UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR WIRTSCHAFTSWISSENSCHAFTEN
INSTITUT FÜR INFORMATIK UND WIRTSCHAFTSINFORMATIK
LEHRSTUHL FÜR PERVASIVE COMPUTING

Bachelorarbeit

Design and Implementation of a Distributed, Low-Power, Signal Strength Measurement System

Jan Nauber
Matrikelnummer: 3003433

Networked Embedded Systems Group
Institut für Informatik und Wirtschaftsinformatik
Fakultät für Wirtschaftswissenschaften
Universität Duisburg-Essen

January 24, 2017

Erstprüfer: Prof. Dr. Pedro José Marrón
Zweitprüfer: Prof. Dr. Torben Weis
Zeitraum: 18. November 2016 - 10. Februar 2017

Contents

Abstract

The function of the abstract is to summarize, in one or two paragraphs, the major aspects of the entire bachelor or master thesis. It is usually written after writing most of the chapters.

It should include the following:

- Definition of the problem (the question(s) that you want to answer) and its purpose (Introduction).
- Methods used and experiments designed to solve it. Try to describe it basically, without covering too many details.
- Quantitative results or conclusions. Talk about the final results in a general way and how they can solve the problem (how they answer the question(s)).

Even if the Title can be a reference of the work's meaning, the Abstract should help the reader to understand in a quick view, the full meaning of the work. The abstract length should be around 300 words.

Abstracts are protected under copyright law just as any other form of written speech is protected. However, publishers of scientific articles invariably make abstracts publicly available, even when the article itself is protected by a toll barrier. For example, articles in the biomedical literature are available publicly from MEDLINE which is accessible through PubMed. It is a common misconception that the abstracts in MEDLINE provide sufficient information for medical practitioners, students, scholars and patients[citation needed]. The abstract can convey the main results and conclusions of a scientific article but the full text article must be consulted for details of the methodology, the full experimental results, and a critical discussion of the interpretations and conclusions. Consulting the abstract alone is inadequate for scholarship and may lead to inappropriate medical decisions[2].

An abstract[**?**, **?**, **?**, **?**] allows one to sift through copious amounts of papers for ones in which the researcher can have more confidence that they will be relevant to his research. Once papers are chosen based on the abstract, they must be read carefully to be evaluated for relevance. It is commonly surmised that one must not base reference citations on the abstract alone, but the entire merits of a paper.

Chapter 1

Introduction

[You should answer the question: What is the problem?]

This paragraph should establish the context of the reported work. To do that, authors discuss over related literature (with citations[1]) and summarize the knowledge of the author in the investigated problem.

ToDo: how to make citations

An introduction should answer (most of) the following questions:

- What is the problem that I want to solve?
- Why is it a relevant question?
- What is known before the study?
- How can the study improve the current solutions?

To write it, use if possible active voice:

- We are going to watch a film tonight (Active voice).
- A film is going to be watched by us tonight (Passive voice).

The use of the first person is accepted.

1.1 Motivation

A good introduction usually starts presenting a general view of the topic and continues focusing on the problem studied. Begin it clarifying the subject area of interest and establishing the context (remember to support it with related bibliography).

[1] To cite a work in latex

1.2 Problem definition

Additionally, focuses the text on the relevant points of your investigation and problems that you want to solve, relating them with the first part.

1.3 Thesis/Diplom/Bachelor/Master Structure

Present your work to the reader giving a brief overview of what is going to cover every chapter. Write only general concepts, no more than one or two sentences per chapter should be necessary.

Chapter 2

Materials and Methods

This section is to clarify the pre-existing tools, defining what was developed in this field until now, and why this tool was used instead of others.

The general structure is the following:

- Definition of the specific tool(s) studied (robots, sensor nodes, smart-phones). When relevant, pre-existing experiments.
- Definition of the context of use (indoor/outdoor, humans/animals/robots, with-/without connection).
- Definition of used protocols (How the data are collected, when, etc.)

