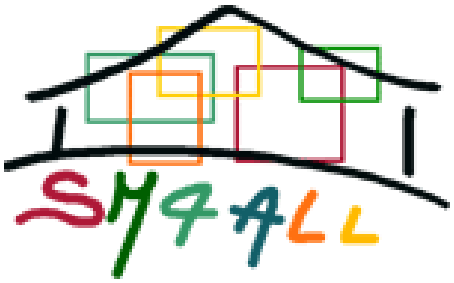
University of Groningen
Institute of Mathematics and Computing Science

Master of Science Thesis

JLocT: Java Location Tracker

Doing indoor location-tracking with a Wireless Sensor Network



Version 1.0

Author: Martijn de Groote (s1650130)

Supervisor and first reviewer: prof. dr. ir. M. Aiello

Second reviewer: prof. dr. G.R. Renardel de Lavalette

Groningen, 25 March 2011

Acknowledgements

First of all I would sincerely thank prof. dr. ir. Marco Aiello. Without his supervision, feedback and help this project and thesis would not have come into being. I would also like to thank prof. dr. G.R. Renardel de Lavalette for his feedback on my thesis. His feedback allowed me to further increase the quality of my thesis. A special thanks goes out to prof. dr. Ernst Wit for providing important information on the statistics-part of the project. Last but not least I would like to thank my parents for the love and support they gave me during the project and writing the thesis.

Contents

Abstract

Ubiquitous systems are those for which the computation becomes invisible and pervades the environment [1]. Ubiquitous systems are also known as pervasive systems. One important problem that needs to be addressed in such systems is localization. A pervasive system needs to be aware of the current locations of its users. Localization is the subject of this thesis that relates to a pervasive system such as that of the project “Smart Homes for All” (SM4ALL). Note that information gained by location-tracking can have a substantial added value to the decision-making process of a smart home.

One popular way of doing location-tracking is by use of a Wireless Sensor Network (WSN). A WSN is composed of small nodes called motes that are at least equipped with processing capabilities, sensors and wireless communication. Normally they can be used to gather and process sensor-information. Moreover information obtained from characteristics of radio-signals can be used in determining the location of users.

Popular characteristics of radio-signals used in location-tracking are Angle of Arrival (AOA), Time of Arrival (TOA), Time Difference of Arrival (TDOA) and Received Signal Strength (RSS). Except for RSS these techniques either require specialized hardware and/or time-synchronization to work. In an attempt to keep the architecture as simple as possible, RSS is chosen as the characteristic of a radio-signal used for location-tracking.

A location-tracking system called “Java Location Tracker” (JLocT) is proposed. JLocT is implemented in Java and uses Sentilla JCreates as motes to form a WSN. JLocT uses RSS-based information in determining the location of a user in an indoor-environment. Since JLocT does not require a site-survey or floor-plan, fast deployment in an indoor-environment is possible.

During development several challenges are encountered that have to be overcome. One of these problems is the lack of multihop-communication facilities on the motes used. Because of this an implementation of a multihop-protocol is created.
Other challenges with RSS based location-tracking systems, and thus with JLocT, are “multipath interference” and “transient blocking of a radio-signal”. Multipath-interference occurs because radio-signals can travel indirectly or directly from a sender to a receiver due to reflecting materials in an indoor environment. This results in higher or lower readings of signal strength than is the case if the radio-signal is traveling in a vacuum. Transient blocking of a radio-signal basically happens when an obstacle is temporarily blocking the path of a radio-signal between sender and receiver. JLocT employs known techniques that can at least mitigate the effects of these environmental conditions.

The thesis first introduces the project, its concepts and sets its requirements. Next related work is investigated giving an idea of the current state of art. This is followed by sections explaining

how location-tracking and auxiliary processes are done in JLocT. The thesis then discusses the experiments that are done using JLocT, the corresponding results, conclusions and future work.

1 Introduction

1.1 Context

Computers can take on different forms in indoor pervasive systems. These computers can also perform different tasks in a building. An EU project called “Smart Homes for All”-project (SM4ALL) tries to handle these different forms that computers might have in an indoor pervasive system.

The goal of SM4ALL is to build middleware for a dynamic embedded environment, transforming a normal home into a “smart home”. Such a smart home can take advantage in decision-taking if it constantly is aware of in what rooms humans are located. Thus a smart home actually needs a localization component in order to do its work.

For example if the smart home knows that a user is in the living room it can regulate heating, light, airco or turn on the television. Alternatively if the smart home knows that nobody is in the house it can turn off any appliances to save energy. That is why this project can be seen as a sub-project for of the SM4ALL-project.

WSN’s allow data to be collected at one place and send it using multihop-communication to another location that may be separated much further away. In fact by just adding new infrastructure nodes to the system a WSN suddenly has the ability to cover a much greater area. This scalability-feature is especially an advantage in larger (indoor-)environments.

Sometimes areas in an indoor environment can be hard to reach using radio-signals from for example WiFi-hotspots. The flexibility of just adding new infrastructure nodes to the system also allows such areas to be covered. This is done by just adding a new infrastructure node close to the area that is hard to reach.

The software developed in this project does still require extensions to be fully compatible with the SM4ALL-project. However this Master-project only focuses on trying to solve the problem of location-tracking for a WSN.

1.2 Problem statement

The main goal of the project is the development of a location-tracking system using a WSN. Here the WSN is comprised of a collection of motes. To be exact the motes used in this project are Sentilla JCreates [2]. These motes run a Java Virtual Machine (JVM). Thus programming by default is done in Java. This is also the programming language chosen for this project. That is why the developed location-tracking system is called “Java Location Tracker” (JLocT). For more information about the Sentilla JCreates see Section 9.1.

The basic assumption for the project is a setting of a WSN with fixed infrastructure motes and one moving (roaming) mote. Another assumption is that the location-tracking system has a precision on room-granularity. The project comprises multiple problems that need to be solved.

1) Can a location-tracking system be created with technology that is by default present in all WSN's?
2) Can a location-tracking system using a WSN be build that does not require doing a floor-plan or site-survey during deployment?
3) Should such a location-tracking system let roaming motes do classification or is it better to have a dedicated computer for this task?
4) How should the design of such a location-tracking system look like when using Sentilla JCreates as motes in a WSN?
5) Is multihop-communication by default supported on the used motes? If this is not the case how should the design and implementation of a multihop-protocol look like?
6) Are there factors that might impact the probability of successful creation of a multihop-network / routing-topology?
7) Are there environmental conditions that might impact classification-performance?
8) If there are such environmental conditions are there suitable techniques available that can mitigate the effects on classification-performance?
9) What worst-case classification-performance is possible using only a limited amount of fixed infrastructure nodes?
10) What factors other than environmental conditions might impact classification-performance?

The original idea was to create a hybrid network that can do location-tracking. This might be done using both Sentilla JCreates and SunSPOTs [3]. Unfortunately attempts to create such a hybrid network failed. For more information about the attempts and suspected reasons of failure see appendix A.

1.3 Structure of this report

Section 1.1 explains the position of this project within the SM4ALL-project. Therefore Chapter 2 first gives some background information about the SM4ALL-project and its goals. Once we have gained some knowledge about the SM4ALL-project, Chapter 3 presents the case study of this project. There some requirements of the project and JLocT are presented. Chapter 4 discusses the results of the literature research performed. Basically related work done in the area of location-tracking is presented there.

The performed literature research has given rise to a location-tracking technique used during this project. This location-tracking technique and its important concepts are discussed in depth in Chapter 5. Once a tracking-technique is chosen a system can be build around it. Chapter 6 therefore discusses the software-architecture of JLocT that is developed during this project.

Unfortunately no multihop-protocol is by default in place for the Sentilla JCreates. This leads to the design and implementation of such a protocol as also being part of this project. That is why Chapter 7 gives a more theoretical view of the design and implementation of such a multihop-protocol. Now that support for multihop-communication is in place JLocT can be developed. The implementation-details of JLocT are therefore the subject of Chapter 8. Once JLocT is developed, experiments and tests can be done using it. The results of these performed experiments and tests are discussed in Chapter 9.

Finally Chapter 10 draws conclusions from the results of the performed tests and experiments. However Chapter 10 also talks about the possible future of JLocT. This is done in the form of discussing future work and possible extensions / enhancements.

2 The Smart Homes for All project

The project SM4ALL (Smart Homes for All) is about developing and studying a middleware platform for inter-communication between smart embedded devices. This is done in an environment where the needs of persons are central and by using semantic and composability techniques [4].

The aim here is to be able to make guarantees about dynamicity, dependability and scalability and by the preservation of security and privacy of the platform and its users. All of this is translated to the scenario of private/home/building in the presence of different users that have different needs and abilities (e.g. aged disabled and young able bodied).

The platform developed by the SM4ALL group offers such features as scalability (number of embedded services to be managed) and dynamicity (managing insertion/removal of (new) sensors/devices/appliances). This is done by using techniques from peer-to-peer systems and autonomic computing. Also these techniques are offered built-in without need of a subsequent extension. Using P2P-techniques also allows the platform to cope, to some extend, with failing devices. Preserving privacy of human users is also guaranteed as well as security of the whole environment.

Domotics and homecare comprise the showcase of the middleware platform that the SM4ALL project is developing. And this Master-project can be used as part of a showcase for the SM4ALL project. JLocT can make information available about where a user currently resides in a building. This information could than be used by the SM4ALL-platform in decision-making. For example when a smart-home “knows” that a user currently resides in room A, it can decide to increase the temperature for that room. That basically is the connection between the two projects.

More information about the SM4ALL project can be found at [4].

3 Requirements for doing indoor location-tracking

Presented here are the project-requirements for JLocT using Sentilla JCreates [2]. These Sentilla motes are made available by the university of Groningen. Sentilla JCreates by default have a Java virtual machine running that can execute Java-written applications [5]. In principle the Sentilla motes should be used to form a WSN. This implies that one of the goals is to do location-tracking using a WSN. The indoor environment used for the experiments consists of two rooms (5161.0564 and 5161.0566) in the Bernoulliborg-building from the university of Groningen.

As is stated in the previous chapter domotics and homecare are choosen as the showcase for the middleware platform of the SM4ALL project. It is agreed that the showcases of the SM4ALL-platform could benefit a lot from knowing in what room a user currently resides. Therefore JLocT should have a granularity at room-level. With such information the SM4ALL-platform can already make decisions to control things like temperature, lighting, radio, television and airco in certain rooms.

A mobile node might be responsible for doing classification. On the other hand if the classification-algorithm requires much processing-power and memory a dedicated computer might be more appropriate. One way of comparing these types of architectural solutions is by looking at the efficiency. This means that the responsiveness of a solution is just as important as the effectiveness of the classification-algorithm used.

Deployment of JLocT should also be as easy as possible. If possible no floor-plan or site-survey should be performed. Environmental conditions might impact classification-accuracy depending on the location-tracking technique used. JLocT should therefore employ techniques that can at least mitigate the effects of certain environmental conditions.

Although some classes of the Sentilla API might suggest otherwise, the Sentilla JCreates by default only (appear) to support single-hop communication. Having multihop-communication allows a location-tracking system to cover bigger indoor areas by just adding new infrastructure-motes. This means that multihop-communication is also a requirement for JLocT.

It is regarded as a nice feature if “sufficient” location-tracking can be done using only radio-based techniques. The reason for this is since radio-communication is by default available in WSN’s. To be able to say something about the acceptability of classification-performance the term “sufficient” is introduced here. We are calling a success-rate of 60% as “sufficient” classification-performance. However better classification-performance should be possible when extensions are done to JLocT. This is especially the case if JLocT is really used in a showcase for the middleware platform of the SM4ALL project.

4 Related work

Several types of indoor location-tracking systems are developed over the years. These systems can use different techniques to do location-tracking. Some systems even combine these different techniques in an attempt to achieve better performance. Below these techniques are explained more in depth.

4.1 Radio-based technologies

Location-tracking systems can use several characteristics of a radio-signal to determine the position of a person in a building. Since one of the main characteristics of this project is the use of a WSN, the idea of using radio-based technologies comes naturally. One advantage of using radio-based technologies is that a person can wear a mobile node in his/her pocket. Such a mobile node can even be placed inside a wallet. This sometimes becomes problematic in systems that have mobile nodes depending on ultrasound or infrared. Below some of the most popular characteristics of a radio-signal that are used in location-tracking systems are discussed.

4.1.1 Angle of Arrival

The “Angle of Arrival” (AOA) basically is the angle between some reference direction and the propagation direction of a wave that come together [6]. The reference direction is called “orientation”. This orientation can be represented in a clock-wise direction starting at North with 0 degrees. The purpose of the orientation is for measuring against Angles of Arrival.

In a WSN there may be “unknown motes” whose location needs to be determined. This is done by computing the AOA of radio-waves that come from “beacon motes”. However it may be that the orientations of the “unknown motes” are not known when the system is deployed. Determining the position of “unknown motes” in either situation can be done by using triangulation [6]. Doing triangulation with and without orientation-information being available, can also be seen in Figure 1 below.

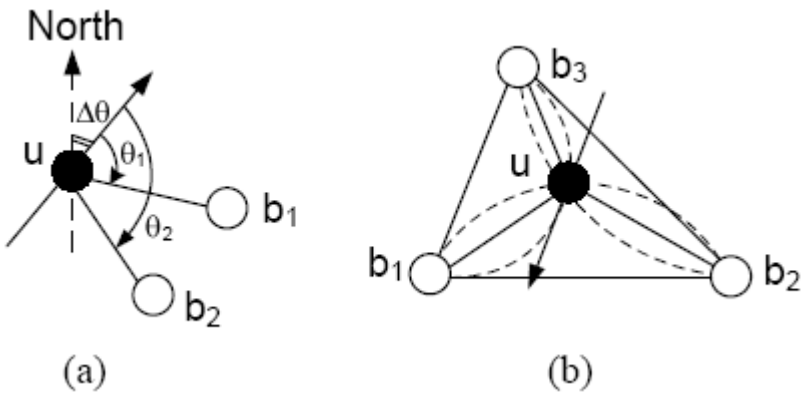


Figure 1: using triangulation in AOA localization from [6]: (a) orientation information available; (b) orientation information not available

In Figure 1 above the “unknown mote” is represented by “u”. The “beacon-motes” are called b1, b2 and b3. The orientation is represented by the uninterrupted line arrow and by $\Delta\theta$ in situation a. Finally in situation a the computed AOAs for “beacon mote 1” and “beacon mote 2” are represented by respectively θ_1 and θ_2.

Note that in the situation where orientation information is available at time of deployment, only two beacons are necessary to determine the location of the “unknown mote” [6]. This is shown in situation a in Figure 1 above. In the situation where orientation information is not available at time of deployment, at least three “beacon motes” are required. These three “beacon motes” are then used to both compute the orientation and to determine the location of the “unknown mote” [6]. This can be seen in situation b in Figure 1 above.

Sometimes AOA is used in combination with Time Of Arrival (TOA) to get better accuracy. Both techniques can use triangulation methods to determine the location of a mote or person. An example of such a hybrid-technique is introduced in [7]. The purpose of using AOA and TOA techniques in that article is for determining the position of a mobile phone in a GSM cell-phone network.

A disadvantage of using AOA in determining the position of a mote in a WSN is that it requires “unknown motes” to have antennas that are capable of detecting the angles of incoming signals [6]. Thus such antennas are capable of direction sensing.

4.1.2 Time of Arrival

Another way to use radio-signals in determining the position of a node is “Time of Arrival” (TOA) [8]. The concept of TOA can be explained as the time it takes for a radio-signal to travel from a sender to a receiver. The idea is that the longer it takes for a radio-signal to travel from a sender to a receiver, the further they are separated.

As suggested in Section 4.1.1 TOA, like AOA, can be used in combination with triangulation methods to determine the position of a node in the network [7]. However another popular technique used is TOA tri-lateration. A fundamental difference is that in triangulation angles are computed, while in tri-lateration the distance is computed. In general this requires at least three “known nodes” whose positions are known.

The “unknown node” starts the process by sending a radio-signal. For each of these “known nodes” the TOA is computed respective of the “unknown node”, whose location is not (yet) known. From the TOAs distances can be computed because the propagation-velocity of a radio-signal is known (light-speed in free space) [9][10][11][12].

The Distances can then be used to create circular plots (2D) or spherical plots (3D) around “known nodes”. The intersection of these plots indicates the location of the “unknown node” [12].

An 2D-example of this is shown in Figure 2 below. Here the “known nodes” are A, B and C. The “unknown node” is denoted by X. The distances from the “known nodes” to the “unknown node” X are denoted by D_A, D_B and D_C respectively.

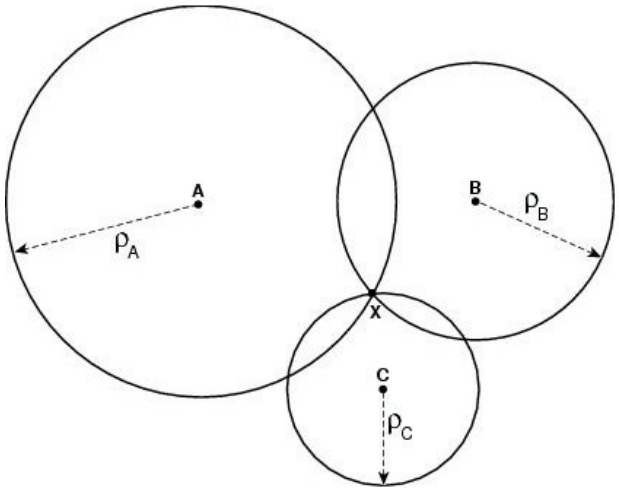


Figure 2: TOA tri-lateration-technique applied using 2D circles from [12]

Note that time based methods (such as TOA) rely on accurately computing the time needed between sending a radio-signal by a transmitter and receiving it at the receiving-side [8]. However it is well known that the speed of a radio-signal in a vacuum equals the speed of light [9][10][11][12]. This means that all stations involved in TOA-localization should have very precise time-synchronization [12]. This is especially the case for the mobile-device. This usually means that specialized hardware for time-synchronization is needed.

Furthermore location-tracking systems that use TOA can give poor accuracy in situations where a large amount of multipath-interference or noise may exist [12]. An example of a TOA system is

the Global Positioning System (GPS) as is also mentioned in [12]. In GPS atomic clocks are used for time-synchronization.

4.1.3 Time Difference of Arrival

Localization-systems that employ Time Difference of Arrival (TDOA) do not require the mobile sending node to have its time synchronized with a time-source [12]. In TDOA a transmission with some unknown starting-time is received by the fixed nodes. These receiving fixed nodes do require time-synchronization.

Computing TDOA for location-tracking usually means making use of a mathematical concept known as hyperbolic lateration [12]. For this to work at least three receiver nodes are required. As an example Figure 3 below from [12] assumes that X is the mobile node. X transmits a message that is arrived at receiving nodes A and B with times T_A and T_B respectively. Now the TDOA for the transmitted message is computed between the locations of node B and node A. The value is stored in a positive constant *k*:

$k = TDOA_{B-A} = | T_B - T_A |$

The value *k* can be used for the construction of a hyperbola having focus-points at the locations of nodes A and B. The most likely position of the mobile node X is represented as a point along this hyperbola. To get a more accurate indication of the location of X a third receiving node C is used. The TDOA is also computed between node C and A:

$k_1 = TDOA_{C-A} = | T_C - T_A |$

Now that we know k_1 we can compute a second hyperbola having focus-points at the locations of nodes A and C.

Figure 3 below from [12] shows how the position of mobile node X is resolved. This is done using the intersection of the two computed hyperbolas $TDOA_{C-A}$ and $TDOA_{B-A}$.

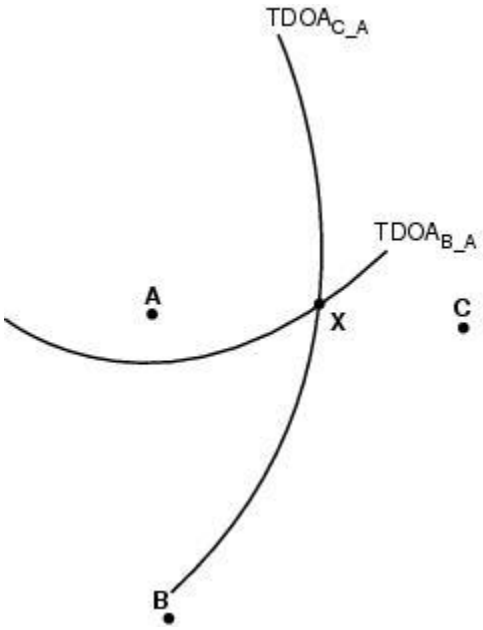


Figure 3: hyperbolic tri-lateration using computed TDOAs from [12]

It may occur that more than one solution is found using the TDOA hyperbolic tri-lateration technique described above. In such a situation a fourth receiving node may be added giving rise to a third hyperbola to enhance the accuracy [12].

Using TDOA can give rise to high accuracy in determining where mobile nodes are located. However as is mentioned in [13] the fixed receiver nodes do require some sort of periodic time-synchronization. Otherwise the localization-accuracy is affected. Several algorithms are developed that perform time-synchronization for TDOA-based systems. Some known examples of these are [14][15].

