Network

Curriculum Vitae

Personal Data

Surname	Siegemund
First Name	Gerry
Date of Birth	September 12, 1984
Place of Birth	Eisenach, Germany
Nationality	German

Education and Work

06.2017 – today	IoT Consultant and Team Lead Digital Transformation *CGI Deutschland Ltd. & Co. KG - Erfurt, Germany*
08.2016 – 06.2017	Unemployed - Finishing PhD Thesis on own accord
02.2012 – 07.2016	Research and Teaching Fellow (PhD Candidate) *Institute of Telematics, Hamburg University of Technology (TUHH), Germany*
9.2014 – 12.2014	Conjoined Research abroad *École Polytechnique, Paris, France*
07.2011 – 01.2012	Research and Teaching Fellow (PhD Candidate) *Institute for Software Systems, Hamburg University of Technology (TUHH), Germany*
10.2004 – 06.2011	Studies in Computer Science and Engineering (Diplomstudiengang Informatik-Ingenieurwesen) *Hamburg University of Technology (TUHH), Germany* *Final degree: Diploma (Diplom-Ingenieur)*
04.2010 – 09.2010	Internship IBM *Böblingen Germany, Smart Analytics Optimizer for DB2*
08.2001 – 06.2002	High School abroad *North Monterey County High School, Castroville, CA, USA* *Final degree: High School Diploma*
08.1995 – 06.2004	Secondary School *Elisabeth-Gymnasium, Eisenach, Germany* *Final degree: Abitur*