2.1 Wireless Sensor Networks

A wireless sensor networks are a collection of low-cost, low-power and multifunctional sensor nodes that are placed inside a to be monitored area

2.2 Radio Tomographic Imaging

2.3 Timeslots

2.4 Testbed

The testbed is a wireless sensor network set up in the third floor of the SA building at the University Duisburg Essen. It covers on half of the whole floor containing a main corridor, two laboratories, two side corridors leading to three offices each, an elevator, seven smaller storage rooms and one server room. All in all the area is

$531\mathrm{m}^2.To monitor the floor there are 32 nodes placed at the positions shown in figure (random number). This me$
$hop network.$

2.4.1 Motes

2.4.2 TinyOs

Chapter 3

Approach

This chapter explains the general structure and functionality of a system that is supposed to measure the received signal strength of every link in a network. Write about what this chapter explains

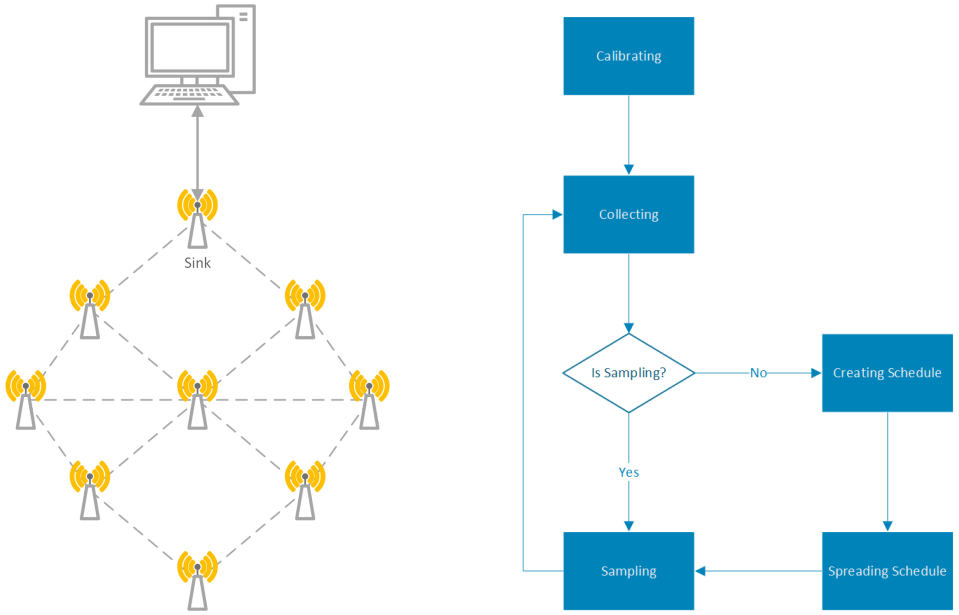
3.1 General Structure

To measure the RSSI of each link all the nodes need to send messages. Since two nodes sending a message at the same time will distort the RSSI measurements we need to make sure that only one node sends a message at a given point in time. One approach to achieve this is the timeslot based method explained in Chapter 2.3. This method however includes an error inside its timeslots, resulting in a small delay between messages. To eliminate the delay timeslots bring along a different approach is suggested in this thesis which is based on predefined predecessors for each node. Based on this a node will be able to send a message directly after receiving a message from its predecessor and at the same time making sure that only one node sends at a given point in time.

ToDo: Complex stuff because of not every node hearing the others

The system consists of a base station with high processing power and a large memory as well as multiple distributed low power nodes. One of the nodes is the sink which is directly connected to the base station. The nodes are able to communicate with each other via radio, however not every node can hear all the other nodes. This structure is represented by Figure 3.1a. The fact that not every node is able to hear all the other nodes makes it a challenge to collect data from the network and to create a fitting schedule that defines predecessors for all the nodes inside the network. To be able to do so a first calibration phase is needed to create paths from every node to the sink and collect information about the connections between nodes. Then the information about the connections need to be collected at the sink and sent to the base station in a collection phase. When the base station received all the information it can create the schedule. After that the base station has to send the schedule to the sink that starts spreading it inside the network.

Is the schedule spread and all the nodes know when to send their messages, the sampling of the RSSI can start. Therefore messages are sent according to the predecessors defined in the schedule. All the nodes receiving the messages are now able to measure the received signal strength to the sending node and store this information. When the schedule is completed another collection of the data is needed to gather the measured RSSI at the base station for further processing. When the collection is done, the system can start sampling again and then again collect the data until the system is stopped. This process is shown in Figure 3.1b.



(a) The physical architecture of the system

(b) The general processes and the order in which they are executed

Figure 3.1

3.2 Base Station

Since creating the schedule and storing and processing the data need a lot of processing power and storage, a base station is needed that fulfils these requirements. This base station needs to be connected to the sink so it can receive the collected data to process it. It also needs to be able to send data to the sink, so the schedule can be spread inside the network.

ToDo: kill this section?