Either way TDOA-based systems do require that time-synchronization occurs for fixed receiving nodes. This means that extra network-traffic occurs for sending and receiving synchronization-packets. An exception of this is of course the scenario where the fixed motes are directly connected to the same external time-source. But this would of course require extra expenses since a (atomic-)clock would have to be bought.

One interesting example of a system that uses TDOA for location-tracking is a “wildlife radiolocation system” [16].

4.1.4 Received Signal Strength

The received signal strength (RSS) is the power of a received radio-signal. This can also be used as a metric in determining the distance between sender and receiver. Received Signal Strength Indicator (RSSI) is a commonly used term to denote the power that is present in a received radio-signal [17].

An advantage of using this technique is that no special hardware, like an atomic-clock, has to be purchased. It is therefore not surprising that extensive research is done on location-tracking / localization using RSS.

One popular method that uses RSS to determine the location of a mobile node is using a path-loss model [12][18]. As is mentioned in [18] propagation models indicate that average power of a received signal actually decreases logarithmically with distance. This is stated to be irrespective of whether the radio signals travel indoors or outdoors [18].

The RADAR-system [19] is an example of a location-tracking system using a propagation-model. It combines the use of a propagation-model with pattern-matching to determine the position of a mobile node. The article [19] about the RADAR-system also indicates that doing empirical signal-strength measurements instead of using a propagation-model gives better results.

This is in line with an article about the MERRIT-system [20]. Here it is stated that radio-signals that are transmitted can take different paths to reach a receiving node. The reason that this can occur is because radio-signals can travel indirectly or directly due to reflections from indoor surfaces.

This concept is called "multipath-interference" and can result in lower or higher readings of signal-strength than is the case if the radio signal travels in free space only. This means that it may be very difficult to find a theoretical propagation-model that is accurate for a certain indoor-environment [20].

Systems like RADAR [19] and SpotON [21] can provide fine-grained localization-granularity in the form of coordinates.

However as is argued in [20] many systems require only granularity at the room-level. This means that once the exact coordinates are computed, a floor-plan should be used for determining the room a mobile-node is in. A better solution would compute the room directly without using a floor-plan [20].

For pattern-matching systems like RADAR [19] another cost is introduced in the form of a site-survey that has to be done. Doing such a site-survey requires additional time when deploying the system to a specific indoor-environment. This is in contrast to a system like MERRIT [20] that does not require this. However the MERRIT-system only provides granularity at the room-level. But as is mentioned before, many systems only require granularity at the room-level. The location-tracking system for the SM4ALL-project is no exception.

The MERRIT system actually tries to statistically cope with multipath-interference. This is done by letting infrastructure nodes receive multiple messages and computing a average RSSI before sending it to a location-monitor. This concept is called “temporal diversity” [20]. Since each

infrastructure node represents a room, multiple averages coming from infrastructure nodes are used in determining the location of a mobile node. The average is computed of multiple averages representing the same room. This concept is called “spatial diversity” [20]. The room the mobile node is deemed in is that room with the highest cluster-average associated with it.

Another example of a system that uses RSS is Ferret [22]. Ferret can do localization using a potentiometer technique and a RSSI technique. For deploying Ferret in a new indoor environment a calibration-tool has to be run.

For the potentiometer technique the calibration-tool creates a table that maps the power level used in transmission with the communication ranges of a radio-signal (distances in feet). The potentiometer technique of course requires some API giving access to power-settings of the transceiver. In [22] a open-source embedded operating-system called TinyOS [23] is used that allows access to the power-settings.

The general idea of the RSSI technique of Ferret is actually quite similar to that used by MERRIT [20]. In Ferret the mobile node sends out 5 messages. Infrastructure nodes respond to each message. The mobile node records RSSI values and ID’s for every packet received from the infrastructure nodes. Then the mobile node computes the average RSSI for each infrastructure node that it heard from (“temporal diversity”). Next it identifies the three nearest neighbors by identifying the largest averages. The data of these three are sent to the base station / location-monitor for determining the actual location (“spatial diversity”). The location is then computed by the base station by determination of the point in state-space having the minimum sum of errors. Here the sum of errors is computed by summing the computed errors for the identified three nearest neighbors.

RSS-based systems sometimes use RFID tags and receivers to create an infrastructure. These systems typically can give fine-grained granularity in the form of coordinates. An example of such a system is LANDMARC [24]. In LANDMARC the distance to fixed reference tags is computed by measuring RSS. The smaller the distance to a certain reference tag the closer that person is regarded to be. The deployment of LANDMARC requires that reference tags have to be placed as well as readers in each room of the indoor-environment. Another example of a RSS-based system that uses RFID tags is that of the VIRE approach [25].

4.2 Hybrid and other technologies

Location-tracking systems can use various different technologies other than just radio-signals. Discussed below are several systems that employ techniques for location-tracking, other than the way radio-signals are used in the previous section.

4.2.1 Bat

Bat-system [26] [27] uses ultra-sound in determining distances. The times-of-flight of the ultrasound pulse from the mobile node to the receivers is converged into a distance (to a receiver). If it is possible to get the distance from three or more receivers that are collinear, the position in 3D space can be found. This is done by using trilateration in combination with three or more measurements [26] [27]. According to [27] an accuracy of around 3 cm in three dimensions is achieved.

4.2.2 Active badge

The Active-badge system [28] uses infrared signals for signaling between a mobile badge and fixed positioned sensors. Each badge periodically sends a unique signal that can be received by a whole network of sensors. One advantage of using infrared signals is that they cannot penetrate walls separating rooms. This is in contrast to the use of radio-signals that do have this feature.

4.2.3 SignPost

In [29] a mobile augmented reality (AR) system called SignPost is presented. SignPost is able to guide a user through an unfamiliar building to a destination-room. To do this two techniques are used for tracking the location of a user. First an indoor-environment is equipped with wall-mounted markers that can be observed by a head-mounted camera. The markers identify physical locations in the indoor environment.

However it may happen that a user is walking without seeing any marker at that point in time. The system then relies on an inertial tracker, that provides angular information with respect to the last observed marker. This information is updated when a new marker is observed.

The proposed tracking method seems to depend heavily on the density between markers. It is suggested in [29] that a marker should be placed each 2-3 meters in a hallway. Also each room should have at least 4 markers.

The angular precision of optical tracking sometimes seems to suffer from a jitter. The effects of this jitter can be decreased by using more markers. Also the correct recognition of markers seems to depend heavily on the lighting conditions in rooms. Finally the inertial tracker at times leads the user away from his path by drifting in another direction. This problem is corrected by going to a wall and observing a marker from a close position. This then adjusts the path shown on the view of the user to the correct path again.

4.2.4 Cricket

Cricket is described in [30] as being a “location-support system”. An interesting feature of Cricket is that it does not rely on any centralized management or control. Instead Cricket is

decentralized. Cricket also does not explicitly tracks the location of a user. Instead applications that run on static and mobile nodes can "learn" what their physical location is. This is done by using listeners that monitor information coming from beacon-nodes spread throughout an indoor-environment. There is no explicit coordination between the beacon-nodes. If a node “knows” what its physical location is, it can decide which other node(s) should be notified of this information.

Cricket makes use of both radio signals and ultrasound. Beacon-nodes are attached to walls and ceilings in an indoor-environment. The beacon-nodes publish location-information by transmitting a radio-signal. When such a advertisement is done by radio-signal, an ultrasonic pulse is transmitted concurrently. Listeners that are used by applications running on static or mobile nodes receive these radio- and ultrasonic signals. When this happens the listeners match the radio-signals to the ultrasonic signals. Next an estimate of the distances to different beacons is computed. This is done by computing the TDOA (Time Difference Of Arrival) for the radio- and ultrasonic signals. From this information the physical location can be derived.

4.2.5 Bluetooth location network

The use of Bluetooth for localization is what is happening in the “Bluetooth location network” (BLN) described in [31]. In the BLN a mobile unit can be tracked using areas called cells. Each cell is represented by an infrastructure node. In the BLN infrastructure nodes send inquiry-messages and collect response-messages from mobile nodes. The addresses of the Bluetooth stations detecting mobile nodes are transmitted over multiple infrastructure nodes to a master infrastructure node. The master node determines the location of the mobile node. This can be done since the master node knows the location of each cell that corresponds to the received addresses. Ultimately a mobile node can be tracked at the granularity of a region. This region is created by the intersection of transmissions of multiple infrastructure nodes.

4.3 Discussion

One of the criteria of this research-project is the use of a WSN. Such a WSN by default has radio-capability. Therefore it is decided to only consider radio-based technology for this project, since no extra hardware is required. Nevertheless the discussed other types of systems give a more general view of what is currently possible in the area of location-tracking.

The papers about RSS-techniques are found to be especially interesting. The reason is because the signal-strength of a radio-signal is easy to obtain in a WSN. Also no specialized hardware is required and no extra network-traffic occurs for synchronization. The latter being the case for some TDOA-implementations [14][15].

Also since a show-case for SM4ALL requires classification at the room-level (and thus also JLocT) no finer-grained granularity is required. This means that a patter-matching system is

considered overkill. The reason is because it provides fine-grained granularity at the expense of a required site-survey during the deployment-phase.

One RSS-based paper about the MERRIT-system [20] is found to be particularly interesting. The reason lies in the easy deployment of the system and its capabilities to cope with “multipath-interference”. It is for these reasons that the general architecture of JLocT is largely based on the MERRIT-system. The next chapter covers the theory about the location-tracking technique used in JLocT. A comparison of the some discussed technique’s and systems can be found in Table 1 below. Note that not all the mentioned characteristics of the technique’s and systems are shown in the table.

Technique / system	Technology used	Granularity / precision	Extra note
Angle of Arrival (AOA)	Radio-signal	Fine-grained: due to triangulation	"Unknown motes" require antennas that are capable of detecting angles of radio-signals.
Time of Arrival (TOA)	Radio-signal	Fine-grained: intersection-area of circles (2D) or spheres (3D)	Requires time-synchronization in the form of specialized hardware (atomic clock).
Time Difference of Arrival (TDOA)	Radio-signal (sometimes in combination with an ultrasound signal)	Fine-grained: intersection of hyperbolas in tri-lateration	Receiving fixed nodes do require time-synchronization (by using specialized hardware or by extra network-traffic).
Received Signal Strength (RSS)	Radio-signal	Room-level, Fine-grained: coordinates (due to pattern-matching or use of RFID-tags)	Easy to implement, requires no extra hardware (except for in the case of using RFID-tags).
Bat	Ultra-sound	Fine-grained: coordinates with accuracy of around 3 cm in three dimensions	Motes in a WSN can not by default send/receive Ultra-sound signals.
Active-badge	Infrared signal	Room-level	Motes in a WSN can not by default send/receive Infrared signals.
SignPost	Camera in combination with wall mounted markers, inertial tracker	Coordinates: wall mounted markers	Motes in a WSN do not by default have camera's or an inertial tracker.
Cricket	Radio-signal and Ultra-sound	Fine-grained: use of TDOA	Motes in a WSN can not by default send/receive Ultra-sound signals.
Bluetooth location network	Bluetooth	Area created by intersection of blue-tooth signals. Worst-case deviation of 10-m in scatternet-topology	Motes in a WSN do not by default have Bluetooth.

Table 1: comparing different location-tracking techniques and systems

5 Location-tracking in JLocT

RSS-based techniques are the preferred choice for location-tracking in JLocT. The reason lies in the fact that WSN's by default have radio-communication. Additionally signal-strength is easy to obtain as no specialized hardware is required. Of course also no additional network-traffic occurs for (time-)synchronization as is the case in some TDOA-systems [14][15]. Therefore RSS-based techniques are chosen as the main type of technology used in JLocT for doing location-tracking.

As is stated in the discussion of the previous chapter, the method that JLocT uses for location-tracking is based on that of the MERRIT-system [20]. The reason why the MERRIT-system is found to be important, is because it can be quickly deployed and employs coping-techniques for environmental interference. In this chapter the location-tracking technique and its concepts used in JLocT are investigated. At times a comparison may be made with regard to the MERRIT-system.

5.1 Temporal diversity

Radio-signals may be impacted by indoor environmental effects. One particular type of environmental effect is "transient blocking of a radio-signal" [20]. This comes down to an obstacle temporarily blocking the path of a radio-signal between sender and receiver.

To mitigate the indoor environmental effects on radio-signals, and in particular transient blocking, JLocT and the MERRIT-system [20] employ "temporal diversity". A comparable system that does not use "temporal diversity" would just receive signal-strength measurements from receivers and directly use these for classification by a location-monitor.

However for each receiver JLocT and MERRIT first receive multiple measurements over a time-period. Next each receiver computes an average over its received measurements. These averaged measurements are then used for classification by a location-monitor. This concept is called "temporal diversity" [20].

5.2 Spatial diversity

In multipath-interference radio-signals sent by a transmitter can travel along different paths before reaching a receiver [20]. These radio-signals may bounce of from indoor surfaces. Ultimately this may lead to lower or higher readings of signal-strength. This means that systems that rely on a theoretical signal-strength attenuation model may suffer from bad classification-performance [20]. In an attempt to cope with the problem of multipath-interference both JLocT and MERRIT employ the concept of "spatial diversity" [20]. This means that the effects of multipath-interference can be statistically filtered out.

A way to do this is by using the signal-strength measurements from multiple locations in a room [20]. This is in contrast to the scenario where each room is represented by a receiver positioned on a single location. So for employing spatial diversity in both JLocT and MERRIT a room should be represented to the location-monitor by multiple fixed infrastructure nodes.

5.3 Doing classification

In JLocT the mobile roaming mote periodically broadcasts a locationRequest-message. This locationRequest-message has a sequence-number associated with it. After broadcasting 5 messages with the same sequence-number a new number is used during sending.

Receiving fixed motes derive a signal-strength value from the transmission. These RSSI-values are stored by a receiving fixed mote until a new sequence-number is observed. At this point the fixed motes compute an averaged RSSI-value for the roaming mote. This averaging of the RSSI-values is an implementation of the concept of “temporal diversity” as discussed in Section 5.1.

Averaged RSSI-values coming from a single or multiple fixed motes are sent to the location-tracker by use of the multihop-network. So averaged RSSI-values coming from multiple fixed motes can contain information about the location of a single roaming mote. The consideration of location-information coming from multiple fixed motes for classification is an implementation of the concept of “spatial diversity”. For more information about this concept the interested reader is referred to Section 5.2.

Threads are running at the location-tracker for each roaming mote in the system. Periodically such a thread computes a new location for the roaming mote based on the received averaged RSSI-values. As is discussed in Section 5.2 a room in JLocT should be represented by multiple fixed motes, thus providing “spatial diversity”. This means that the location-tracker can receive averaged RSSI-values coming from different fixed motes but representing the same physical room.

For each room these averaged RSSI-values are then used to compute a individual value representing that room. This is done by computing the average over the received averaged RSSI-values for a certain room. The MERRIT-article [20] calls these values “cluster-averages”. Both JLocT as well as the MERRIT-system determine that a roaming mobile node is in the room with the highest cluster-average associated with it.

6 Architecture

Here a general overview of JLocT is presented in the form of two architectural views.

6.1 Process view

Presented here is a view on how different processes are communicating with each other. The communicating processes may run on different nodes.

SD01: creating a routing-topology

Description:

Below you can see a sequence-diagram showing a simple situation of how the creation of a routing-topology and deployment of location-information can take place. The sequence-diagram shows one location-tracker and two fixed motes that try to establish a routing-topology. First the computer running the location-tracker broadcasts a “STARTFLOODING”-message. In this situation the message is received only by FixedMote1, who responds by sending a “PARENTFLOODING”-message. Now the location-tracker adds it as a child-node in the spanning-tree. FixedMote1 in turn broadcasts a “STARTFLOODING”-message that is received by FixedMote2. Now FixedMote1 has done its mandatory transmissions. It notifies the location-tracker of this event by sending a “MOTECONVERGED”-message. This is actually used by the location-tracker to mark FixedMote1 as being “converged”. At this point FixedMote2 responds to FixedMote1 by sending a “PARENTFLOODING”-message. This is used by FixedMote1 to mark FixedMote2 as its child-node. Next FixedMote2 sends a “MOTECONVERGED”-message to FixedMote1. This message is forwarded by FixedMote1 to the location-tracker. The location-tracker then marks FixedMote2 as being “converged”. Now that all fixed motes are marked “converged”, the location-tracker begins deployment of location-information. This is done by sending a “DEPLOYLOCATION”-message to FixedMote1. FixedMote1 stores the “value-member” of the message. Next the FixedMote1 sends a “LOCATIONDEPLOYED”-message to the location-tracker as a notification. The location-tracker marks FixedMote1 as “initialized”. The location-tracker now sends a “DEPLOYLOCATION”-message to FixedMote2 by using FixedMote1 as a relay. FixedMote2 also stores the “value-member” of the message. Now FixedMote2 sends a “LOCATIONDEPLOYED”-message to the location-tracker by using FixedMote1 as a relay. The location-tracker marks FixedMote2 as being “initialized”. Now that all fixed motes are marked as “initialized” the routing-topology is created.

Note that for reasons of simplicity “ACKRECEIPT”-messages are not shown. These are acknowledging receipt of incoming messages and are part of the “reliable messaging”-implementation. More on the “reliable messaging”-implementation can be found in Section 8.3. Also for reasons of simplicity the broadcasting of the “STARTFLOODING”-message by

FixedMote2 is omitted here. In this simple situation no mote would respond to such a broadcast. The reason is because FixedMote2 is a leaf-node in the spanning-tree / routing-topology.

Trigger: a user presses a “create-routing topology”-button at the location-tracker resulting in a broadcast of a “STARTFLOODING”-message.

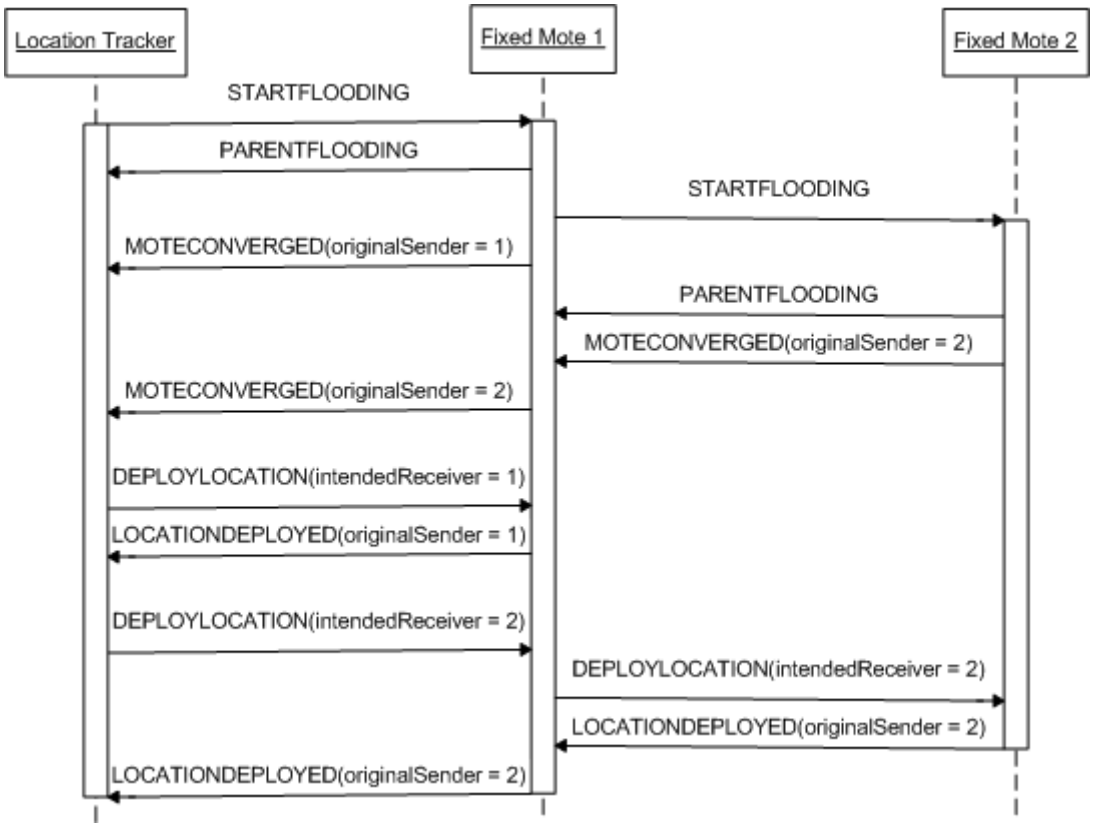


Figure 4: sequence-diagram of traffic during creation of a routing-topology and deployment of location-information

SD02: updating the location of a roaming mote

Description:

The sequence-diagram below shows a simple situation where location-tracking is done using a location-tracker, two fixed motes and one roaming mote. First the computer running the location-tracker sends a ProgramMessage to the roaming mote, thus activating it. Now that the roaming mote is activated it broadcasts a locationRequest-message with sequenceNumber = 0. This is repeated five times. The RSSI-value of each received locationRequest-message is stored by the fixed motes. At some point the roaming mote sends a locationRequest-message with new

sequence-number = 1. When this happens the receiving fixed motes compute an average RSSI for the roaming mote. Next this average is being sent in a locationUpdated-message along with other information to the location-tracker by means of the multihop-protocol. Other information that is stored in the same data-member of a locationUpdated-message are the ID of the roaming mote and the sequence-number. The location-tracker receiving the locationUpdated-messages can just store the information. Periodically a thread running on the location-tracker for the roaming mote computes a new location using the received information. When this happens the GUI of the location-tracker is also updated.

In this example it is assumed that the roaming mote is within sending range of both fixed motes. It is also assumed that, for fixed mote 2 to communicate with the location-tracker, traffic has to go through fixed mote 1. The broadcasting of a locationRequest-message done by the roaming mote is actually displayed here as two lines going to both fixed motes. Note also that this broadcasting of messages with sequence-number = 1 is of course also repeated five times. But for reasons of simplicity this is omitted from the diagram. Also omitted for simplicity-reasons are the “ACKRECEIPT”-messages. These are acknowledging receipt of incoming messages and are part of the “reliable messaging”-implementation.

Trigger: a user presses an “Activate roaming-motes”-button at the location-tracker resulting in ProgramMessage-messages being sent to roaming motes.

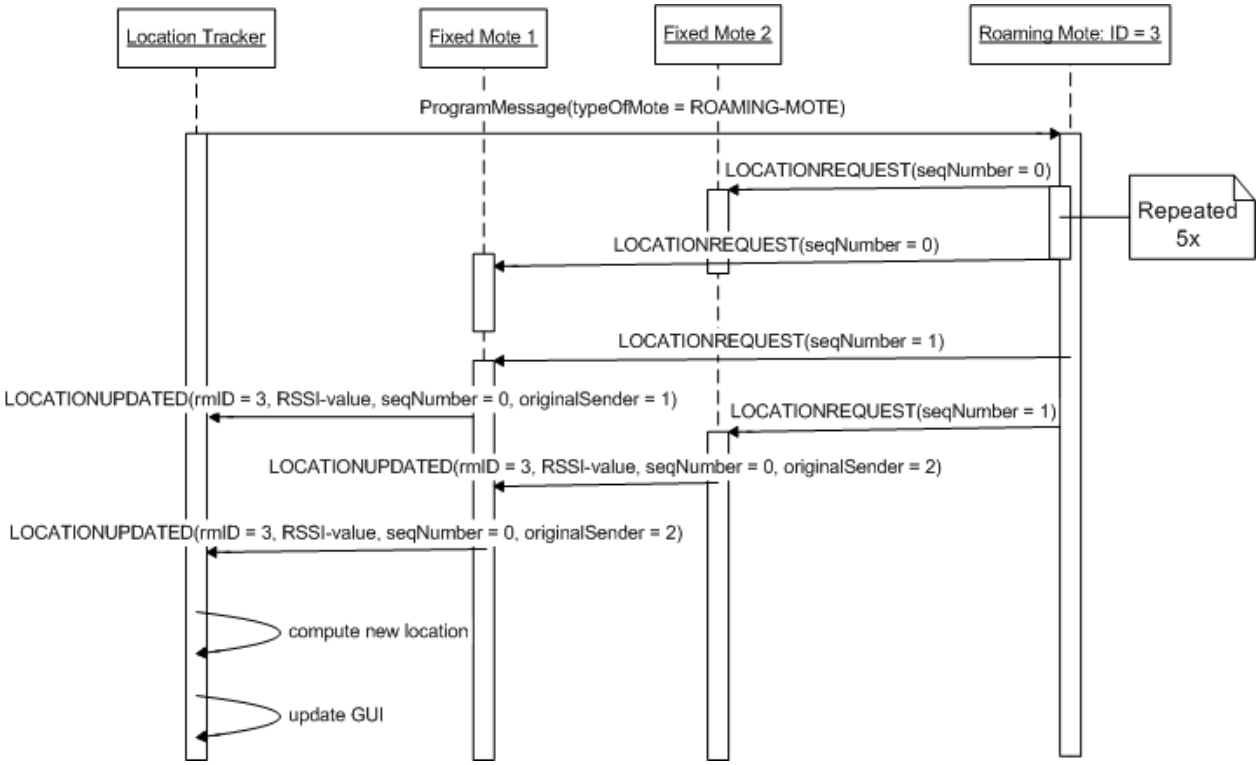


Figure 5: sequence-diagram of updating a roaming mote’s location

6.2 Deployment view

Figure 6 on the next page shows a deployment-diagram of JLocT.

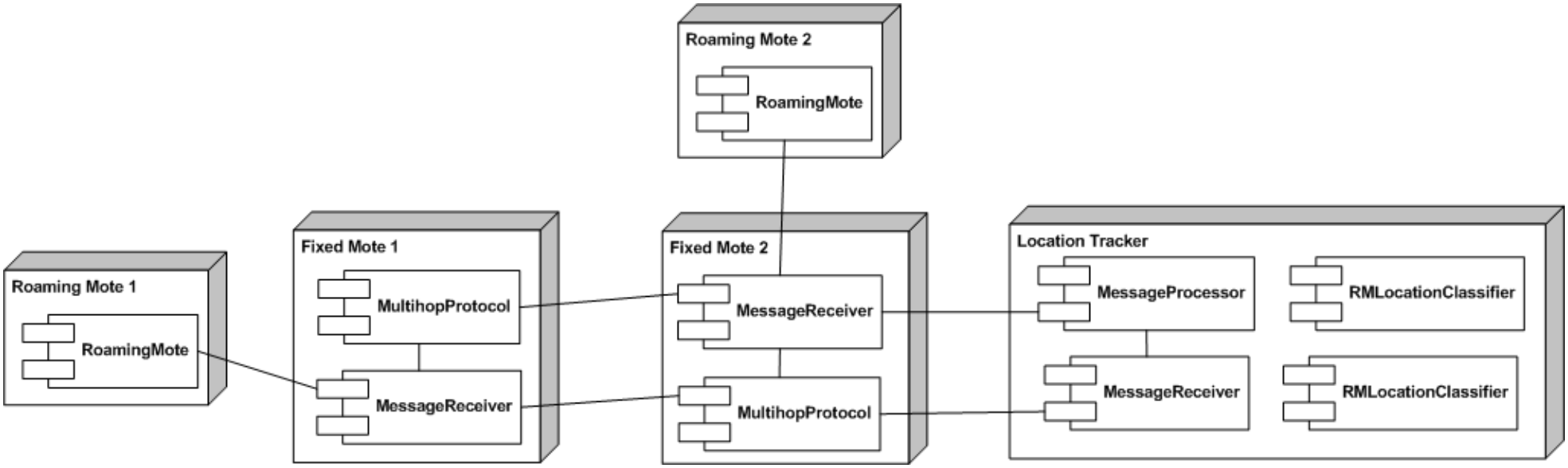


Figure 6: deployment-diagram of JLocT

It shows a simple situation of two roaming motes, two fixed motes and one location-tracker. The roaming motes only have a single threaded component called RoamingMote. This embodies all functionality of the roaming mote doing periodic sending of locationRequest-messages.

Fixed motes only have a MessageReceiver-thread and a MultihopProtocol-thread. The latter uses the former to support the multihop-network. The LocationTracker also has a MessageReceiver-thread that is accessed by a MessageProcessor-thread to do message-processing.

Note that the MessageProcessor updates a data-model when it receives location-updates. The RMLocationClassifier-threads periodically accesses this data-model when it computes the location for a roaming mote. Because communication is indirect, no lines are shown between the RMLocationClassifier-threads and the MessageProcessor-thread. For each roaming mote in JLocT the location-tracker has a RMLocationClassifier-thread running.

7 Flooding and creating a routing-topology

When motes are being moved in a building they may at some point be out of direct sending-range of the computer/laptop that normally receives the messages. To cope with this problem multi-hop protocols are designed. These protocols allow intermediate motes to forward messages that are not meant for them.

However this requires the intermediate motes to “know” to whom they should send a message if the direct recipient is also not the intended recipient. In the situation of JLocT communication only goes from a mote to the laptop or from the laptop to a mote (or multiple motes). That is why a spanning-tree is used in JLocT as an abstraction to support this communication.

One popular algorithm that can be used for creating a spanning tree in a distributed system is the flooding-algorithm [32]. The flooding-algorithm is also shown in pseudo-code below which comes from [32]:

flooding algorithm to construct a spanning tree:
code for processor p_i, $0 \leq i \leq n-1$. **r = index of root**

Initially *parent* $= \perp$, *children* $= \emptyset$, and *other* $= \emptyset$.

```
1:  upon receiving no message:
2:      if p_i = p_r and parent = ⊥ then          // root has not yet sent ⟨M⟩
3:          send ⟨M⟩ to all neighbors
4:          parent := p_i

5:  upon receiving ⟨M⟩ from neighbor p_j:
6:      if parent = ⊥ then                        // p_i has not received ⟨M⟩ before
7:          parent := p_j
8:          send ⟨parent⟩ to p_j
9:          send ⟨M⟩ to all neighbors except p_j
10:     else send ⟨already⟩ to p_j

11: upon receiving ⟨parent⟩ from neighbor p_j:
12:     add p_j to children
13:     if children ∪ other contains all neighbors except parent then
14:         terminate

15: upon receiving ⟨already⟩ from neighbor p_j:
16:     add p_j to other
17:     if children ∪ other contains all neighbors except parent then
18:         terminate
```

Figure 7: pseudo-code of original flooding-algorithm from [32]

In a situation where synchronous communication is guaranteed, this algorithm always converges thus creating a spanning-tree [32]. Because of the convergence-guarantee this flooding-algorithm is found to be a very interesting candidate for creating a multi-hop protocol.

However running the flooding-algorithm on the Sentilla motes it turns out that the algorithm does not always converge. At first it is suspected that it is just a bug and that the algorithm in [32] deviated from the implementation on the Sentilla-motes. But a closer inspection suggested that reliable synchronous communication is not guaranteed. At times messages just are not being sent at all.

One method of trying to cope with this is to implement a form of reliable messaging. However according to the “Sentilla Programmer’s Guide” [33] the garbage collector used on the Sentilla motes can lose packets and interrupts when it blocks the virtual machine. The latter usually is done when “cleaning up” its memory. This possible loss of packets and interrupts may explain why the implementation of reliable messaging sometimes is not reliable, of course not considering here the possibility of a bug in the application-code. For more information about the implementation of reliable messaging see Section 8.3.

The lack of guaranteed reliable messaging, and thus not guaranteeing synchronous communication, leads to the flooding-algorithm not being able to guarantee convergence on the Sentilla-motes.

Therefore it is decided to try to have a “best as possible” approach for creating a routing topology. This means that the original flooding-algorithm is modified to do as little communication as possible. In the original flooding-algorithm described above an individual mote had to know who its neighbors are to determine whether it should converge.

Since only the location-tracker “knows” all the motes and their state, the latter also should “know” when all motes can stop the flooding-algorithm. In such a situation the location-tracker begins deploying indices representing location-labels of the locations-tab in the location-tracker application. Each fixed mote is associated with such an index, thus “knowing” what location it represents.

If a fixed mote receives such a deployLocation-message it quits the loop it is in and terminates the flooder-code. From there on the MultihopProtocol-thread takes over, since it knows what to do with the location-indices. Termination of the flooder-algorithm below is shown in the event of receiving a deployLocation-message.

The “new flooding-algorithm” is shown in pseudo-code below:

```
On receival of STARTFLOODING-message msg
      if parentNode == -1
            parentNode = msg.originalSender
            send PARENTFLOODING to msg.originalSender
            send STARTFLOODING to everyone
            send MOTECONVERGED to parentNode
      end if

On receival of PARENTFLOODING-message msg
      if !containedInChildNodes(msg.originalSender)
            childNodes.add(msg.originalSender)
      end if

On receival of MOTECONVERGED-message msg
      if !containedInChildNodes(msg.originalSender)
            childNodes.add(msg.originalSender)
      end if
      if parentNode != -1
            msg.intendedReceiver = parentNode
            msg.receiver = parentNode
            msg.sender = MAC_ADDRESS_OF_THIS_MOTE
            send msg to parentNode
      end if

On receival of DEPLOYLOCATION-message msg
      stop flooder-algorithm
```

A fixed mote can indicate to the location-tracker that it performed all of the code for the startFlooding-message by sending a moteConverged-message. When an intermediate mote receives such a moteConverged-message it forwards it to its parent. This is repeated by the next mote until finally the location-tracker receives the moteConverged-message.

The “new flooding-algorithm” does not use “ALREADY”-messages that are used in the original flooding-algorithm described before. The reason for this is that in the “new flooding-algorithm” the location-tracker determines whether all fixed motes should stop their flooder-code. Whereas in the original flooding-algorithm each mote could and should determine this. When the location-tracker decides that all motes should stop the flooder-algorithm, it starts deployment of location-information.

8 Implementation

JLocT is a distributed system where, apart from the roaming mote, each node runs a multi-threaded application. This chapter covers implementation-details of JLocT. Also covered are the implementation-details of a experiment. In that experiment the roaming mote actually does classification.

8.1 Different components

When considering the implementation of JLocT we must keep in mind that the system is divided into software running on two types of physical devices. A laptop or personal computer can run the Location-Tracker software that an user can use to interact with the system. The other type of physical device is a type of mote called a Sentilla JCreate [2]. These motes can either take on the role of an roaming/mobile mote or an fixed/architectural mote.

8.1.1 Location Tracker

The location-tracker is a multithreaded application that functions as the client to an Wireless Sensor Network (WSN) comprised of Sentilla motes (a.k.a Sentilla JCreates). It allows an user to manage locations, fixed motes and roaming motes. This GUI-application has three tabs containing corresponding functionality. Note that each time the location tracker starts it reads serialized data from the following files (these need to be present in the root of the project directory):

- *locations.dat*: contains data that is loaded in the locations-tab;
- *fixedmotes.dat*: contains data that is loaded in the fixed-motes tab;
- *roamingmotes.dat*: contains data that is loaded in the roaming-motes tab.

If the application is being closed, it automatically updates these files with the changes made during the execution of the location tracker.

Tab: Locations

The locations-tab is shown in Figure 8 below.

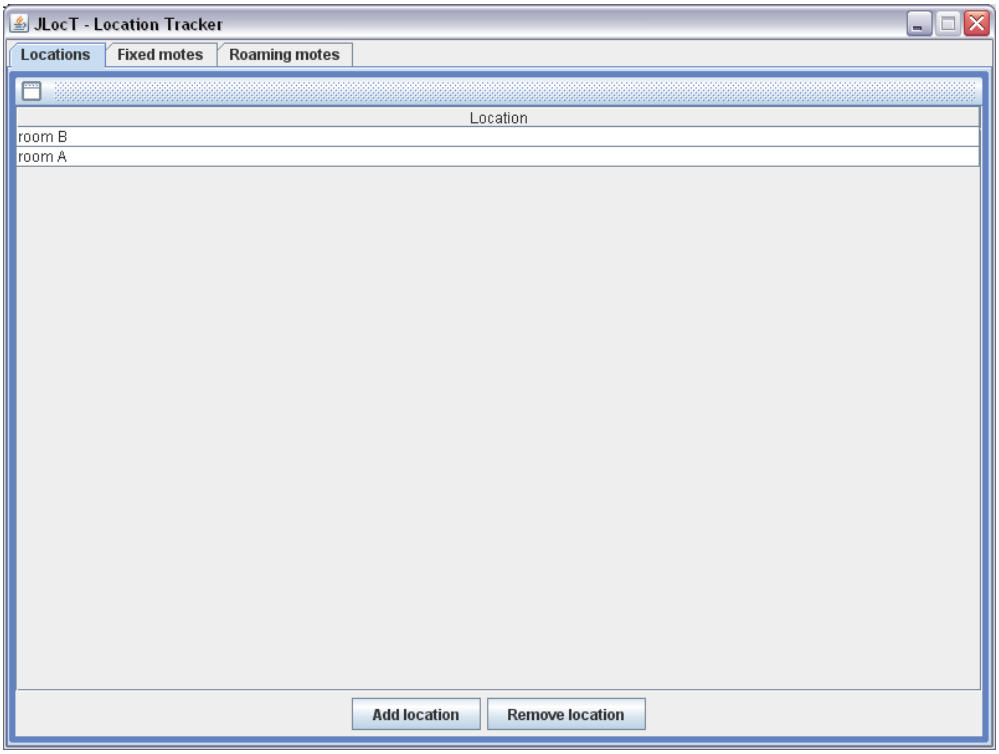


Figure 8: locations-tab of Location Tracker

When JLocT is first deployed in an building a label should be entered for each room that should be known to the system. This comes down to all the rooms in the building where fixed motes are placed. The location-tracker is then able to determine in which room an roaming mote is.

In the locations-tab the user has the option to add new locations, modify or remove existing locations. The latter operation can only be done if no entry in the fixed-motes tab is associated with the location to remove. This restriction is placed to uphold referential integrity. Also every room-name entered should be unique. If this is not the case an error-message appears and the entry is not allowed to commit.

Adding or removing entries in the locations-tab is done by pressing the designated buttons. Changing location-labels can just be done by selecting an entry, making the changes and pressing ENTER. The moment this is done any entries in the fixed-motes tab that are associated with the location are now also being updated.

Tab: Fixed motes

The fixed-motes tab is shown in Figure 9 below.

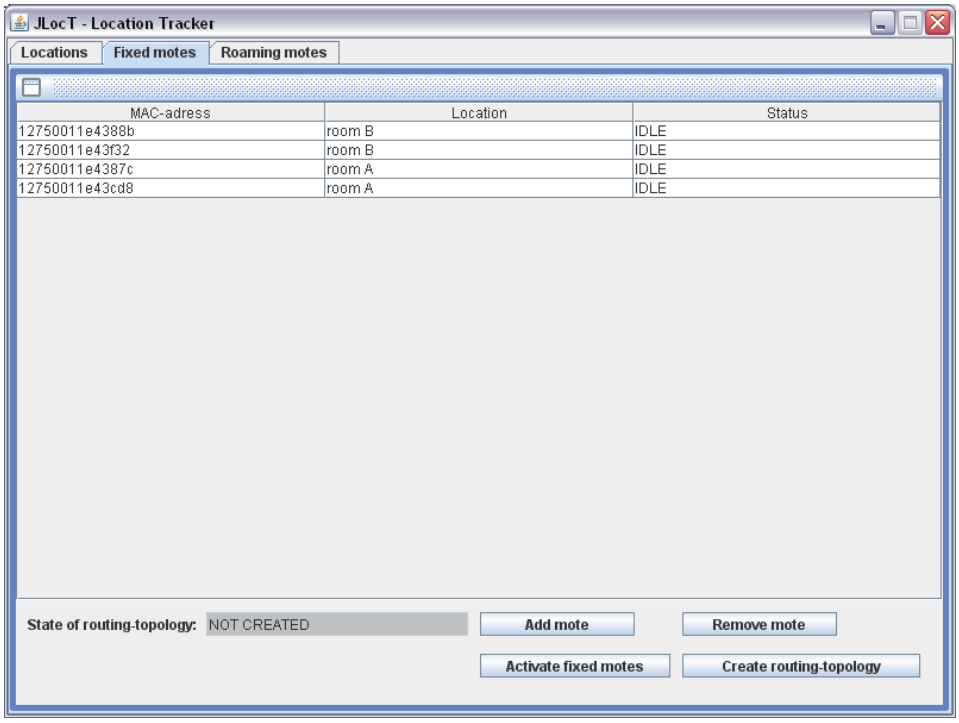


Figure 9: fixed-motes tab of Location Tracker

The fixed-motes tab allows the user to manage the motes that are located on fixed positions in the building. These fixed motes provide a communication-infrastructure that allows roaming motes to communicate with the location-tracker. In the fixed-motes tab the user is allowed to:

- *Add new fixed motes*: this requires entering a unique MAC-address. It is also mandatory to associate a fixed mote with an existing location of the locations-tab. This is done by selecting the required location from the dropdown-list of the Location-column in the fixed-motes tab;
- *Remove existing fixed motes*: this requires the selection of a row;
- *Activate fixed motes*: the location-tracker sends an activation-message to each fixed mote in the fixed-motes tab. Fixed motes that receive this message blink a led as indication of this event. If required the “Activate fixed motes”-button can be pressed again if certain fixed motes do not receive the message;
- *Modify entries of fixed motes*: this comes down to modifying the MAC-address or location of an “fixed mote”-entry. Note that changing the location of an fixed mote is supported by making a selection from a dropdown-list in the location-column;
- *Create routing topology*: this sends a starting-signal for creating a routing-topology to the fixed motes that are in sending-range of the computer/laptop running the location-tracker. This option also disables access to all buttons in the fixed-motes tab;

- *See the current state of the routing-topology*: there is a text field that displays the current state of the routing-topology. More about the process of creating a routing topology can be found in Chapter 7;
- *See the current state of an individual fixed mote*: the column “Status” indicates the status for each fixed mote that is known to the location-tracker.

Tab: Roaming motes

The roaming-motes tab is shown in Figure 10 below.