3.3 Calibration

The calibration has two tasks. It needs to figure out for each node individually which nodes a node has in its range and it needs to create paths to the sink. These paths will then later be used to send data to the sink or spread information inside the network. To achieve these tasks each node will send multiple messages without any specific pattern.

ToDo: define no pattern

Each node needs to be able to keep track of the nodes in range. Therefore each node needs to have its own neighbour table. When a node receives a message it can put the node it received the message from inside that neighbour table. All the neighbour tables will later be the basis to create the schedule.

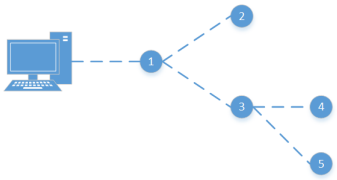
To later collect data from each individual node at the sink, each node needs to know to which node it needs to send its data to so it will reach the sink. This node is the parent of the node. Moreover to spread data inside the network each node also needs to know which nodes will send to itself in direction of the sink. These nodes are the children of a node. When each node found its parent and its children all the paths together will form a tree structure with the sink as the root.

To find the parents for the nodes each node needs to save one extra information and also include this information inside the messages it sends. This information is the quality of the current path a node has to the sink. At the beginning all the nodes except the sink will initialize their path quality with the worst possible value. The sink will initialize its path quality with the best possible value. Now when a node receives a message it will add to the received path quality of the sending node the quality of the link between itself and the sending node to calculate the path quality to the sink for the node, if it would choose the sending node as its parent. The quality of a link is represented by the received signal strength. When the new path quality is calculated the node compares the calculated value with its current path quality. If the calculated value is better the node will set its parent to the sending node and save the new path quality.

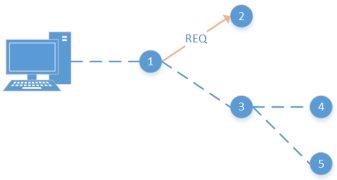
Now, to not only know the direction to the sink but also the direction going away from the sink the nodes need to find their children. Therefore each node will simply include their own parent inside the message it will send. A receiving node will now check if it is the parent of the sending node. If that is the case the receiving node can save the sending node as a child.

3.4 Collection

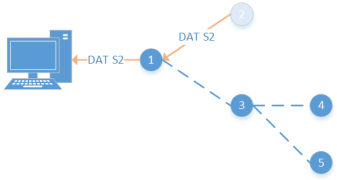
To collect the data from the network, we make use of the created paths and their tree structure. The process is shown in Figure ??. The Figure shows a wireless sensor



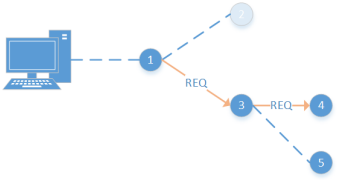
(a) At the beginning all the nodes are unmarked and still have data to send



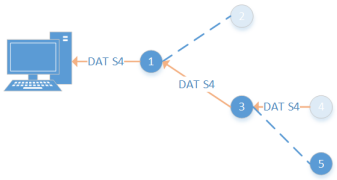
(b) Node 1 sends the first request to node 2



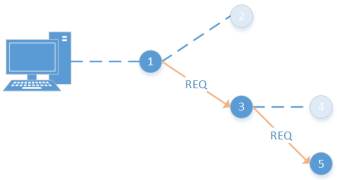
(c) Node 2 does not have any children so it sends its data and gets marked



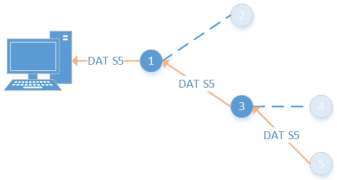
(d) Now node 2 is marked so node 1 sends a request to node 3, which forwards it to node 4



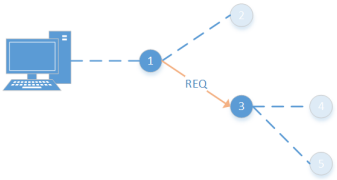
(e) Node 4 does not have any children so it sends its data and gets marked



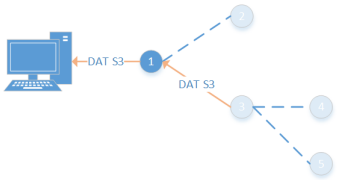
(f) Node 3 is still not marked so node 1 sends a request to it. Node 4 is marked so node 3 forwards the request to node 5



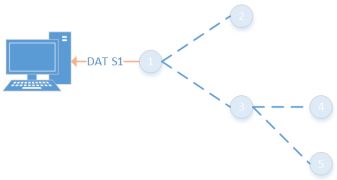
(g) Node 4 does not have any children so it sends its data and gets marked



(h) Node 3 is still not marked so node 1 sends a request to it



(i) All the children of node 3 are marked so it starts sending its data and gets marked itself



(j) Now all the children of node 1 are marked and it can send its own data to the base station and gets marked

Figure 3.2: This is an example for the collection. REQ means request and DAT Sx stands for data from source x

network represented by the nodes and the paths to the sink created in the calibration phase. Note that the nodes could also have other connections between each other.