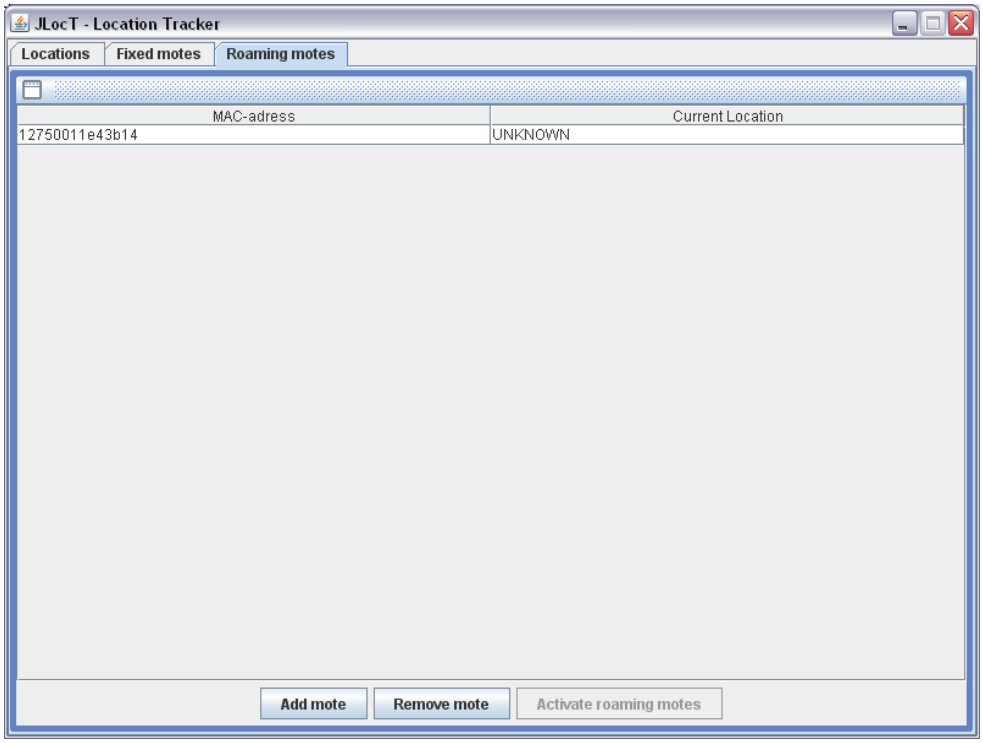


Figure 10: roaming-motes tab of Location Tracker

The roaming-motes tab allows the user to manage the motes that persons wear. These are the motes that need to be tracked in the building. In the roaming-motes tab the user is allowed to:

- *Add new roaming motes*: this requires entering an unique MAC-address. Keep in mind that tracking an individual roaming mote requires running an extra thread for it. When not considering the computational-overhead limit of running parallel threads, the current implementation of JLocT has a theoretical limit of 2148 roaming motes. These can be concurrently tracked. For more on that theoretical limit see Section 8.2;
- *Remove existing roaming motes*: this requires the selection of a row;
- *Activate roaming motes*: the location-tracker sends an activation-message to each roaming mote entered in the roaming-motes tab. Roaming motes that receive this message blink all led’s as an indication of this event. If required the “Activate roaming

motes”-button is still enabled for the event that certain roaming motes do not receive the message the first time. Activating a roaming mote comes down to sending an instance of the “ProgramMessage”-class. For the roaming motes such an instance also has a data-member called rmID. This identifier for the roaming mote basically is the index that the roaming-mote has in the table of the roaming-motes tab. Note also that the “Activate roaming motes”-button is only available after the creation of a routing-topology. Otherwise the button is disabled;
- *Modify entries of roaming motes*: this comes down to changing the MAC-address of an roaming mote. Manually changing the location of a roaming mote is disabled in this tab. Updating the location of the roaming motes is a job for the corresponding RMLocationClassifier-thread.

8.1.2 The mote-application

When the mote-application is first deployed on an mote, it blinks a led for five seconds and turns it off. Then the mote-application waits until it receives an instance of the “ProgramMessage”-class, whose data-member “receiver” equals the motes MAC-address. When this occurs the mote “knows” that it is either an fixed mote or an roaming mote. Sending such an instance of the “ProgramMessage”-class is done by “activation” from the fixed-motes tab or roaming-motes tab in the location-tracker application. Once the mote takes on the role of a fixed mote or a roaming mote different things can happen.

Role: Roaming mote

If a mote becomes a roaming mote it first blinks all its leds for five seconds to indicate that this event is happening. After this a RoamingMote-thread enters into a loop and periodically does the following things in sequence:

- Broadcast a locationRequest-message, thereby requesting nearby fixed motes to keep track of the signal-strength that is associated with the broadcast;
- Check to see if a new sequence-number should be used. In the current implementation this is the case after sending five messages with the same sequence-number. When this is the case the sequence-number is updated. The sequence-number can only take values in the following interval [0, 99]. So after using the sequenceNumber 99 the next sequenceNumber is 0 again;
- Wait a specified time before resending a locationRequest-message. Starting from the moment a locationRequest-message is sent, the time before resending such a message should not be less than one second in the current implementation.

Note that the a locationRequest-message also has a value that is sent. To prevent the Message-class from becoming unnecessarily big and problems that come with this (i.e. increased probability of packet-loss during transmission), the decision is made to have only a single data-

member that can store a general-purpose value. For the locationRequest-message that value actually is computed as follows:

(roamingmote-ID * 1000) + currentSequenceNumber

The roamingmote-ID is an value that is assigned to the mote when it is first activated. It actually is piggy-bagged in the “ProgramMessage”-object that the mote received from the location-tracker. The roamingmote-ID and the currentSequenceNumber are stored in one data-member of the Message-class. To do this the roamingmote-ID is first multiplied with the value 1000. The reason for choosing the latter value is because currentSequenceNumber can only take on values in the range [0,99]. That means that when currentSequenceNumber is two digits long, there is still one zero (from right to left) between it and the roamingmote-ID. Of course the fixed motes are aware of this and extract the roamingmote-ID and currentSequenceNumber from the data-member value. This is done by inverting the above computation.

Role: Fixed mote

The mote-application in the role of a fixed mote actually is a multi-threaded application. When a mote becomes a fixed mote it first blinks a single led for five seconds, to indicate that this event is happening. After that it starts a MessageReceiver-thread whose main responsibility is to make received messages available to other classes.

Normally in “Object Oriented Design” a class should only have a single responsibility to change as is mentioned in [34]. However since threads and memory to store objects don’t come cheap on Sentilla motes, the decision is made to deviate from this at times. That is why the MessageReceiver-thread is also responsible for doing the processing of received locationRequest-messages. This also leads to the fixed mote being more responsive, because the thread that receives such types of messages also directly processes them when they are received.

After the MessageReceiver-thread is created the fixed mote executes the initialization-code for the flooder-algorithm. The latter algorithm is responsible for creating a routing-topology. During the execution of this part of the code the fixed mote go’s into a loop, and processes all incoming messages as if they are meant for the flooder-algorithm. After the flooder-algorithm has stopped the fixed mote knows who its parent-node and its child-nodes are. More on the flooder-algorithm can be found in Chapter 7.

Now the fixed mote starts a thread for the multi-hop protocol and let all led’s burn for 10 seconds. If all went well the fixed mote is now fully initialized and ready to support the communication-infrastructure. At this point the fixed mote is running the MessageReceiver-thread and the MultihopProtocol-thread. The latter is constantly checking to see if the MessageReceiver-thread has received any messages. If so the MultihopProtocol-thread parses them and takes appropriate action.

8.2 Tracking roaming motes

Once a routing-topology is created and location-information is deployed, location-tracking can begin. By now the “Activate roaming-motes”-button in the “Roaming motes”-tab of the location-tracker has become available. Pressing this button while roaming motes are within sending-range of the location-tracker, activates roaming motes and thus starts the location-tracking process.

A roaming mote periodically broadcasts locationRequest-messages that are received by fixed motes. Each locationRequest-message has a sequence-number associated with it. The roaming mote only uses a new sequence-number after sending 5 locationRequest-messages. Until then the fixed motes store the RSSI-values. After receiving a new sequence-number the fixed motes compute a mean value that they send in a locationUpdated-message to the location-tracker.

The parameters for the locationUpdated-message are in reality stored in the data-member “value” of the Message-class. This is done in the folowing way:

(ID_OF_ROAMINGMOTE * 1000000) + (sequenceNumber * 1000) + RSSIvalue

To explain where the magic numbers 1000000 and 1000 come from we must first consider the following:

- The data-member “value” of the Message-class is an integer;
- The maximum value to store in an integer is 2147483647;
- Carefully observing RSSI-values in a test-setting, indicates that no value requiring three digits (or more) is observed when using the Sentilla motes.

This means that the RSSI-value takes no more than two digits as a representation in the “value”-parameter of the Message-class. Because sequence-numbers are always in the range [0, 99], no more than two digits are reserved for its representation. To separate the 3 values a 0 is placed behind each value. This means that 6 of the digits starting from the left of the maximum value are reserved. Meaning that JLocT in its current implementation can differentiate between 2147 + 1 = 2148 roaming motes. The addition of 1 is because the roaming mote with roamingMoteID = 0 is also supported.

Doing location-tracking on 2148 roaming motes also requires an equal amount of threads to be started. After all for each roaming mote a RMLocationClassifier-thread needs to be started. This can have a major impact in terms of system resources.

Since the limit of 2148 roaming motes is not easily exceeded in a normal home / building, an integer instead of a long is used to represent the “value” data-member in the Message-class.

This smaller data-type might even be considered as an optimization since less bytes per message need to be sent.

8.3 Reliable Messaging

Doing reliable messaging in JLocT comes down to some class having a method that keeps resending messages. This is done until the MessageReceiver-thread receives an acknowledgement of receipt from the receiving mote. The latter is a instance of the Message-class having the data-member “messageType” set to some constant indicating “ACKRECEIPT”.

As is mentioned in Chapter 7 the Sentilla motes do not seem to be able to guarantee reliable synchronous communication.

According to the “Sentilla Programmer’s Guide” [33] the garbage collector is allowed to drop messages and interrupts when it stops the virtual machine to clean up allocated memory. Therefore the problem of lacking “reliable” synchronous communication is attributed to the garbage-collector.

In an attempt to cope with this problem “reliable messaging” is introduced. Doing reliable messaging in the process of updating a roaming mote’s location is not considered useful. Components involved with reliable messaging might be more busy with guaranteeing an update than to be open for new incoming updates that must be relayed.

But reliable messaging is considered useful in the process of creating a routing topology and deployment of location-information to fixed motes, since communication-failure here might result in total systems-failure.

That is why the MultihopProtocol-class, used by fixed motes for multihop-communication, uses reliable messaging. This class does not use reliable messaging when supporting location-updates of roaming motes. But for doing deployment of location-information here to fixed motes reliable messaging is an asset.

The Flooder-class used in the flooder-algorithm also supports reliable messaging for the creation of a routing-topology. However there is one situation in the flooder-algorithm, during creation of a routing-topology, where reliable messaging is not used.

That is the case when a fixed mote also initiates sending startFlooding-messages. Other fixed motes indicate on receival of such a message that the sending fixed mote is a parent-node. This is done by sending a parentFlooding-message. However the fixed mote sending startFlooding-messages does not know in advance how many neighboring fixed motes there are. Also the time needed to send a message to all neighboring motes is unknown. Some messages may even be dropped.

Therefore the fixed motes send startFlooding-messages only for a finite amount of times. This also means that in theory it is possible that some neighboring fixed motes do not receive a startFlooding-message. In practice this problem does not appear to occur during the experiments. The number of resend-attempts apparently is sufficiently high (5 times).

For more information on experiments done for the creation of a routing-topology see Section 9.6.

8.4 Experimental variation: roaming mote classifies

In JLocT the location-tracker is ultimately responsible for doing classification, and thus for tracking where a roaming mote resides at some point in time. As an experiment an alternative setup is also explored. In Figure 11 below you can see a sequence-diagram of this alternative setup. It shows a simple situation with two fixed motes, one roaming mote and a location-tracker.

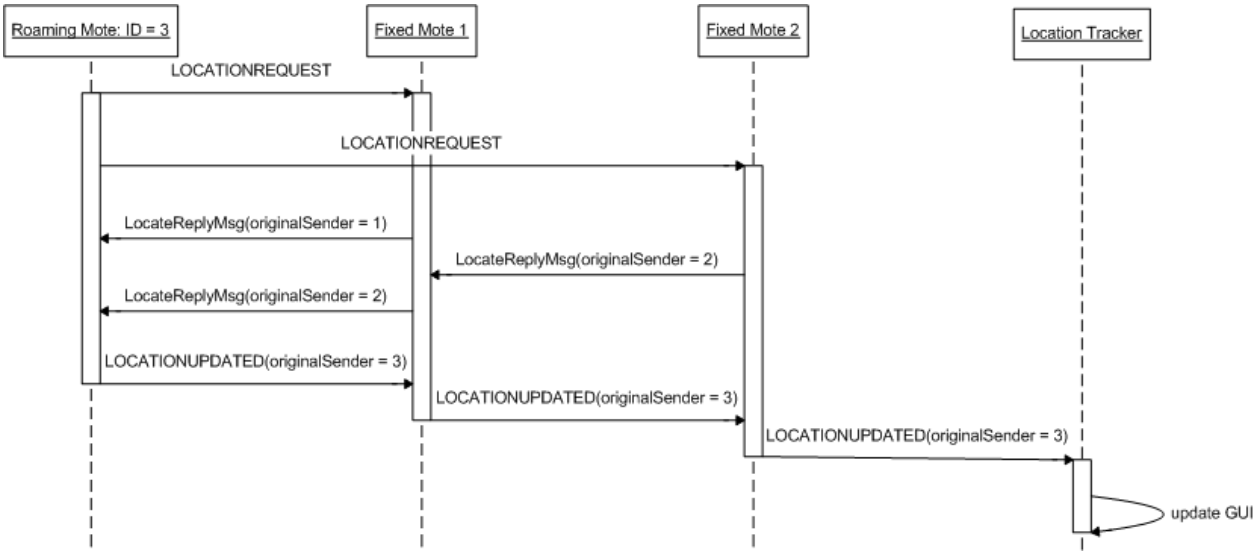


Figure 11: sequence-diagram of the situation where roaming mote classifies

In this experiment the roaming mote again broadcasts locationRequest-messages, which is indicated with the lines going to both fixed motes. However now the fixed motes send instances of a locateReplyMsg-object to the roaming-mote as an reply to the broadcast.

The roaming mote uses these received locateReplyMsg-objects to periodically compute the location. When the roaming mote computes the location it sends a locationUpdated-message to the location-tracker by using the multihop-facilities of the network. This locationUpdated-message also contains the location-index that the location-tracker can use to find the location in the locations-tab.

Note that in this setting the roaming mote has a separate thread running for receiving locateReplyMsg-objects. In this experiment no reliable messaging is used in the process of

updating the location of a roaming mote. This is again done to keep the system more responsive and prevent location-updates with stale data.

To find out more about this experiment and the corresponding results the interested reader is referred to Section 9.5.

9 Experiments and testing

During this project several experiments are done using JLocT. Also experiments are done with a experimental variation of the architecture of JLocT. In those experiments the roaming mote does classification instead of the location-tracker. The experiments can also be subdivided into fixed positions-testing in the test-rooms, as well as those where the roaming mote is constantly moving.

To make sure that JLocT is not making decisions on stale data sometimes the starting-room is also varied. This means that in one test the starting-point is in room A, while in the other test the starting-point is in room B.

Next to experiments where the effectiveness of the system is being tested, also tests are done where the efficiency is being tested. This comes down to analyzing the time-intervals between consecutive updates in the location-tracker. Actually this is also done for the experimental variation of the architecture, where the roaming mote does classification. Analyzing the time-intervals between this experimental architecture and that of JLocT makes comparison in terms of efficiency possible.

Finally the creation of a routing-topology is not always guaranteed. That is why this process is also being tested. The results of all these experiments and tests can be found in the upcoming sections. When looking at classification-results we are calling “insufficient classification-performance” those situations where the classification-accuracy is less than 60%.

9.1 Testing-environment and hardware used

All experiments are done in two test-rooms of the Bernoulliborg-building of the RuG. Now some detailed information is supplied about the rooms so that the results presented in other sections can be duplicated. Rooms A (5161.0564) and B (5161.0566) of the experiments basically have the same dimensions. A map of these two rooms is shown in Figure 12 below:

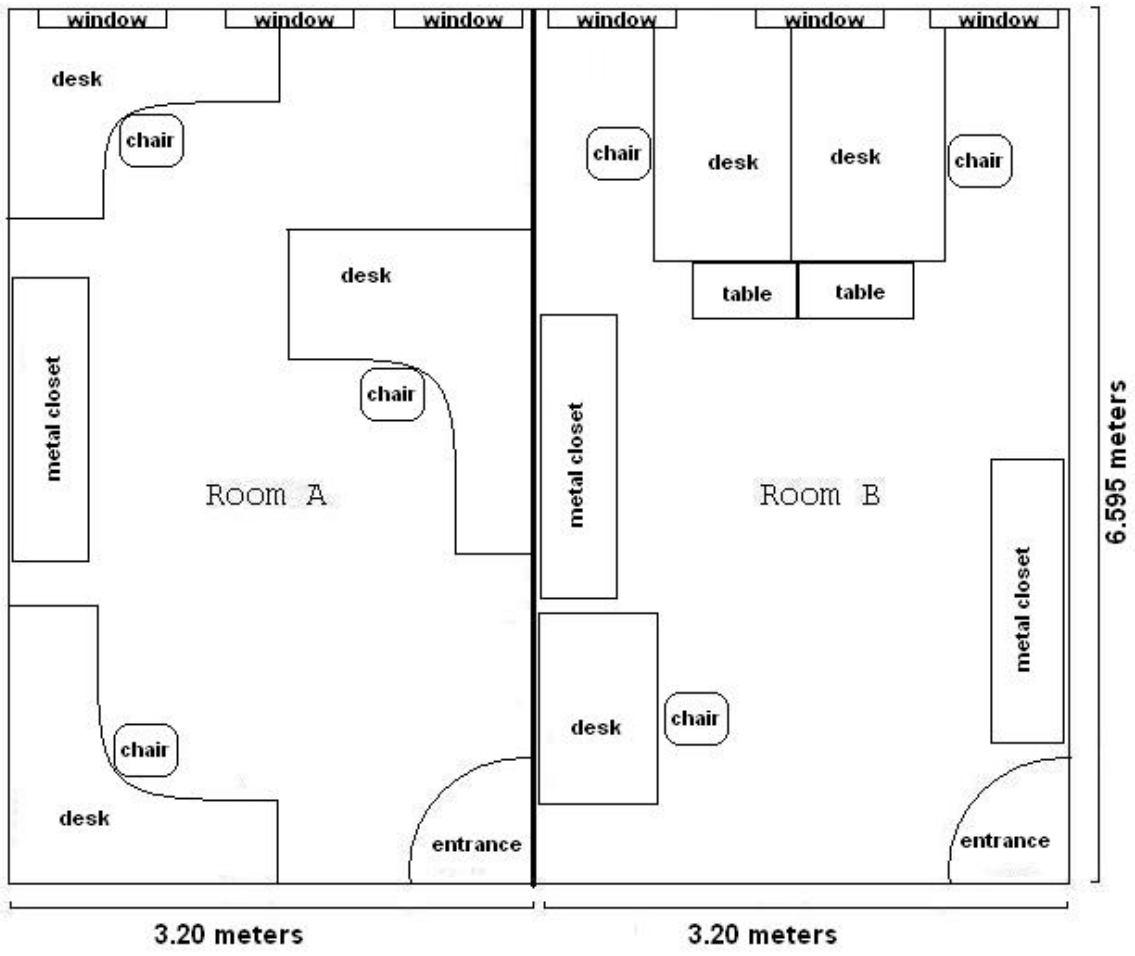


Figure 12: map of the test-rooms A and B

As can be seen from the map room A and B are both 3.20 meters wide and 6.595 meters long. The distance between floor and ceiling of both rooms is 3 meters and 17 cm. An exception on this is the area starting from the entrance until 3.865 meters in the room. Here a cardboard ceiling is hanging at an altitude of approximately 2 meters and 75 cm from the ground. This cardboard ceiling is present at the same location in both rooms. Thus it reduces the actual height of both rooms in those areas with 42 cm.

The composition of the walls, including the wall separating room A and room B, basically is a metal stud wall. Metal stud profiles are formed steel profiles. These profiles have a C- or U-shape and are about 70 mm thick. As an insulator probably mineral wool is used. According to drawings double plasterboards are applied on both sides of the walls.

To get an idea of places where the problem of multipath-interference in the rooms might occur, please see the test-results of the next section. During testing the location-tracker application is running on a laptop with the following specifications:

TravelMate 4501LCi

- Intel Pentium M 715 processor (1.5 GHz, 400 MHz FSB, 2 MB L2 cache)
- 40 GB HDD
- 512 MB DDR memory.
- 802.11b/g wireless LAN

An image of the laptop can be seen in Figure 13 below.

Figure 13: laptop running the location-tracker software during testing

WSN's can be used for observing some phenomenon in the physical world. They consist of nodes called motes. Each mote is a programmable unit that is equipped with sensors, means for wireless communication and batteries. The WSN used during testing is comprised of motes called Sentilla JCreates. Each Sentilla JCreate is equipped with a Texas Instruments 16 bit MSP430 microcontroller [35] and a CC2420 low-power wireless radio [33].

One of the products created by the Sentilla Corporation [36] is the Pervasive Computing Kit (PERK) [2]. This kit comes with two JCreates (the motes), a JCreate Programming Fixture, Sentilla's IDE, a USB gateway, batteries, full documentation and sample applications. The PERK can be seen in Figure 14 below.