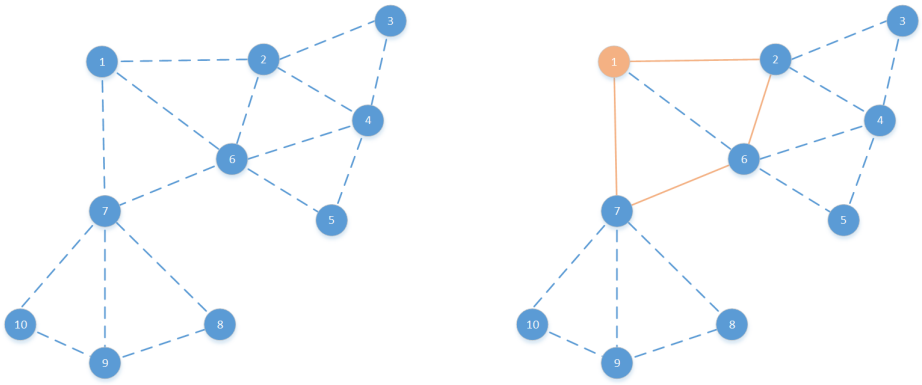
To start the collection the sink sends a request to one of its children. The child receiving that request will check if it has any children himself and if that is the case, it will forward the request to one of them. When the request reaches a node without any children, the node will send its data to its parent which forwards the data to its parent, until the data reaches the sink, which will forward it to the base station. Every node that receives a data message marks the source of that data as done. When the sink sent the received data to the base station, it sends a new request to one of its children that has not been marked as done. That child again forwards that request to a child that has not been marked as done. If the request reaches a node that has no children or every child has been marked as done it sends its own data. This process will be repeated until the sink does not have any more children left that are not marked as done. Then the sink can send its own data to the base station and finish the collection. All nodes now need to unmark all the other nodes, so another collection is possible.

3.5 Creating the Schedule

Creating a schedule is a quite challenging task since we need an algorithm that visits every node in a graph at least once and starts and ends at the same node. The optimal schedule would be a Hamilton-Circle that is a circle inside a graph that visits each node exactly once. To figure out if a Hamilton-Circle exists, there is however only the way to bruteforce all the possible combinations. This is highly inefficient and possibly we do not even get a result at the end. Therefore this thesis suggests a simple method that makes sure every node gets visited at least once and the path starts and ends at the same node. This method however is not able to create an optimal path and could be improved.

3.5.1 Rooted Circles

The suggested method is based on smaller circles inside the graph. These circles will have a root node that is the start and the end point of the circle. All the nodes inside this circle will be in range of the root of the circle. These circles will be called rooted circles. In Figure 3.3b such a rooted circle is represented inside the network depicted in Figure 3.3a. To create a rooted circle, one node needs to be chosen as the root. Then the root will take one node from its neighbour table and choose it as the second node in the circle. Then the second node will look into its neighbour table and choose a node that has the root node inside its neighbour table. The chosen node will do the same and the process will be repeated until a node does not have a neighbour that has the root as



(a) The example network with all its connections between nodes

(b) Example for a rooted circle with node 1 as the root

Figure 3.3: A rooted circle inside a example network

a neighbour. Then the circle will be closed and the path goes to the root. When we look at the example in Figure 3.3b this means node 1 was chosen as the root. Then node 2 was picked as the second node in the circle. Node 2 has multiple neighbours but only one of them, node 6, has the root node 1 as a neighbour. This means node 6 is chosen as the next node in the circle. Node 6 now has node 2 and node 7 as neighbours that also have the root as a neighbour, however node 2 is already inside the circle so node 7 is chosen as the next node. Node 7 now has no more neighbours that have the root as a neighbour and the circle is closed.