Figure 14: The Pervasive Computing Kit (PERK) [2]

Programming of the motes can by default be done using the Java-programming language. For this project the PERK and a total of five of these motes (including the two coming with the PERK) are used.

As is mentioned in Section 1.2 the original idea was to create a hybrid WSN. This hybrid network would consist of both Sentilla JCreate's and SunSPOTs [3].

According to [3] the processor-board of a SunSPOT has the following characteristics:

- 180 MHz 32 bit ARM920T core - 512K RAM/4M Flash
- 2.4 GHz IEEE 802.15.4 radio with integrated antenna
- 3.7V rechargeable 720 mAh lithium-ion battery
- USB interface
- 32 uA deep sleep mode

A SunSPOT Java development kit is used during the attempts of creating a hybrid network. Such a development kit contains 2 Sun SPOT devices and a base station. Also included are development tools, tutorials, sample code and accessories [3]. Figure 15 below shows the SunSPOT Java development kit.

Figure 15: SunSPOT Java development kit [3]

Attempts to create such a hybrid network have ultimately failed. For more information on these attempts see appendix A.

9.2 Testing roaming mote on fixed positions

In the MERRIT article [20] experiments are done to compute the accuracy of a system, by doing measurements on fixed locations. In a similar way computations to determine the accuracy are done at fixed positions in the test-rooms. For this type of experiment an architectural variation is used where the roaming mote does classification instead of the location-tracker. More on the efficiency of this experimental architectural variation can be found in Section 8.4.

The results of this fixed-positions experiment give an indication of positions in the test-rooms, where the system is or is not sufficiently able to cope with multipath-interference. In Figure 16 below you can see the twenty fixed positions in both test-rooms where classifications are done.

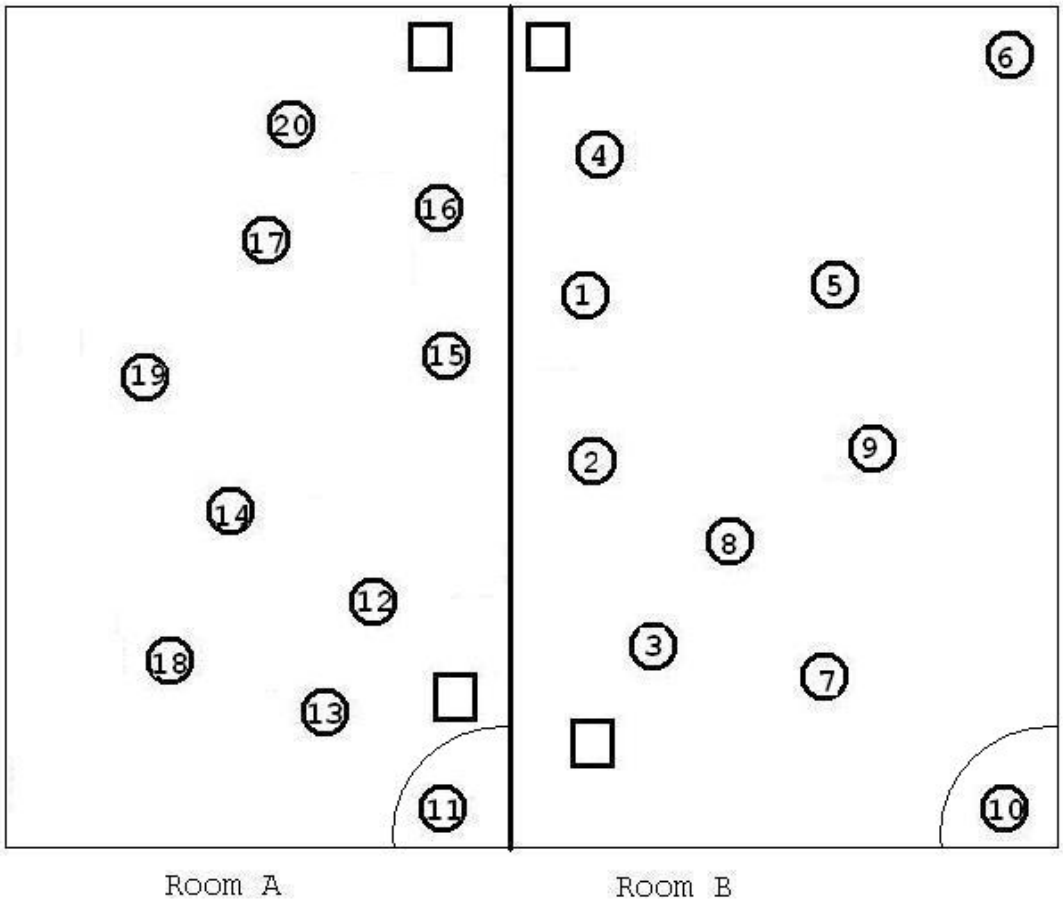


Figure 16: fixed positions in the two testing-rooms where classifications are done

Some locations of these twenty positions are chosen to be closely to the wall separating the two rooms. This is done to see how the system copes with the “neighboring room problem”, where radio-signals sent in one room might also be received by the other room. This might (negatively) influence the classification-results.

Except for keeping in mind to have locations close to the walls, the position of furniture is also thought of. Other than these two criteria the other locations are chosen randomly. In appendix B you can find a table giving the success-rates for each position per room. In Figure 17 below you can see the success-rates for each of the fixed positions. The locations marked red are those whose performance is marked “insufficient”.

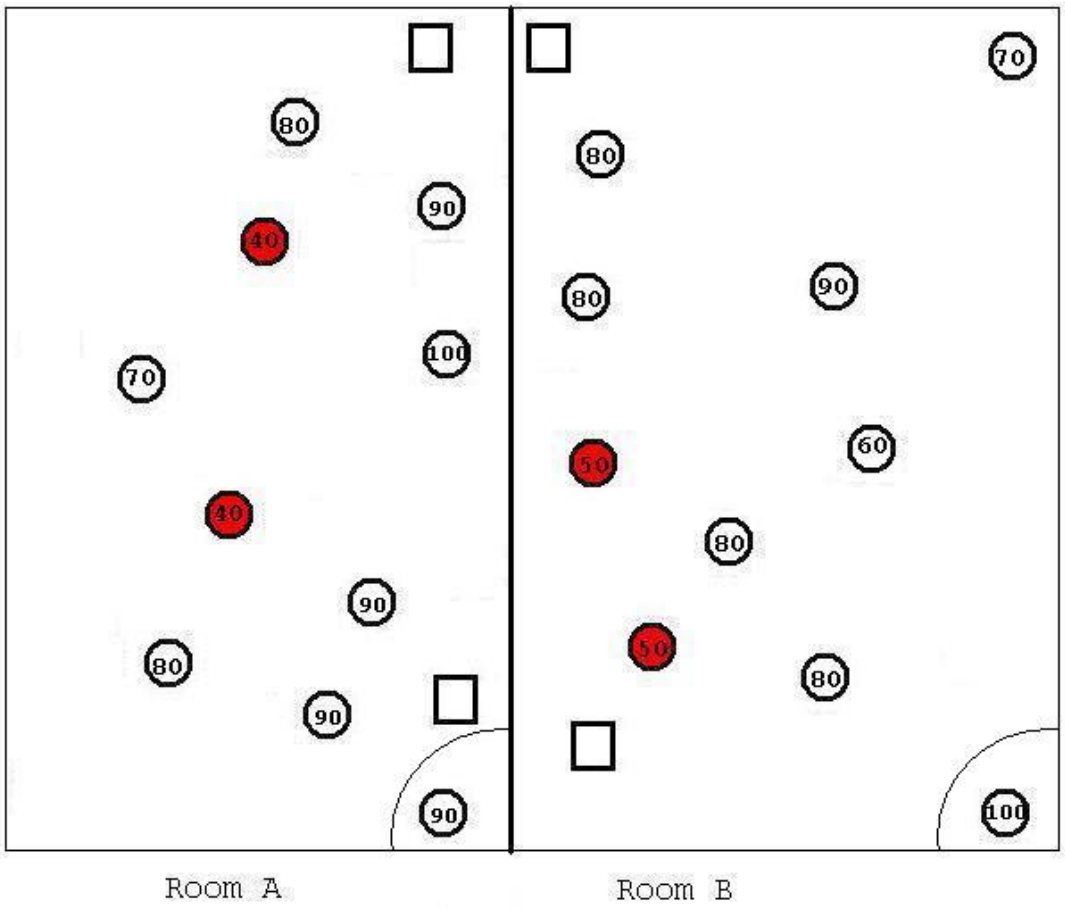


Figure 17: Success-rates for each fixed position in the two testing-rooms where classifications are done

The average success-rate for location-tracking in room A and room B are respectively 77% and 74%, while the success-rate over both rooms is 75% on average. Although this might seem promising there still are the red positions in Figure 17 shown above. These indicate positions at which classification-performance on average is less than 60%. The presence of these “bad positions” means that at those positions the current settings for the concepts of “temporal diversity” and “spatial diversity” are not sufficient to cope with the problem of multipath-interference.

There is one other possible reason for the classification-performance being less than 60% at positions 2, 3, 14 and 17. At the time of this experiment a large metal closet is standing in room B before position 2. In fact it is standing against the wall separating room A from room B. It might be conceivable that this large metal closet is responsible for creating slight attenuations in the radio-signals that are received in both rooms. These attenuations might have a negative impact on the classification-results. However no fixed-position testing are done without the closet.

Instead moving-tests are done both with and without the metal closet in room B. More about the moving-tests can be found in the Section 9.3.

9.3 Testing while moving the roaming mote

Next to experiments on fixed positions in the test-rooms, also the so-called “moving-tests” are done. In these tests a roaming mote stays for approximately 4 minutes in one room, and than go’s to the other room and stay there for 4 minutes. While being in a room the roaming mote is randomly being moved throughout the room. The idea is that this simulates a person walking with a roaming mote. Data is collected by the location-tracker that can later be used to compute a success-rate and failure-rate for the test-configuration for each room.

At each point in time the roaming mote is at an altitude of approximately 160 cm from the ground. It would also be always in direct line of sight with the fixed motes of the room it is in at that moment.

The position and number of fixed motes per room varies per experiment. Sometimes fixed motes are positioned on a window-opening or on cardboard tubes. These tube are 87 cm long having a diameter of 7.3 cm. There are even experiments done where the tubes are put on tables to increase the elevation even further.

Basically the experiments where the roaming mote is constantly moved can be divided into those having one fixed mote per room and those having two fixed motes per room. However in the latter case the tests performed can be divided even further. In Section 9.2 the possible problem of a metal closet interfering with the test-results is introduced. Both “moving-tests” are performed where the closet is in room B and tests where it is removed. The impact of removing such a metal closet on classification-results is the topic of Section 9.4.3. Here we only consider test-situations where the metal-closet is not present in room B.

9.3.1 Single fixed mote per room

In this type of testing each testing-room gets one fixed mote assigned to it. In the center of each testing-room a fixed mote is placed on top of a cardboard tube. The success-rates of these tests are shown in Figure 18 below:

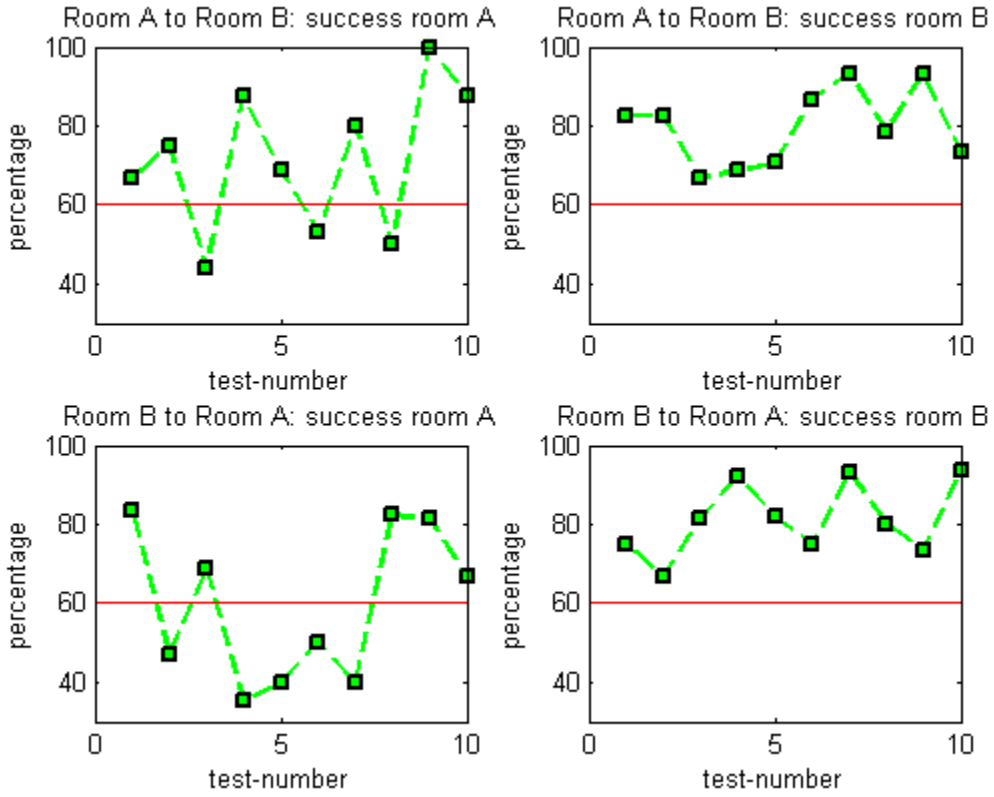


Figure 18: success-rates for having a single fixed mote per room

Note that all tests in room B in this situation give a success-rate above the 60% boundary. However as you can see room A can still at times give classification-performance less than 60%. Thus we call the performance of this configuration as “insufficient”. This seems to be independent of whether the test starts in room A or in room B. For an overview of the success-rates shown above see appendix C. In the next section we see if we can do better with two fixed motes per room.

9.3.2 Two fixed motes per room

Having more than one fixed mote per room provides a bigger “spatial diversity”. This should in theory lead to better classification-results. Here we have a look at the tests where there are two fixed motes in each room. Since the problem of “multipath interference” depends on the characteristics of the test-environment, it might be conceivable that the position of the fixed motes in each room also impacts the classification-performance. That is why some experiments with different positions for fixed motes per room are also being done. Below we discuss this category of different experiments. As mentioned before cardboard tubes used are 87 cm long and have a diameter of 7,3 cm.

Experiment: NE, SE – NW, SW

In this experiment we have the following setup:

- a fixed mote in room A is placed in a window-opening of the North-East corner. This window-opening is elevated 92 cm above the ground;
- a fixed mote in room A is placed on top of a cardboard tube near the South-East corner;
- a fixed mote in room B is placed in a window-opening of the North-West corner. This window-opening is elevated 96 cm above the ground;
- a fixed mote in room B is placed on top of a cardboard tube near the South-West corner.

The locations of the fixed motes in this experiment are the same as is done with the fixed locations-experiment described in Section 9.2. In fact you can see the locations of the fixed motes in Figures 16 and 17. The squares in that figure denote the actual locations of the fixed motes. The success-rates of these tests are shown in Figure 19 below:

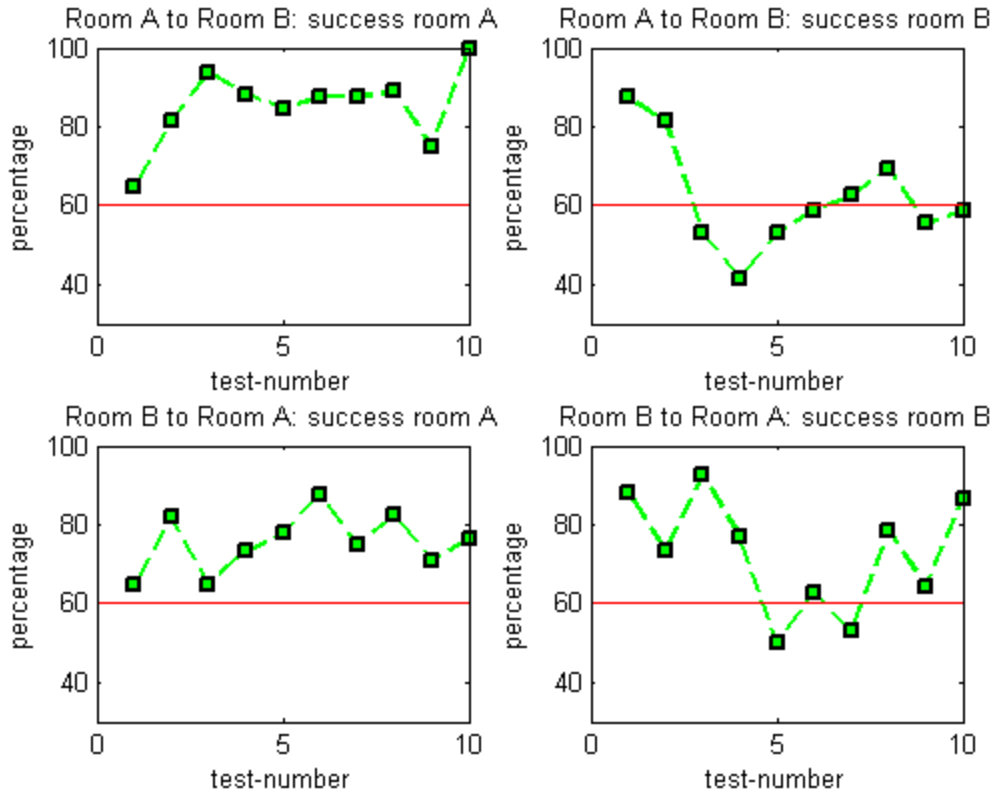


Figure 19: success-rates for experiment: NE, SE – NW, SW

As you can see all tests in room A performed “sufficient” indicating success-rates above the 60% boundary. This is again independent of whether the starting-point is in room A or in room B. So

for room A this test-configuration performs better than having a single fixed mote in the center of both test-rooms.

However tests in room B performs at times below the 60% boundary. This also seems to be independent of the starting-room for the test. Thus we call the performance of this configuration also “insufficient”. For a overview of the success-rates shown above see appendix D.

Experiment: NE, SW – NE, SW

In this experiment we have the following setup:

- a fixed mote in room A is placed in a window-opening of the North-East corner. This window-opening is elevated 92 cm above the ground;
- a fixed mote in room A is placed on top of a cardboard tube, standing on top of a table near the South-West corner. The table is elevated 75 cm above the ground. Adding this to the length of the cardboard tube of 87 cm, makes sure that the fixed mote is elevated 162 cm above the ground;
- a fixed mote in room B is placed in a window-opening of the North-East corner. This window-opening is elevated 111 cm above the ground;
- a fixed mote in room B is placed on top of a cardboard tube, standing on top of a table near the South-West corner. The table is elevated 73 cm above the ground. Adding this to the length of the cardboard tube of 87 cm, makes sure that the fixed mote is elevated 160 cm above the ground.

The success-rates of these tests are shown in Figure 20 below:

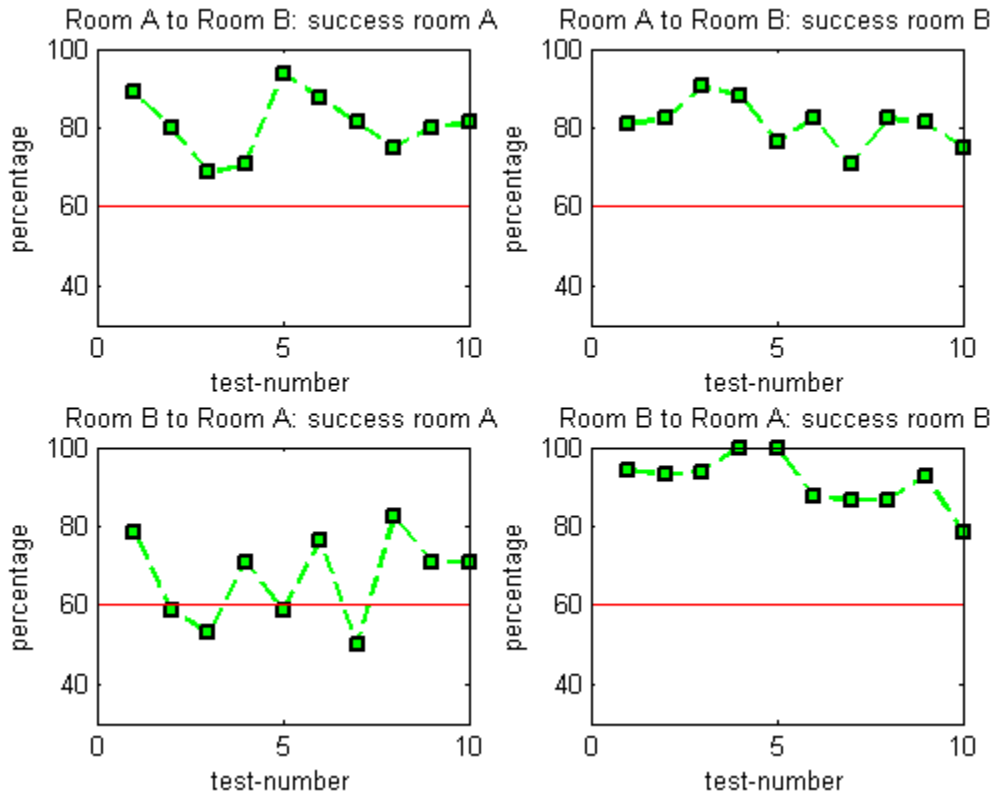


Figure 20: success-rates for experiment: NE, SW – NE, SW

Looking at the graphs from the tests of room A we can see that going from room A to room B gives “sufficient” success-rates. But going from room B to room A suddenly gives success-rates below 60%. Thus this poor performance also deems this configuration of fixed motes as “insufficient” for the testing-rooms.