3.5.2 Creating a Full Schedule

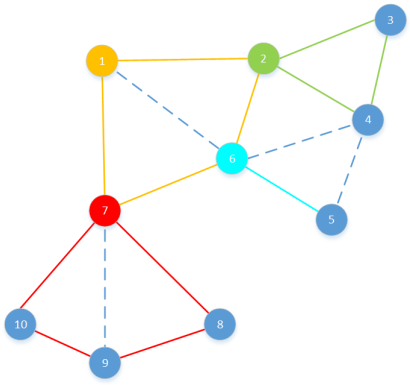


Figure 3.4: Full schedule for the example network of Figure 3.3a. The order in which the nodes send their messages would be: 1 2 3 4 2 6 5 6 7 8 9 10 7 1.

We now want to fill the whole graph with rooted circles. Therefore we will choose the sink as the first root and create a rooted circle around it. When the circle is done, we will go through all the nodes of the circle and, if possible, create rooted circles around them as well. Then we do the same for all the new circles until every node was suggested as a root once. When creating new circles it is not possible to choose a node already inside another circle to be part of the new circle. In Figure 3.4 the network from Figure 3.3a is fully covered by rooted circles. The first circle created was the one that has node 1 as a root. Then all the nodes inside that circle where chosen as new roots to create new rooted circles. The first thing one could notice is that the circle with the root 6 only has one other node and does not really form a circle. This happens if the circle is closed directly after the second node following the root is chosen because there are no more neighbours left that also have the root as a neighbour. When running the schedule, this means a message would be sent from node 6 to node 5 and then from node 5 back to node 6. Also note that in theory there is a bigger rooted circle possible with the root 6 when node 4 would be included, but node 2 created its rooted circle first and included node 4 and therefore blocked it for rooted circles created at a later point in time. In Listing 3.1 the pseudocode that represents an algorithm to cover a whole graph with rooted circles is shown.

Listing 3.1: Pseudocode that covers a graph with rooted circles

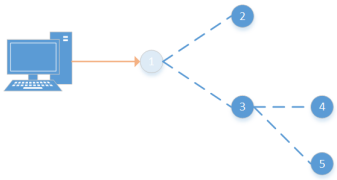
```
RootedCircle createRootedCircle(Node root) {
  Node node = getNextForCircle(root, root)

  If(node != null) {
    RootedCircle rootedCircle = new rootedCircle(root)
    rootedCircle.addNode(node)
    node.setPartOfCircle(true)

    while((node = getNextForCircle(root, node, rootedCircle)) != null) {
      rootedCircle.add(node)
      node.setPartOfCircle(true)
    }

    return rootedCircle
  } else {
    return null
  }
}

Node getNextForCircle(Node root, Node neighbour) {
  for each (Node node in root.getNeighbourList)
    if(node.isNeighbourOf(neighbour) and not node.isPartOfACircle())
```

```
      return node
  return null
}

List<RootedCircle> coverGraphWithRootedCircles(Node firstRoot) {
  List<RootedCircle> circleList = new List<RootedCircle()>

  circleList.add(createRootedCircle(firstRoot))

  for each (RootedCircle circle in circleList) {
    for each (Node node in circle) {
      RootedCircle newCircle = createRootedCircle(node)
      if(newCircle != null)
        circleList.add(newCircle);
    }
  }
}
```

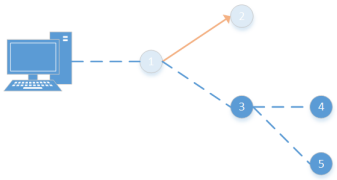
3.6 Spreading the Schedule

To spread the schedule we will again make use of the tree structure of the created path. When the sink received the schedule from the base station it will forward it to its first child. The child will forward it to one of his children and so on. When a node that received the schedule has no more children, it will send the schedule back to its parent. The parent receiving the schedule will now forward the schedule to its next child. If a parent received the schedule back from all its children, it will forward it to its own parent. In Figure 3.5 an example for this process is given. The Figure shows a wireless sensor network represented by the nodes and the paths to the sink created in the calibration phase. Note that the nodes could also have other connections between each other.

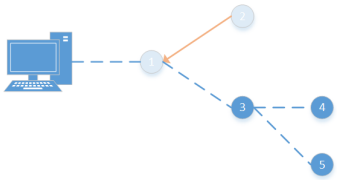
When you look at the example, one can see that already in Figure 3.5g the schedule is received by all the nodes but still there are messages sent that could seem useless at this point. However at this point in time we do not know if the sink has any more children that did not receive the schedule yet. Therefore the schedule needs to travel all the way back to the sink. Only at the moment the schedule reaches the sink and the sink does not have any more children left that need to receive the schedule, we know for sure that the schedule is fully spread.