Although this setup at times gives “insufficient” success-rates for room A there is a lesson to be learned here. Looking at the performance of this setup in room B we see only “sufficient” success-rates. In fact it seems that success-rates of 80% and higher in this setup are more rule than exception. This is especially the situation when looking at the scenario of going from room B to room A.

For a overview of the success-rates shown above see appendix E.

Experiment: NE, S – NE, SW

In this experiment we have the following setup:

- a fixed mote in room A is placed in a window-opening of the North-East corner. This window-opening is elevated 92 cm above the ground;

- a fixed mote in room A is placed on top of a cardboard tube, standing on top of a table against the South-wall. The table is elevated 75 cm above the ground. Adding this to the length of the cardboard tube of 87 cm, makes sure that the fixed mote is elevated 162 cm above the ground;
- a fixed mote in room B is placed in a window-opening of the North-East corner. This window-opening is elevated 111 cm above the ground;
- a fixed mote in room B is placed on top of a cardboard tube, standing on top of a table near the South-West corner. The table is elevated 73 cm above the ground. Adding this to the length of the cardboard tube of 87 cm, makes sure that the fixed mote is elevated 160 cm above the ground.

The idea for this setup came from the perception of how well having fixed motes in the corners North-East / South-West worked for the success-rates of room B. Not only is the performance "sufficient" but it seems that having 80% success-rate (or higher) is more rule than exception. Only the success-rate of room A is at times lower than 60% and thus marked as "insufficient". That is why an attempt is made to increase success-rates in room A while keeping the performance of room B maked as "sufficient". This is done by moving the fixed mote in the south-west corner to the center of the south-wall. The fixed mote is still elevated a total of 162 cm above the ground. To be clear on how this setup looks like see Figure 21 below.

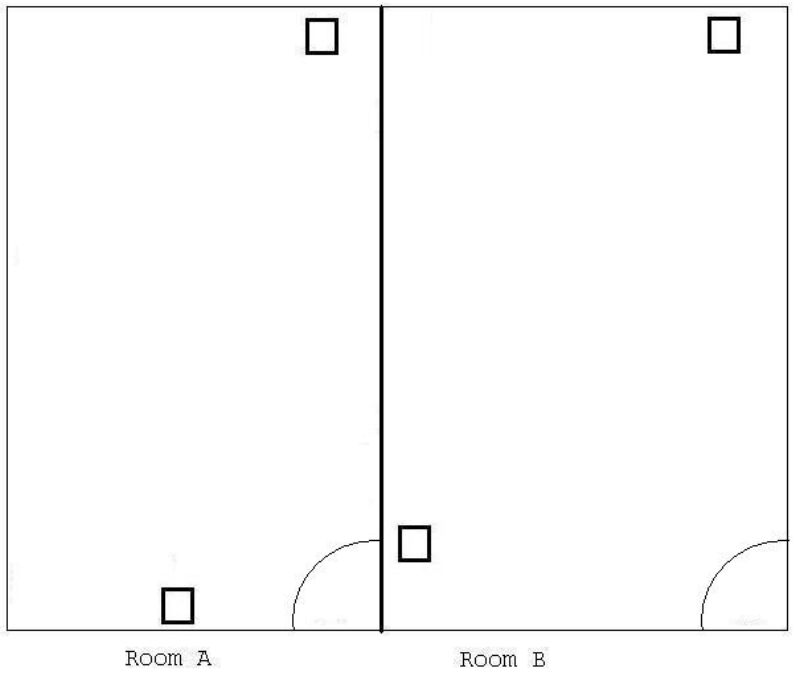


Figure 21: positions of fixed motes in experiment: NE, S – NE, SW

The success-rates of these tests are shown in Figure 22 below:

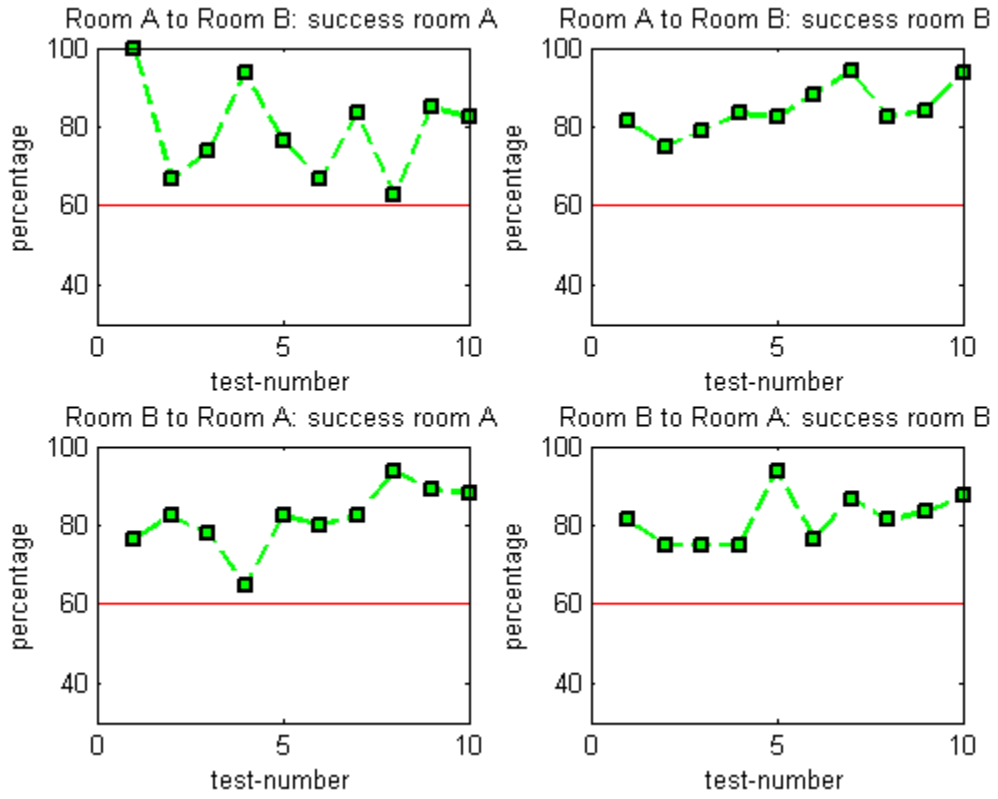


Figure 22: success-rates for experiment: NE, S – NE, SW

This configuration of the fixed motes in the test-rooms still leads room B to frequently have success-rates close to 80%. But what is more important is that you can see that the success-rates of room A and B are now both above the 60% boundary. This is the case for both starting in room A or in room B. That is why this configuration of the fixed motes in the test-rooms is called “sufficient”. It is also the reason why this is the recommended configuration of the fixed motes in the test-rooms.

For a overview of the success-rates shown above see appendix F.

9.4 Impact on classification-effectiveness

During the execution of the experiments there are some factors that can have an additional impact on classification. These factors are discussed here.

9.4.1 Distance to walls

Signals may be received stronger by receivers in a neighboring room, than by receivers in the room from which the signal originated. This problem often occurs when being close to walls separating rooms which a system should be able to distinguish. Another name for this type of problem is the “neighboring-room problem”.

JLocT does employ techniques that can mitigate the effects of this problem. These techniques are called “temporal diversity” and “spatial diversity” and come from the MERRIT-article [20]. These concepts and the MERRIT-article are also talked about in Section 4.1.4 and Chapter 5.

The location of the fixed motes in the test-rooms also has a large impact on the extend that this problem impacts classification. During each test a person walks with a roaming mote randomly in a room. When doing this the roaming mote is also at times put close to the wall separating room A and room B. This means that the experiments do say something about which setup of fixed motes can cope better with this problem.

Unfortunately no guarantees can be given that the neighboring-room problem does not occur.

9.4.2 Low battery life

Another factor that can have an impact on classification-performance is battery life of the motes. If the batteries in a mote are almost empty that mote’s signal-strength, when sending signals, also decreases.

There is one good example during testing of how a low battery life can impact signal-strength and thus decrease classification-effectiveness. This occurred when doing testing in the situation of experiment “NE, S – NE, SW”. The setup of that experiment is considered to be the recommended setup for the test-rooms.

However at some point during a test the battery life of the fixed mote, standing against the south-wall in room A, is below 40%. The result is a success-rate for room A of 27%. In contrast the success-rate for room B is 92%. When the batteries of the fixed mote against the south-wall in room A are replaced, newer tests never seem to come below the 60% boundary.

So it appears that the signal-strengths of the fixed motes in room A just could not match those in room B if battery life of a single fixed mote is low. As a side-note the results of this “bad-run” are omitted from the results discussed in Section 9.3.2. The reason for this is that having low battery life creates a form of bias that should be omitted as much as possible in the testing-results.

When considering the tests: if battery life of a mote is 40% or lower they should be replaced. This is kind of a rule of thumb that is based on observations in signal-strength and battery life during testing.

9.4.3 Metal closet in room B

At some point in time a metal closet is standing against a wall in room B. This wall separates room A from room B. During testing it is noted that when a roaming mote comes close to this closet the fixed motes in room A indicate higher signal-strength values.

In fact at times the fixed motes in room A indicated higher signal-strength values than those in room B where the roaming mote actually is. That is why this metal closet potentially can create a form of bias in the test-results. This is also the reason why the closet is at some point in time removed from room B.

When considering the discussed experiments in Section 9.3 no metal closet is present in room B. This is due to the possibility of bias in the test-results. However one type of experiment is done with and without the metal closet in room B. We now offset the results of having a metal closet in room B with the corresponding results of experiments in Section 9.3. For details on the locations of the fixed motes in this experiment the interested reader is referred to the corresponding experiment in Section 9.3.

Experiment: NE, SE – NW, SW

In Figure 23 below you can see the success-rates of this experiment with and without the metal closet in room B.

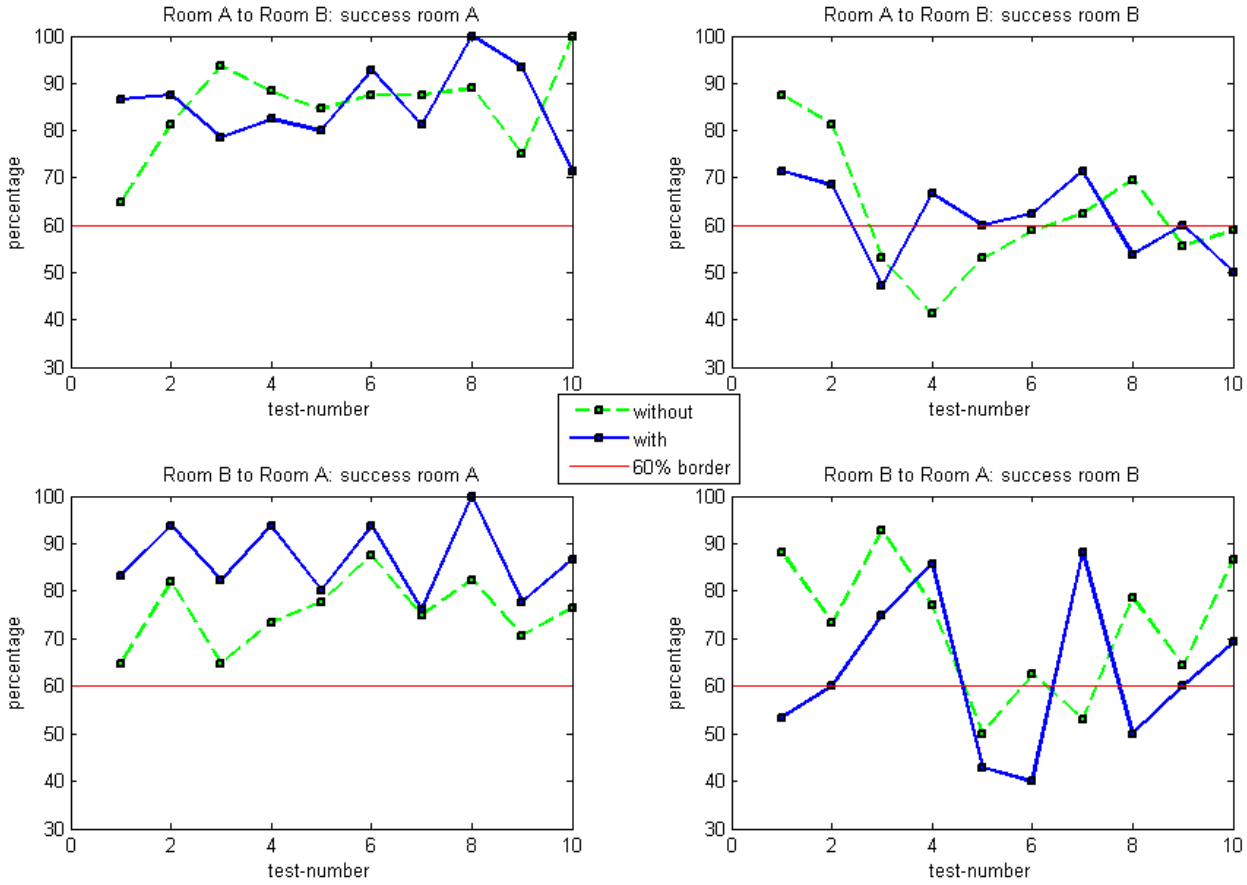


Figure 23: success-rates for experiment "NE, SE – NW, SW" with and without metal-closet in room B

When looking at the graphs of these tests our intuition might tell us removing the metal-closet from room B has no effect on the probability off successful classification. A side-note might be made when looking at the success-rates of room A for the experiment of “going from room B to room A”. We might believe in that situation that the metal-closet in room B gives an increased probability of successful classification when being in room A. To be more confident about the effects of the metal-closet in room B on classification-results we perform statistical hypothesis-testing and compute a P-value for the test [37].

First we make the observation that the classification-data in the experiments is binomial, since outcomes can be either “correct” or “incorrect”. The total number of successes in the data follows a binomial distribution [37]. The goal is to test whether the probability *p* of successful classification is different for both rooms when the metal-closet in room B is removed.

We assume that going from room A to room B has the same kind of effect on success-probability of classification as going from room B to room A. Therefore only data from the situation “from

room A to room B" is used here. Four different types of samples of classification-data are used here. To be exact:

- 150 samples of room A before the metal closet is moved;
- 150 samples of room A after the metal closet is moved;
- 150 samples of room B before the metal closet is moved;
- 150 samples of room B after the metal closet is moved.

For the test on room A we compute an estimate of the success-probability p_0. This is done based on the 150 samples of room A before the closet is removed. In a similar way for testing room B the success-probability p_0 is estimated based on the 150 samples of room B before the closet is removed. The number of samples *n* is 150 in both tests.

To determine whether we should use a large-sample test the equation in Figure 24 below must hold. The figure below shows Equation 6.3.1 from the book [37].

$$0 < np_0 - 3\sqrt{np_0(1-p_0)} < np_0 + 3\sqrt{np_0(1-p_0)} < n$$

Figure 24: equation for determining whether large-sample or small-sample binomial test should be done from the book [37]

However we have to take into account that p_0 in our situation is not known. It is estimated by dividing the number of successes by the total number of samples *n*. Also the observed p_1 is estimated by dividing the number of successes *k* by the number of samples *n*. If we compute the variance of the difference between p_0 and p_1 we now get:

$$\text{variance}\left(\hat{P}_0 - \hat{P}_1\right) = \text{variance}\left(\hat{P}_0\right) + \text{variance}\left(\hat{P}_1\right)$$

$$= \frac{P(1-P)}{n} + \frac{P(1-P)}{n}$$

$$= \frac{2P(1-P)}{n}$$

Figure 25: variance-formula for difference between two estimators of binomial parameter P

The observed multiplication by 2 here gives rise to a new variant of the equation:

$$0 < np_0 - 3\sqrt{np_0\,2(1-p_0)} < np_0 + 3\sqrt{np_0\,2(1-p_0)} < n$$

Figure 26: adapted equation for determining whether large-sample or small-sample binomial test should be done

This gives us the following for the two tests:

Room A

n = 150
number of successes *k* before removing closet = 128
$p_0 = k / n = 0.8533$

$$0 < 150 * 0.8533 - 3 * \sqrt{150 * 2 * 0.8533 * (1 - 0.8533)} < 150 * 0.8533 + 3 * \sqrt{150 * 2 * 0.8533 * (1 - 0.8533)} < 150$$

=
0 < 110 < 146 < 150
=
TRUE

Room B

n = 150
number of successes *k* before removing closet = 91
$p_0 = k / n = 0{,}6067$

$$0 < 150 * 0.6067 - 3 * \sqrt{150 * 2 * 0.6067 * (1 - 0.6067)} < 150 * 0.6067 + 3 * \sqrt{150 * 2 * 0.6067 * (1 - 0.6067)} < 150$$

=
0 < 66 < 116 < 150
=
TRUE

Of course some rounding off is done here, but the equation still holds if this is not done. So for testing both room A and room B in the situation “from room A to room B” we can use a large-sample test based on a approximate Z ratio. A note must be placed that since the presence of the closet in room B can either increase or decrease the success-probability for a room, we perform the following two-sided test for the binomial success-parameter *p*:

$$H_0 : P = P_0$$
$$\text{versus}$$
$$H_1 : P \neq P_0$$

level of significance: $\alpha = 0.05$

This means that H_0 is rejected if *z* is either

$$\leq -z_{\alpha/2} \text{ or } \geq z_{\alpha/2}$$

The border-values for the critical region can be found using a math-program or looking at a table showing cumulative areas under the standard normal distribution such as in [37]. In this situation it is found that H_0 should be rejected if z is either <= -1.96 or >= 1.96.

The formula from [37] for computing a z-ratio is as follows:

$$z = \frac{k - np_0}{\sqrt{np_0 (1 - p_0)}}$$

Figure 27: formula for computing a z-ratio in a binomial hypothesis-test when p_0 is known

The formula here assumes that p_0 is known. However in our situation it is estimated by taking the average of 150 samples. So again we need to modify this formula so that it keeps this in mind.

The value where the square-root is computed off, in the denominator, needs to be multiplied by 2 in order to fix this problem. This is for the same reason as the multiplication by 2 in the equation of Figure 26. The adapted version of the test-statistic in Figure 27 now becomes:

$$z = \frac{k - np_0}{\sqrt{n\,2\,p_0 (1 - p_0)}}$$

Figure 28: adapted formula for computing a z-ratio in a binomial hypothesis-test

For room A after the metal closet in room B is removed this gives:
n = 150
number of successes *k* after removing closet = 126
p_0 = 0,8533

$$z = \frac{126 - 150 * 0.8533}{\sqrt{150 * 2 * 0.8533 * (1 - 0.8533)}} = -0.3255$$

For room B after the metal closet in room B is removed this gives:
n = 150
number of successes *k* after removing closet = 95
p_0 = 0,6067

$$z = \frac{95 - 150 * 0.6067}{\sqrt{150 * 2 * 0.6067 * (1 - 0.6067)}} = 0.4722$$

So the computed z-ratio's for the rooms, after the closet is removed, is not less than or equal to -1.96. Also these computed z-ratio's are not greater than or equal to 1.96. Thus we conclude at the $\alpha = 0.05$ level of significance, that removing the metal closet from room B does not have an impact on the probability of successful classification in both rooms.

To be more confident about the results of the hypothesis-tests for both rooms we now compute a P-value for each test. Again this is done by either using a math-program or looking at a table showing cumulative areas under the standard normal distribution such as in [37].

For room A we use the computed test-statistic -0.3255. This gives the P-value:

P-value = P(Z <= -0.3255) + P(Z >= 0.3255)
= 0.3707 + 0.3707
= 0.7414

This means that for room A for any α < 0.7414 the conclusion is, that the metal closet in room B does not impact the success-probability of classification in room A. In other words: assuming that the null-hypothesis is true, the probability of seeing values as extreme or more extreme as is actually observed is 74.14%.

For room B we use the computed test-statistic 0.4722. This gives the P-value:

P-value = P(Z <= -0.4722) + P(Z >= 0.4722)
= 0.3192 + 0.3192
= 0.6384

This means that for room B for any α < 0.6384 the conclusion is that the metal closet in room B does not impact the success-probability of classification in room B. In other words: assuming that the null-hypothesis is true, the probability of seeing values as extreme or more extreme as is actually observed is 63.84%.

One final remark must be made about the results of these hypothesis-tests. The results of the hypothesis-tests indicate that the presence of the metal-closet in room B does not impact the probability of successful classification in either rooms. However it must be noted that during testing a person is more or less constantly moving the roaming mote when being in a room. The results of the fixed-position experiments in Section 9.2 indicate that positions two and three show “insufficient performance”. This indicates a classification-performance below 60%. These two positions are very near the metal-closet. That is why it might be possible that the metal-closet can still interfere with classification-success, when a person just stands near it for a period of time.