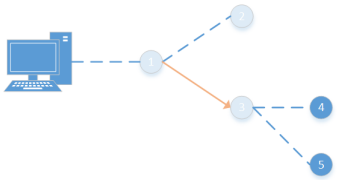
(a) The base station sends the schedule to node 1.



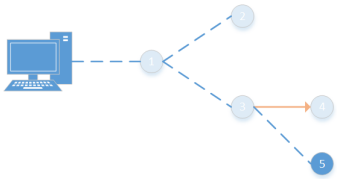
(b) Node 1 forwards the schedule to node 2



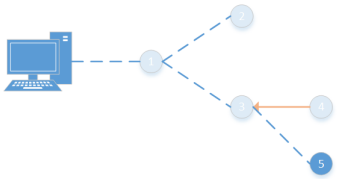
(c) Node 2 does not have any children and sends the schedule to its parent node 1



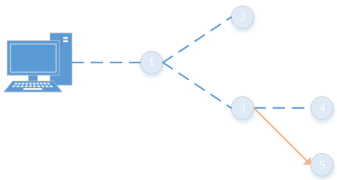
(d) Node 2 is now marked so node 1 will forward the schedule to node 3



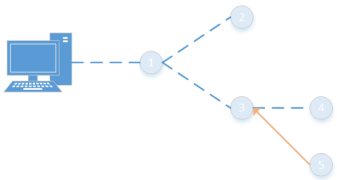
(e) Node 3 forwards the schedule to node 4



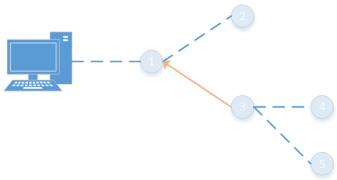
(f) Node 4 does not have any children and sends the schedule to its parent node 3



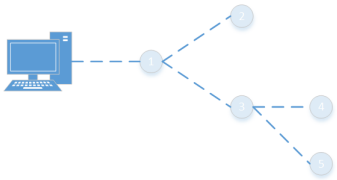
(g) Node 3 has still node 5 as an unmarked child and forwards the schedule to it



(h) Node 5 does not have any children and sends the schedule to its parent node 3



(i) Node 3 has no more unmarked children and sends the schedule to its parent node 1



(j) Since node 1 does not have any more unmarked children the schedule is now fully spread inside the network

Figure 3.5: This is an example for a path a schedule message takes to be spread inside the network

3.7 Sampling

When all nodes received the schedule it is possible to sample the received signal strength by sending messages according to it. Therefore the sink will start sending a message. When its successor receives the message it can send its message and so on until the last message was send and the sampled data can be collected. However we need to take into account that one node could appear multiple times inside the schedule, meaning a node could have multiple predecessors and successors. Therefore we need to include the successor of the sending node inside the message so a receiving node can see if he is the correct successor at that moment.

3.7.1 Message Drops

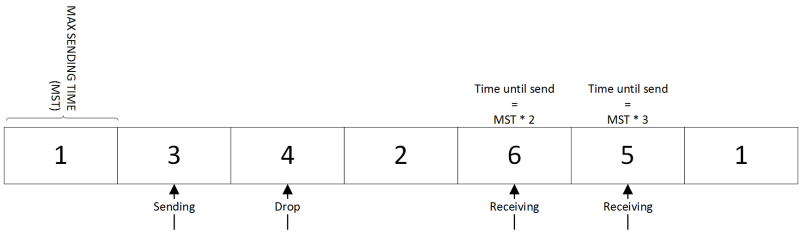


Figure 3.6: Representation of a schedule. Node 3 sends a message. Node 4 drops the message but node 6 and 5 receive it so that they can calculate the time when they should send

A problem of the proposed method are message drops. If a successor does not receive the message of its predecessor the whole system would stop. Here a similar technique a timeslots based system uses comes in handy. For this method every node needs to know the whole schedule and not only its own predecessors and successors. Then whenever a node receives a message it can look up how many nodes need to send between the node that just send and itself. Than the amount of sending nodes is multiplied by the maximal time a node needs to send a message. The result is the time after the node can send its own message, without receiving a message from its predecessor. Again we need to take into account that a node can appear multiple times in the schedule. Therefore after a node send a message it needs to calculate the maximal time until it will send the next message.

Chapter 4

Implementation

In this chapter a way to implement a system like the one described in Chapter 3 will be explained by an example based on TinyOS.

4.1 General Structure

The system is dividable into two main components. The base station, that stores and processes the data, and the motes, that collect the data. Both parts of the system need their own hard- and software to complete their tasks.

4.1.1 General Base Station

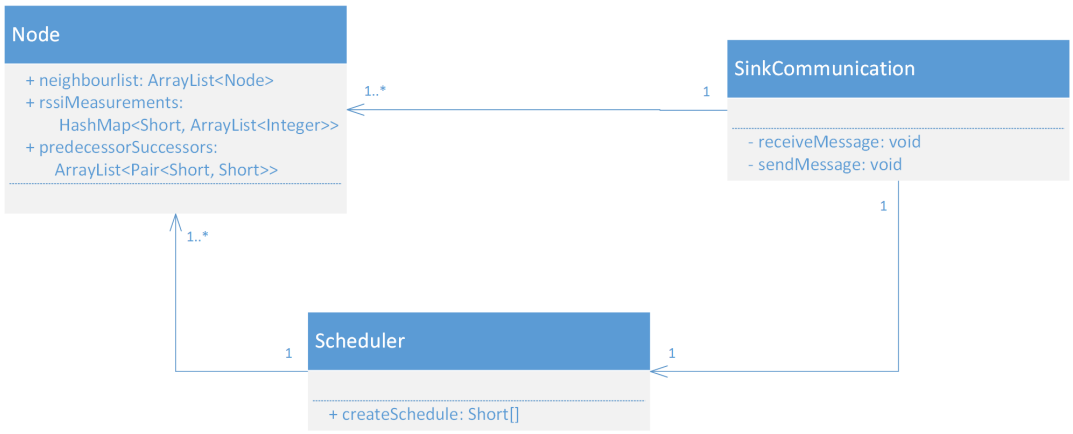


Figure 4.1: General structure and functionality of the base station

The base station can be a computer running a Java application connected to the sink via a USB-Cable. In Figure 4.1 you can see the general structure of the Java application with its classes and their basic functionality. First, there is the SinkCommunication

class that handles the communication with the sink. It is responsible for receiving and processing messages and to send messages to the sink. Then there is the Node class which stores all the information about one node. It stores the neighbours of the node, the measurements a node made and its predecessors and successors. The last class is the Scheduler that is responsible of creating a schedule based on the data saved in the node classes.

4.1.2 General Mote Application

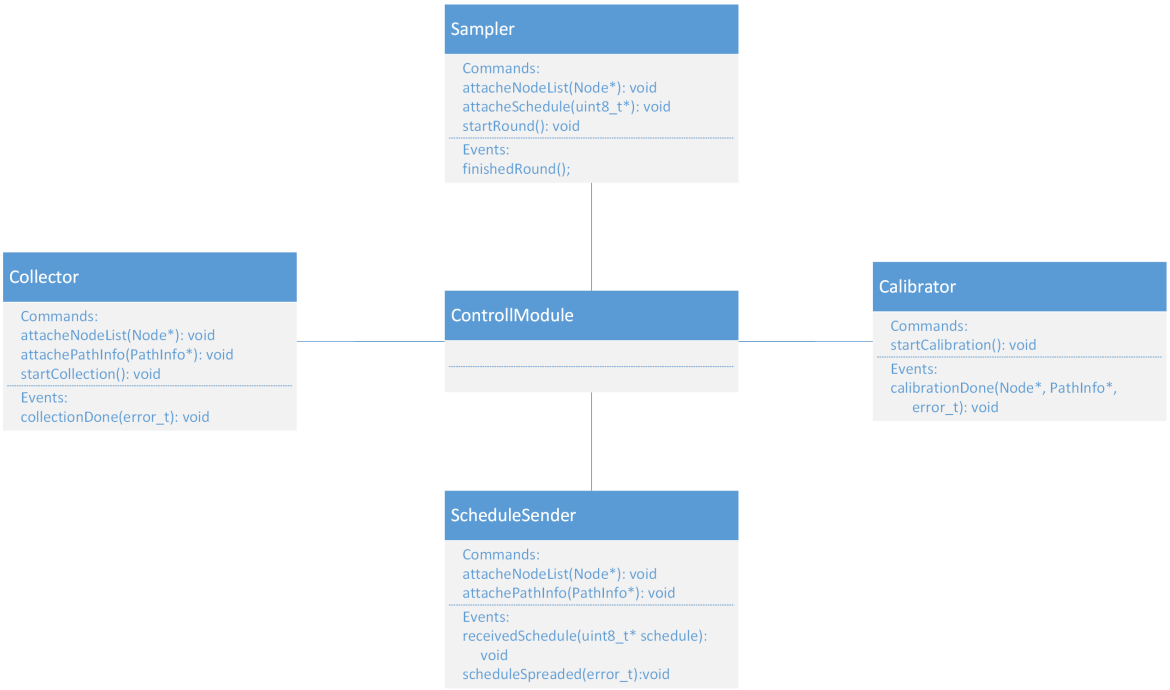


Figure 4.2: The interfaces each module provides. The ControllModule does only use all the other interfaces and does not provide one itself