9.5 Efficiency of letting the roaming mote do classification

One responsibility of the location-tracker in JLocT is to determine where a roaming mote is at a point in time. In Section 8.4 an alternative experimental setup is described. There the roaming mote runs the classification-algorithm. The outcome is then sent, by means of the multihop-network, to the location-tracker. As an experiment for this alternative setup, 180 consecutive samples are taken from the classification-data for the situation where the roaming mote does

classification. In Figure 29 below you can see a histogram of the time-intervals between the 180 consecutive location-updates, from the situation where the roaming mote does classification.

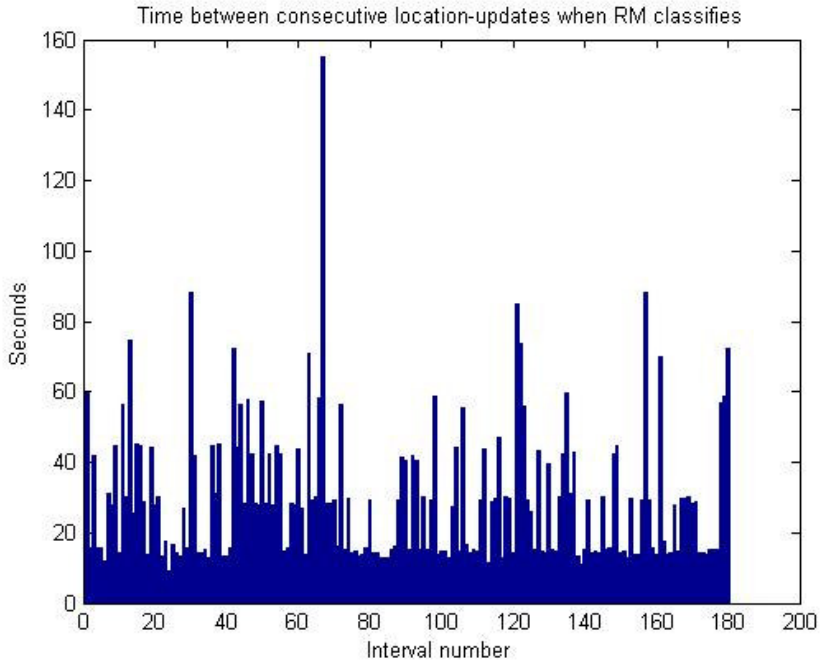


Figure 29: for 180 samples the time between consecutive location-updates when RM classifies

Preferably the histogram of Figure 29 should show a distribution that is as uniform as possible. A perfect uniform distribution means that it is possible to guarantee location-updates at certain time-intervals. However as you can see this is not the case. The histogram shows periodically peaks in the time it takes to update the location of a roaming mote. There is even a peak where it took 155 seconds before the location of a roaming mote could be updated. This is very inefficient. The mean waiting time here is 29.7 seconds.

We now offset these results with the way how JLocT does classification. That means taking 180 samples from the test-data where the location-tracker does classification. In Figure 30 below you can see a histogram, of the time-intervals between the 180 consecutive location-updates, from the situation where the location-tracker does classification.

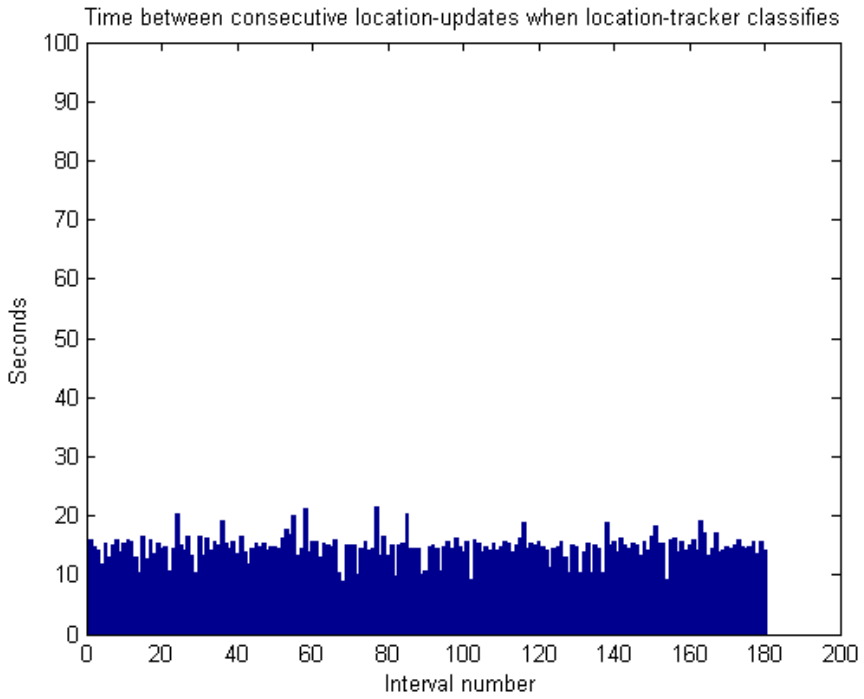


Figure 30: for 180 samples the time between consecutive location-updates when location-tracker classifies

As you can see the histogram is much more uniform than in the case where the roaming mote does classification. A note must be made that there are still outliers. The mean waiting time here is 14.7 seconds. The longest time between consecutive updates that is recorded here is 22 seconds. Of course this is much better than the maximum waiting time of 155 seconds in the situation where the roaming mote does classification.

The cost for this enhanced efficiency comes in the form of reduced scalability of JLocT. Because now for each roaming mote that must be concurrently tracked a thread has to be started on the location-tracker.

A final remark must be made that the experiment where the roaming mote does classification is conducted with the following settings:

- Time between resending locationRequestMsg-messages is 2 seconds;
- Waiting time before recomputing the location is 10 seconds.

In contrast the settings of the experiment where the location-tracker performs classification are:

- Time between resending locationRequestMsg-messages is 1 seconds;
- A new sequence-number is used after the roaming mote (attempts to) sends 5 messages. After which fixed motes send a locationUpdated-message to the location-tracker by means of the multihop-network;

- Every 7.5 seconds the location-tracker re-computes the location of a roaming mote.

Any peaks in the time required to update the location of a roaming mote might be attributed to the garbage-collector of a Sentilla-mote. When a message goes from roaming mote through the multihop-network to the location-tracker a computation-delay is happening at each hop. Basically this delay can vary per fixed mote due to the garbage-collector starting to run.

To strength this suspicion a small test is done. In this test a mote is configured to send a message to a test-application every 2 seconds. The laptop running the test-application should just display the time taken between receiving messages. This means that it is expected that the resulting graph of such an test should be somewhat uniform. The actual graph is shown in Figure 31 below.

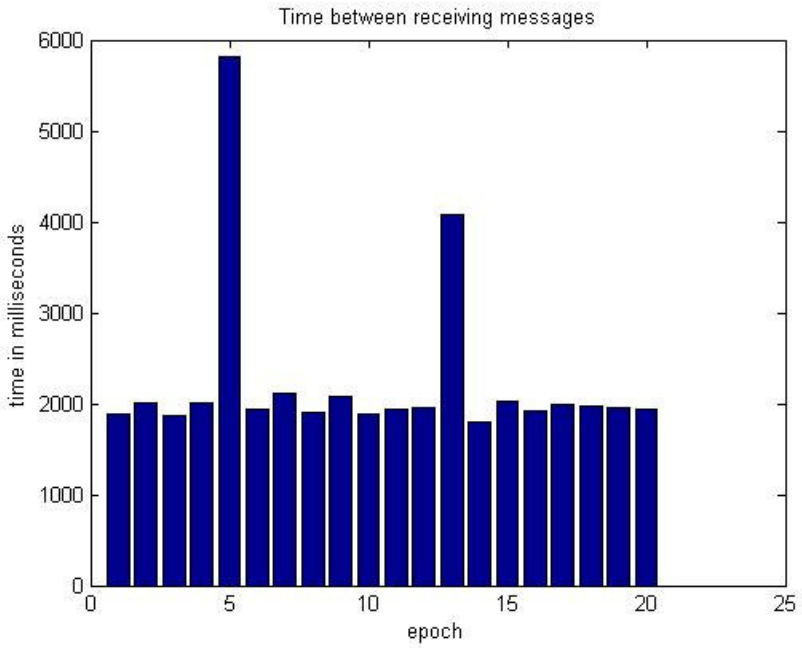


Figure 31: time between receiving messages

The resulting graph shows that even for a small sample-size of 20 messages being received outliers can occur. The outliers can clearly be seen in the 5th and 13th epoch. This observation strengthens the suspicion that periodic sending of messages on Sentilla-motes running the accompanied Java virtual machine cannot be guaranteed.

9.6 Routing-topology tests

As is stated in Chapter 7 the process of creating a routing-topology in JLocT sometimes is not fully completed. In JLocT this process also includes distributing location-information to fixed motes. This sub-process might also fail to initialize. To be able to say something about JLocT's

ability of creating a routing-topology binomial data is gathered. This data indicates whether the whole process is “successful” or “not successful”. Note that creation of a routing-topology here includes convergence of a flooder-algorithm and successful distribution of location-information to fixed motes.

In Figure 32 below you can see a graph, indicating the success-percentages computed over all 180 samples that are collected.

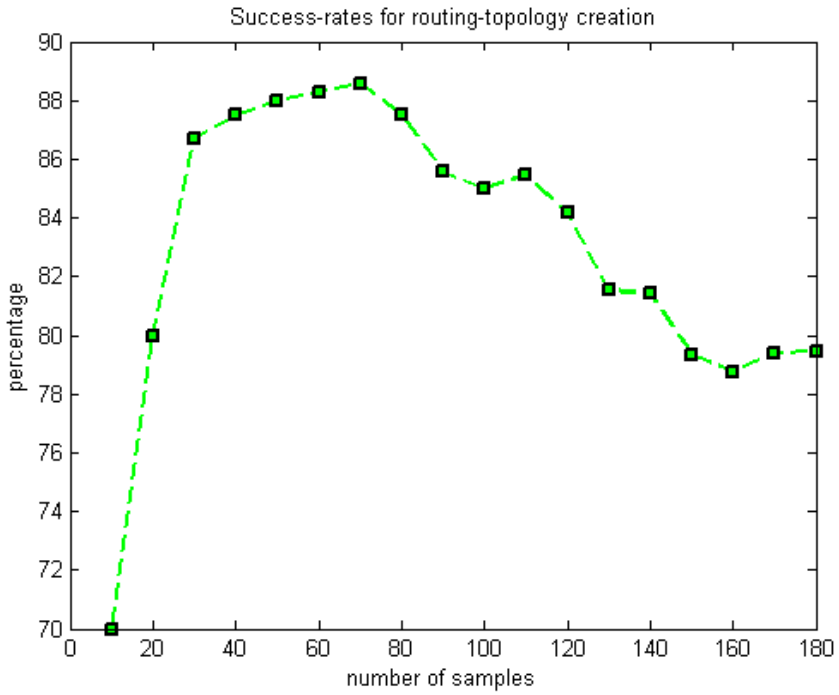


Figure 32: success-rates for creation of a routing-topology using all samples

When looking at this graph we see something strange happening. We see an initial increase of the success-rate until somewhere near the 80th sample. From there on the success-rate gradually goes down until 79% at the 180th sample to be exact. As might be suspected after the 85th sample something is changed during the setup of routing-topology creation.

Until then the location-tracker is minimized and the network-monitor window in the Sentilla-environment is displayed. This is done to monitor what is going on in the network. Strangely enough this approach appears to have had a positive impact on the total convergence of the flooder-algorithm and the distribution of the location-information to fixed motes.

After the 85th sample is collected the setup at the laptop running the location-tracker is changed. The network-monitor is not displayed in the Sentilla-environment. Instead the GUI of the location-tracker is shown. Intuitively by looking at the graph we would suspect that this behavior resulted in the decrease of the success-rate for creating a routing-topology. To be more confident about this, it is decided to do a two-sided binomial hypothesis-test at level of

significance: $\alpha = 0.05$. Since the data is following a binomial distribution we test whether the success-probability *p* has changed with the new setup at the laptop running the location-tracker. For this test $n = 80$ samples are taken from both situations.

The binomial data is represented as either a 1 (success) or a 0 (failure). In the situation where the network-monitor in the Sentilla-environment is shown and the GUI of the location-monitor is minimized there are 70 successes. From now on the technique used in that situation is denoted as method A. The technique in the other situation where the network-monitor is not shown in the Sentilla-environment, and the GUI of the location-monitor is shown is denoted as method B. The number of successes in method B is denoted as $k = 55$.

The initial probability p_0 is estimated by taking the average of the first 80 samples. This gives a success-rate of $p_0 = 70 / 80 = 0.875$ for Method A. The estimated success-rate for Method B comes down to $p = 55 / 80 = 0.6875$.

To determine whether a large-sample test or a small-sample test should be used, we again turn to the adapted equation shown by Figure 26 in Section 9.4.3. Filling in the values of $p_0 = 0.875$ and $n = 80$ we get the following:

$$0 < np_0 - 3\sqrt{np_0\,2\,(1-p_0)} < np_0 + 3\sqrt{np_0\,2\,(1-p_0)} < n$$

=

$$0 < 80 * 0.875 - 3 * \sqrt{80 * 0.875 * 2 * (1 - 0.875)} < 80 * 0.875 + 3 * \sqrt{80 * 0.875 * 2 * (1 - 0.875)} < 80$$

=

$0 < 57.4501 < 82.5499 < 80$

=

FALSE

Because the equation does not hold, a small-sample test is considered to be more appropriate. The reason lies in the fact that not enough samples are collected to sufficiently let the data approximate a normal-distribution for a large-sample test. Since we have 80 samples for method B and a value for p_0 we can now perform the small-sample test. We begin by outputting the probability of successes, using the binomial pdf (probability density function) and p_0, for each success-value *k* in the range [0,80]. For more information about the binomial pdf the interested user is referred to any statistics book like [37].

The actual outputted binomial pdf for this small-sample test can be found in appendix G. Now that we have a mapping of the success-probabilities for each value *k* in the range [0,80] we can create a critical region. We want to perform a two-sided binomial hypothesis-test at the level of significance: $\alpha = 0.05$. Specifically we want to test:

$$H_0 : P = P_0$$

versus

$$H_1 : P \neq P_0$$

Because it is a two-sided test the critical region is split into two areas. The first area consists of all values k, starting at $k = 0$, for which the total accumulated probability is ≤ 0.025. The second area consists of all values of k, starting at $k = 80$, for which the total accumulated probability is ≤ 0.025. Using this approach the following two-sided critical region is found:

$k \leq 63$ and $k \geq 76$

The number of success k in the data of method B equals 55. This is less than 63, meaning it is in the critical region. Thus we **reject** the null-hypothesis at level of significance: $\alpha = 0.05$. This basically means that when creating a routing-topology, a user should minimize the GUI of the location-monitor and display the network-monitor in the Sentilla-environment. Strangely enough this appears to give the highest probability of success when creating a routing-topology.

Since the alpha of this test may not be exactly 0.05 we compute it:

$$\begin{aligned} P(\ k \in \text{CriticalRegion}\ \mid H_0 \text{ is true}) &= P(k \leq 63 \mid p = 0.875) + P(k \geq 76 \mid p = 0.875) \\ &= 0.0190 + 0.0224 \\ &= 0.0414 \end{aligned}$$

So if we assume H_0 actually is correct, the probability of incorrect rejection is even smaller for this test than $\alpha = 0.05$. This gives much confidence to the results of this test, since we still rejected the null-hypothesis with this smaller alpha. To be even more confident about the results of this test we compute a P-value for this test.

$$\begin{aligned} \text{P-value} &= 2 * P(k \leq 55) \\ &= 2 * 0.0000 \\ &= 0 \end{aligned}$$

For precision four digits are used here. Since a two-sided test is performed, the probability is multiplied here by 2. We can say something about the test. If we assume that H_0 is in fact true the probability of observing a value of 55 or more extreme is 0% (at four digits precision). Thus we can be very confident about the results of this test. It can therefore be concluded that to maximize the success-probability of routing-topology-creation, the GUI of the location-tracker should be minimized. Also the network-monitor in the Sentilla-environment should be displayed. This approach is most likely to lead to successful creation of a routing-topology.

9.7 Results and evaluation

In this chapter several types of experiments and tests are discussed. We have discussed tests where the roaming mote is on fixed positions in both test-rooms. This type of testing used a configuration where for room A, a fixed mote is placed in the north-east and south-east corner. In contrast room B had a fixed mote placed in the north-west and south-west corner. Doing testing with a roaming mote on fixed positions in this setup gives an average success-rate of 75% over both rooms. This might seem promising but four positions during this experiment give a success-rate of less than 60%. This type of classification-performance is therefore deemed “insufficient” for the purposes of the SM4ALL-project.

Another type of experiment are the tests where the roaming mote is being moved while JLocT tries to determine in which room it is. In these so-called “moving tests” a person stays for approximately 4 minutes in one room followed by 4 minutes in the other room. While being in a room the person walks around randomly. Compared to the tests where the roaming mote does not move, here multiple scenario’s for positioning the fixed motes in both test-rooms are explored.

In the first explored scenario each room has a single fixed mote positioned in the center of the room. For room B this type of testing gives success-rates above the 60% boundary. Unfortunately this is not the case for room A, where several success-rates are below 60%. Multiple scenario’s are explored for two fixed motes per room while doing “moving tests”.

In one scenario room A has a fixed mote positioned in the north-east corner and one in the south-east corner. In this same scenario room B has a fixed mote positioned in the north-west and the south-west corner. All tests in room A are deemed "sufficient" indicating success-rates above the 60% boundary. However tests in room B at times give success-rates below the 60% boundary. The classification-performance is therefore also marked as being "insufficient".

Another setup of the fixed motes is the case where, for room A, a fixed mote is placed in the north-east corner and one in the south-west corner. The same setup holds for room B. Note that here the fixed motes in the south-west corners are elevated even further. This is done by placing the cardboard tubes on top of a table in both rooms. The results for this experiment show that classification in room B in this setup gives good success-rates. All success-rates are above 60%. In fact it seems that success-rates of 80% and higher are more rule than exception. The results in room A in this setup are still showing success-rates below 60%. That is why this setup is also deemed "insufficient".

The final scenario for the “moving tests” is based upon a lesson learned in the previously described setup. In this previously described setup a fixed mote in both rooms is placed in the north-east corner and one in the south-west corner. It is noticed that room B suddenly performed much better than in the first described scenario’s of the “moving-tests”. This gives rise to a scenario where the setup of room B is not changed. Meaning one fixed mote in the north-east

corner and one fixed mote in the south-west corner. For room A however a fixed mote is positioned in the north-east corner and one against the center of the south-wall. This latter positioned fixed mote is still positioned on the same table, as is done in the previously described test. The results of this new test show that success-rates in room B are still frequently close to 80 %. But what is more important is that now the success-rates of both room A and room B are above the 60% boundary. This means that this researched setup of the fixed motes in both rooms, is the only one that has “sufficient” classification-performance in the test-rooms.

Another type of experiment that is discussed has to do with the efficiency of how JLocT does classification. Experimental code is created where instead of the location-tracker the roaming mote does classification. In that experimental setup the location-tracker only receives location-updates from the roaming mote through the multihop-network. It turns out that this experimental setup systematically gives peaks in the time needed to update the location of a roaming mote. The highest peak in time recorded is 155 seconds before the location of the roaming mote could be updated. In contrast the setup where the location-tracker is responsible for classification systematically gives far shorter waiting-times. The longest recorded peak in time with this setup is 22 seconds. This means that it is much more efficient to let the location-tracker do classification than an individual roaming mote.

Since the creation of a routing-topology could not be guaranteed within JLocT, data is collected to see what the success-rate is. However during testing it is noticed that the performance of routing-topology-creation varies, depending on what happens at the computer running the location-tracker. Creating a routing-topology seems to go best if the GUI of the location-tracker is minimized and the network-monitor of the Sentilla-environment is displayed. The probability of successfully creating a routing-topology in this situation is estimated (by averaging 80 samples) to be 87.5%. If the GUI of the location-tracker is displayed and the network-monitor is not, the probability of success is estimated (by averaging another 80 samples) to be 68.75%. To be sure that this difference is statistically significant a two-sided small-sample binomial hypothesis-test is performed at approximately $\alpha = 0.05$ (alpha = 0.0414 to be exact). The results of this test indeed strengthens the suspicion that the probability of successfully creating a routing-topology is higher when the GUI of the location-tracker is minimized. Also the network-monitor in the Sentilla-environment should be shown. A computation for a P-value of this test gives 0% at four digits precision. If we assume that the setup at the location-tracker is not important the probability of seeing the observed results is 0% (at four digits precision). Thus we conclude that the setup at the location-tracker is in fact important. Specifically to maximize the probability of successful creation of a routing-topology the GUI of the location-tracker should be minimized. Also the network-monitor should be shown in the Sentilla-environment.

During testing it is noted that some factors have an influence on the probability of successful classification. The distance of a roaming mote to the wall separating the test-rooms for example seems to be able to influence outcomes. This “neighboring-room problem” seems to impact classification-performance more in some location-configurations of the fixed motes than in other.