ToDo: Grafic: attach statt attache und spread statt spreaded, neighbourlist

The application running on the motes has multiple tasks. It needs to be able to calibrate the network, collect data from the network, spread the schedule inside the network and sample the RSSI. Therefore it can be split into multiple modules with fitting interfaces that each handle exactly one task. The suggested structure of the application is shown in Figure 4.2. The ControllModule connects all the other modules and makes sure the informations the other modules provide get delivered correctly to the modules needing the informations. The Calibrator is responsible for the calibration of the network. It needs a command to start the calibration and an event that triggers when the calibration is done and provides the gathered informations. Collecting the data from the network

and sending them to the pc is covered by the Collector module. This module needs to know the neighbours and the information about the nodes parent and children. Therefore the interface of the module provides two commands to attach this information to the module. Moreover it needs a event that triggers when the collection is done. To spread the schedule inside the network there is the ScheduleSender module. It also needs the information about the neighbours and the parent and children of the node and has the suitable commands for this. It does not necessary need a command to start the spreading since it can directly react when the node receives a schedule message from the base station. Lastly the module has two events. One that triggers when a node received the schedule and one that triggers when the schedule is spread. The event triggering when the schedule was received provides the received schedule. The schedule spread event will only be triggered at the sink since it is the only node able to detect if the schedule is spread. The last module is the Sampler. It needs the neighbour list and the schedule and has the fitting commands. Moreover it has a command to start one round of the schedule and an event that triggers when this round is finished.

4.2 Base Station

4.2.1 Receiving and Sending messages

To receive and send messages from the sink it is possible to use the

4.2.2 Storing the information

4.2.3 Creating the Schedule

4.3 Calibration

4.4 Collection

4.5 Creating the Schedule

4.6 Spreading the Schedule

4.7 Sampling

Chapter 5

Evaluation

In this chapter you should describe the previous (if possible) and final experiments performed on the implementation.

Every single experiment should be explained individually, providing to the reader information about the meaning of the experiment, the expected (theoretical) results, the final results, the comparison between them and others (if possible) and the conclusions.

Each experiment should include a description, covering (when possible) the following information:

- Significant physical features (obstacles present on the environment, human presence, temperature, humidity, possible noise sources, computational speed of the machine, etc.)
- The precise location of the experiment (latitude and longitude, room number or citation to a description of the used laboratory).
- Sampling design (variable(s) measured, transformation performed to the data, samples collected, replication, comparative with a Ground Truth system, collecting data protocol).
- Analysis design (how the data are processed, statistical procedures used, statistical level to determine significance).

The provided information should be sufficient to allow other scientists to repeat your experiment in the same conditions. Thus, the use of standard and well-known equipment could only be represented by a simple sentence, but the non-standard equipment should be described in detail, citing the source (vendor) and most important characteristics.

To write it, try to use the third person when describing the experiments and results. Avoid to use first person. Past tense should be the dominant conjugation (the work is done and was performed in the past).

Note: Graphics represent really well data, use them! (Matlab or Octave could be useful for that).

Chapter 6

Discussion

The meaning of this paragraph is to interpret the results of the performed work. It will always connect the introduction, the postulated hypothesis and the results of the thesis/bachelor/master.

It should answer the following questions:

- Could your results answer your initial questions?
- Did your results agree with your initial hypothesis?
- Did you close your problem, or there are still things to be solved? If yes, what will you do to solve them?

Chapter 7

Acknowledgements

(This part is optional, and it could be completely excluded by deleting
\include {content/chapters/chapter7}
from the Firstname_Lastname_Diplom_Master_arbeit.tex file)

This paragraph could mention people or institutions that supported you to some extent with your work or friends and relatives that supported you during your study period.

Erklärung

German

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

English

I hereby declare that I have written this Bachelor thesis independently, using no other than the specified sources and resources, and that all quotations have been indicated.

Essen, January 24, 2017
(Place, Date)

Jan Nauber