Therefore it is noted that the location of the fixed motes in the test-rooms also has a large impact on the extend that this problem impacts classification.

Another factor that has the potential to influence the outcomes of classification is the battery life of the motes. It is noted that if the battery life of a single mote becomes less than 40%, than the batteries should be replaced. If this is not done correct classification might be jeopardized. This is especially the case in a situation where few fixed motes represent a single room.

Room B has at some point in time a large metal closet in it. This large metal closet is standing against the wall separating room A from room B. During testing it is noted that if a person is standing in a certain angle from the closet for a period of time, than JLocT gives consecutive faulty classifications. It is suspected that the flat metal sides of the closet amplified radio-signals going to room A. Thus classifications when being in room B are suspected to suffer in this situation. However two two-sided large-sample binomial hypothesis-tests of 300 samples (150 samples with and without the closet), one for each room, indicated that this is probably not the case for the “moving-tests”. Computations of P-value’s for these tests give 0.7414 and 0.6384 for room A and B respectively. This means that it can be assumed that the metal closet does not impact the success-probability of classification. Under this assumption the probability of seeing the observed number of successful classifications or more are 74.14% and 63.84% for room A and room B respectively.

It seems that classification in the moving-tests does not suffer from the presence of the metal closet in room B. But this does not rule out that a metal closet might still impact classification-performance. This might be the case in the event a person is just standing in room B in a certain angle from the closet for a period of time. This suspicion is strengthened by a observation in the “fixed position”-tests. It can be seen from those results that two positions in room B give classification-performance below 60%. These positions just happen to be near the metal-closet.

10 Conclusions and Future work

One of the goals of the project is to do location-tracking with a WSN. Radio-based localization-techniques are preferred for this project since radio-communication is by default present in WSN's. The JLocT-system uses signal-strength of radio-signals to determine the position of mobile roaming motes. This type of feature of radio-signals is chosen because it is easy to obtain, requires no specialized hardware and does not require extra network-traffic for time-synchronization.

JLocT can track roaming motes on the granularity of rooms. The technique that it uses for this is based on that of the MERRIT-system [20]. JLocT and MERRIT allow fast deployment as both do not require a site-survey. Also both JLocT and MERRIT have implementations of concepts like "temporal diversity" and "spatial diversity". This allows for mitigating effects of transient blocking of radio-signals and multipath-interference.

While doing testing other factors are also observed that may have an impact on classification-performance. For example the distance of a roaming mote to the wall separating two rooms might raise the so-called "neighboring-room problem". This happens if a person walks very close to a wall separating two rooms. In such a case emitted signals may be received stronger by receivers in a neighboring room, than by receivers in the room from which the signal originated. A combination of "temporal diversity" and "spatial diversity" might be able to mitigate these effects. However until now test-results have shown that the neighboring-room problem might still occur despite these techniques.

Battery life of the (fixed-)motes is also a factor that may have an impact on classification-performance. During testing it is observed that when battery life drops below 40%, classification-performance might suffer. The hypothesis is that if for example a fixed mote in room A has a battery life lower than 40% it may not receive a signal originating from a roaming mote. And if a signal is received the corresponding signal-strength value may not be very strong. Thus leading JLocT to find it more likely that the roaming mote is in the other room. Therefore when battery life becomes lower than 40% batteries should be replaced.

Yet another factor that may impact classification-performance is the presence of a metal closet in one of the testing-rooms. However this factor has a bit controversy associated with it. When looking at the results of the fixed-position tests, there are 2 positions close to the closet that give results below 60%. Thus performance at those locations is marked as "insufficient". This gives rise to the hypothesis that the metal closet might impact classification-performance. However a large-sample two-sided binomial hypothesis-test seems to reject this hypothesis. It must be mentioned that the hypothesis-test is performed on data collected during moving-tests. In these tests a person is more or less constantly moving randomly in the room. So the statistical hypothesis-test seems to reject the hypothesis that the metal closet impacts classification-performance. But it might still be possible if the user is standing in certain positions close to the metal closet.

From the experiments it is observed that positions and number of fixed motes in rooms also appear to have a big influence on the success-rate of classification. From the performed experiments the best results are obtained with two fixed motes per room. These are placed in the setup “NE, S – NE, SW”. This means that room A has one fixed mote in the north-east corner and one placed against the south-wall. Furthermore room B also has one fixed mote placed in the north-east corner and one in the south-west corner. The results gained with this setup give at least a success-rate of 60%. Most of the time however the success-rate is around 80%.

Better classification-performance might be possible when using more than two fixed motes to represent a single room. The MERRIT-article [20] claims to be able to achieve a classification-performance of up to 98.9% with 4 infrastructure nodes per room. Of course they also make use of another technique equipping each infrastructure node with an aluminum reflector for radio-signals. Nevertheless using more than 2 infrastructure nodes per room might allow JLocT to give higher classification-performance as well.

During the development of JLocT some problems are observed with the Java virtual machine (JVM) and its garbage-collector used by the Sentilla motes. For example guaranteed consecutive sending and receiving of messages on a time-interval does not seem to be possible. This observation is raised by the results of a small test where one mote sends 20 consecutive messages to a receiving mote. The test shows variation in the form of 2 peaks in the intervals needed to send 20 consecutive messages.

This problem means that time-intervals between location-updates within JLocT may also vary. In fact this problem might attribute to the outcome of another experiment. In that experiment the roaming mote is responsible for doing classification instead of the location-tracker. A graphical plot of the time-intervals between consecutive location-updates shows much variation. One recorded incident even shows a waiting-time of 155 seconds, while the mean-waiting time between consecutive updates is 29.7 seconds. This is why the current architecture of JLocT lets the location-tracker do classification instead of the roaming mote. A graphical plot of the time-intervals between consecutive location-updates of JLocT shows a more or less uniform distribution. The maximum recorded waiting-time between consecutive location-updates of JLocT is 22 seconds. The mean waiting time here is 14.7 seconds. Of course using other parameters within JLocT might give rise to even better responsiveness.

There is another observed problem with the JVM and its garbage-collector used by the Sentilla motes. According to the “Sentilla Programmer’s Guide” [33] the garbage collector can lose packets and interrupts when it blocks the virtual machine. This might explain why during the development-phase it is observed that sometimes messages appear to not being sent at all. It is therefore assumed that guaranteed reliable messaging appears not to be possible for the Sentilla motes.

In an attempt to cope with this problem JLocT is equipped with its own implementation of reliable messaging. This implementation repeatedly sends messages until an acknowledgement of receipt is received.

However at times messages still appear not to being sent at all. This in turn leads to problems with the process of creating a routing-topology. A routing-topology is created using a flooding-algorithm that relies on guaranteed synchronous communication [32] for convergence.

Since reliable communication is assumed not to be possible, the flooding-algorithm is adapted. Specifically the new flooder-algorithm tries to create a routing-topology with as little communication as possible. On the other hand it does rely on the implementation of reliable messaging. So much network-traffic might still be a reality in the case of motes that are not instantly responding when a message is received.

Another modification to the original flooding-algorithm is the concept of “centralizing the decision to terminate flooding”. In the original flooding-algorithm discussed in Chapter 7, each node knows who its direct neighbors are. This information allows a node in the original flooding-algorithm to terminate flooding. However in JLocT the location-tracker keeps track of the state of individual fixed motes. Because of this JLocT lets the location-tracker decide when all motes should stop the flooder-algorithm.

As is stated before the new flooding-algorithm cannot be guaranteed to converge due to problems with the garbage-collector running on the motes. To be able to say something about the effectiveness of this new flooding-algorithm data is collected. This data is in the form of a total count on successful and non-successful runs.

A total of 180 samples gives rise to an estimated success-rate of 79%. The 180 samples are collected by means of two different methods. In the first method the GUI at the location-tracker is minimized and the network-monitor shown. The network-monitor is not shown in the second method. Also the second method does show the GUI of the location-tracker. A plot of the success-rates and the number of samples used in the computation suggests that the first method gives better classification-performance.

The results of a small-sample two-sided binomial hypothesis test seem to strengthen this suspicion. The estimated success-rate of the first method comes down to 87.5%. The second method has an estimated success-rate of 68.75%. Therefore it is concluded that the best way to create a routing-topology is to minimize the GUI at the location-tracker and to display the network-monitor in the Sentilla-environment. Although doubtful it is of course possible that a large-sample two-sided binomial test might say that the difference between the methods is not statistically significant. But this of course would require more samples to be obtained by using both methods of creating a routing-topology.

One possible improvement of JLocT might be to let the roaming motes perform motion detection. When a roaming mote “knows” when it is not being moved it can stop broadcasting messages to fixed motes. This can save a lot of battery-power. Not only for the roaming mote itself but also for any fixed motes that send location-updates to the location-tracker.

The MERRIT-system [20] used a maximum of 4 fixed nodes per room to boost “spatial diversity” and thus classification-performance. It might be interesting to see what the classification-performance of JLocT is if it uses 4 fixed nodes per room. Maybe even more fixed nodes per room can create an even further improvement of JLocT’s classification-performance.

In the MERRIT-article [20] claims are made of boosting classification-performance up to 98.9%. This is done by equipping each fixed node with a aluminum reflector for radio-signals. It might be very interesting to see if JLocT could give similar performance, when using a combination of 4 fixed nodes per room and aluminum reflectors at each fixed mote. Even with less fixed motes per room, JLocT might still benefit a lot in terms of classification-performance. Therefore equipping fixed nodes with aluminum reflectors seems to be a nice future extension of JLocT.

The current implementation of routing-topology-creation leaves room for improvement. A future version of JLocT might allow automatic detection and recovery of failing fixed motes after a routing-topology is created. Perhaps a future version of JLocT might also support spontaneous networking. Here newly discovered fixed motes are automatically assigned to the correct leaf-node in the spanning-tree. Newly discovered roaming motes might automatically be added to the system. On the other hand these optimizations do require extra communication and processing, thus additionally reducing battery life. Still these optimizations are interesting and can provide a more dynamic location-tracking system.

If JLocT really is to be used within the SM4ALL-project some sort of interface might be required. Through this interface components of the SM4ALL-platform could then communicate with JLocT. Or vice-versa. But this requires some thought of whether JLocT should actively update components about the status of roaming motes. This might be done by using SOAP (Simple Object Access Protocol)-requests to invoke web services. Alternatively components might need to periodically poll JLocT to request an update about the locations of individual roaming motes. Perhaps this would require JLocT to host a web service providing location-information about roaming motes. Either way if JLocT is to be used in combination with the SM4ALL-platform some sort of interface is required.

To conclude JLocT is a location-tracking system that, even with only two fixed motes per room, already gives “reasonable” results. Further improvements are still possible. Both for classification-performance as well as for the process of routing-topology creation. In short JLocT might grow out to be a promising location-tracking system that may be useful for the SM4ALL-project.

Bibliography

[1] http://www.ubiq.com/hypertext/weiser/UbiHome.html, accessed 24 March 2011

[2] http://www.sentilla.com/perk_faq.html, accessed 18 June 2010

[3] http://www.sunspotworld.com, accessed 7 July 2010

[4] http://www.sm4all-project.eu, accessed 23 October 2010

[5] http://www.oracle.com/technetwork/java/index.html, accessed 27 October 2010

[6] Rong Peng and Mihail L. Sichitiu. “Angle of Arrival Localization for Wireless Sensor Networks”. *In Proc. Conf. Sensor and Ad-Hoc Comm. and Networks,* 2006.

[7] Nikos Deligiannis. “Mobile Positioning based on Existing Signalling Messages in GSM Networks”.

[8] M. Kanaan, K. Pahlavan. “Algorithm for TOA-based indoor geolocation”. *IEE Electronics Letters, Vol. 40, No. 22,* October 2004.

[9] http://en.wikipedia.org/wiki/Wave_propagation_speed, accessed 10 November 2010

[10] http://www.aemc.com/techinfo/appnotes/cabletesters/APP-CableTesters-VelocityPropagation.pdf, accessed 11 November 2010

[11] http://www.generalcable.co.nz/Technical/10.4.3.1.pdf, accessed 11 November 2010

[12] http://www.cisco.com/en/US/docs/solutions/Enterprise/Mobility/wifich2.html, accessed 12 November 2010

[13] J.X. Lee, Z.W. Lin, P.S. Chin and C.L. Law. “A Scheme to Compensate Time Drift in Time Difference of Arrival Localization Among Non-Synchronized Sensor Nodes”.

[14] J. Elson, L. Girod, and D. Estrin. “Fine-Grained Network Time Synchronization using Reference Broadcasts”. *In Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Winter 2002.

[15] J. V. Greunen and J. Rabaey. “Lightweight Time Synchronization for Sensor Networks”. *Proc. 2nd ACM Int. Conf. Wireless Sensor Networks and Apps*, 2003.

[16] http://www.npwrc.usgs.gov/resource/wildlife/telemtry/timediff.htm, accessed 14 November 2010

[17] http://en.wikipedia.org/wiki/Received_signal_strength_indication, accessed 20 November 2010

[18] T.S. Rappaport. "Wireless Communications Principles and Practice". *Prentice Hall*, 1996, pp 102.

[19] P. Bahl and V. N. Padmanabham. “Radar: An in-building rf-based user location and tracking system”. *In Proceedings of the IEEE INFOCOM 2000*, 2000, pp 775–784.

[20] Y. W. Lee, E. Stuntebeck, and S. C. Miller. “MERIT: mesh of RF sensors for Indoor Tracking”. *In Proc. Conf. Sensor and Ad-Hoc Comm. and Networks*, 2006, pp 545-554.

[21] J. Hightower, G. Boriello and R. Want. "SpotON: An indoor 3D Location Sensing Technology Based on RF Signal Strength". *University of Washington, UW CSE 00-02-02*, 2000.

[22] M. Terwilliger, A. Gupta, V. Bhuse, Z. H. Kamal, and M. A. Salahuddin. “A Localization System using Wireless Network Sensors: A Comparison of Two Techniques”. 2004.

[23] http://www.tinyos.net, accessed 4 June 2010
[24] L. M. Ni, Y. Liu, Y. C. Lau, and A. P. Patil, “LANDMARC: Indoor Location Sensing Using Active RFID” *Wireless Networks, volume 10, number 6*, 2004, pp 701–710.
[25] Y. Zhao, Y. Liu, and L.M. Ni. "VIRE: Active RFID-based Localization Using Virtual Reference Elimination". *2007 International Conference on Parallel Processing (ICPP 2007)*, 2007, pp 56.
[26] A. Harter, A. Hopper, P. Steggles, A.Ward and P.Webster. "The Anatomy of a Context-Aware Application". *Wireless Networks Volume 8*, 2002.
[27] http://www.cl.cam.ac.uk/research/dtg/attarchive/bat/, accessed 30 November 2010
[28] R. Want, A. Hopper, V. Falcao, and J. Gibbons. " The Active Badge Location System". *ACM Transactions on Information Systems (TOIS). Volume 10, Issue 1*, January 1992. pp 91-102.
[29] M. Kalkusch, T. Lidy, M. Knapp, G. Reitmayr, H. Kaufmann and D. Schmalstieg. "Structured Visual Markers for Indoor Pathfinding". *Augmented Reality Toolkit, In First IEEE International Workshop*, 2002.
[30] N. B. Priyantha, A. Chakraborthy and H. Balakrishnan. "The Cricket Location-Support System". *Proceedings of 6th annual International Conference on Mobile Computing and Networking*, August 2000.
[31] F. J. Gonzalez-Castano, J. García-Reinoso, F. Gil-Castineira, E. Costa-Montenegro and J. M. Pousada-Carballo. "Bluetooth-assisted context-awareness in educational data networks". *Computers & Education, Volume 45, Issue 1*, August 2005, pp 105-121.
[32] H. Attiya and J. Welch. "Distributed Computing: Fundamentals, Simulations and Advanced Topics Second Edition". 2004, pp 22.
[33] “Sentilla Programmer's Guide”, July 2008.
[34] R.C. Martin. "Agile Software Development, Principles, Patterns, and Practices". *Prentice Hall: 1st Edition*, 2002.
[35] http://electronicdesign.com/article/embedded/java-eclipses-zigbee20610.aspx, accessed 28 November 2010
[36] http://www.sentilla.com, accessed 4 April 2010
[37] R. J. Larsen, and M. L. Marx. "An Introduction to Mathematical Statistics and Its Applications". *4th edition. Pearson Prentice-Hall*, 2005.

Appendix A: Discussion on communication problems between motes from different vendors

Creating a hybrid network of Sentilla JCreates[1] and SunSPOTs[2] allows to generalize hypothesizes beyond the situation, where only a single hardware-platform is available. So the idea is to let SunSPOTs and the Sentilla motes communicate with each other and create a hybrid network.

We explore four different approaches that attempt to establish communication between Sentilla motes and SunSPOTs. The first approach just tries to send data-values. Sending an actual serialized Message-object is the second approach. Using a 802.15.4 MAC layer (or simply MAC-layer) to send the serialized Message-object is the third approach. Finally the fourth approach combines the third approach with encryption.

Approach 1 - sending raw data

A laptop is equipped with a Sentilla network-adapter. The laptop is running the “Sentilla Work environment”. In this environment the “Mote Network Traffic Console” is run. This window in the “Sentilla Work environment” allows seeing what network-traffic comes in from (Sentilla) motes.

In this experiment a SunSPOT is sending in sequence one short, and two longs. At the time of the experiment the user-defined Message-class of the location-tracking software only contains a short and two longs. Therefore it is tried to send the values of the short and the two longs to the laptop running the “Mote Network Traffic Console”. Sending the values in this way does not result in the “Mote Network Traffic Console” receiving them. In fact no indication of any type of receival of data is shown.

The most important reason why it is believed that this approach does not work is that Sentilla motes can only send messages that are Java-objects. These objects need to implement a well-known interface called Serializable. Therefore a SunSPOT that merely sends raw data in the form of a short or two longs is not sending an object but just raw data in the form of a byte sequence.

Approach 2 – serializing a Message-object and sending this as a byte-sequence

A laptop is equipped with a Sentilla network-adapter and running the network-monitor inside the Sentilla-environment. In this experiment an instantiated message-object is serialized and written to a file using Java-code written in the Sentilla environment. The byte-sequence is

[1] http://www.sentilla.com

[2] http://www.sunspotworld.com

read from the file and hardcoded (in hexadecimal format) in the SunSpot-code and sent in two messages as a byte-array to the Sentilla adapter of the laptop. The reason for sending the byte-sequence in two messages is because the total size of the serialized messages is 109 bytes, and according to the "Sentilla programmer's guide" only payloads of 86 bytes are allowed.

The reason why it is suspected that this approach fails is because the byte-arrays probably should be sent over a 802.15.4 MAC layer using the correct PAN-ID (Personal Area Network ID). The idea for this approach came from the general-faq of sunspotworld.com[3]. On this faq you could see the following question and answer (accessed on july 2010):

> *Can Sun SPOTs and Motes talk to each other?*
>
> > *In theory yes, but the current Sun SPOT radio stack is incompatible with the packet format used by the Motes. We do provide direct access to the 802.15.4 layer via the I802_15_4_MAC and I802_15_4_PHY interfaces (look inside com.sun.spot.peripheral.radio). Currently, the LowPAN layer sits on top 802.15.4. Since the motes do not use LowPAN, you'll need to create your own layer modelled after LowPAN that sits directly on 802.15.4.*

Another reason why it is suspected that this approach does not work is that the Sentilla motes seem to employ symmetrical encryption that is not used in this experiment. More on this in the next approach.

Approach 3 – serializing a Message-object and sending this as a byte-sequence using the MAC-layer

A laptop is equipped with a Sentilla network-adapter and running the network-monitor inside the Sentilla-environment. The sender is still a SunSPOT. In this experiment the 802.15.4 MAC layer is now used to send the bytes representing a serialized Message-object. The idea is that since both Sentilla motes and SunSPOTs support the 802.15.4 MAC layer, it is believed that some incoming packets should be seen in the network-monitor. Unfortunately this is not the case, and this approach also fails.

It turns out that Sentilla motes use symmetrical encryption when sending its messages and that the Sentilla-motes and SunSPOTs use something called a PAN-ID to filter messages that are meant for their PAN (Personal Area Network). Although it is possible to modify the PAN-ID used by the SunSPOTs, the PAN-ID used by the Sentilla motes could not be retrieved by either some API, documentation, posting in the support-section of the Sentillaforum or some other resource found on the internet.

Approach 4 - serializing a Message-object and sending it as a byte-sequence using the MAC-layer and employing symmetrical encryption

[3] http://www.sunspotworld.com/docs/general-faq.php

Same as the previous approach but now a piece of code from the programmer's guide is used to retrieve what symmetrical key the Sentilla motes are using to encrypt the data they send. The encryption-key is used to XOR it with the payload of the message (the serialized Message-object).This encrypted data is now being used to send it to the laptop. Of all experiments done to get communication started between SunSPOTs and Sentilla motes this approach seems the most promising. But unfortunately it does not work either.

One could speculate on what goes wrong here. Maybe it is the PAN-ID that is still unknown for the Sentilla-motes and that it mismatches the one used by the SunSPOTs. Perhaps Sentilla still does some undocumented things under the hood when sending messages. Either way it is still unknown why this experiment does not work.

To summarize the different approaches and the suggested reasons why communication fails Table 2 shown below is added.

Approach-number	Approach-description	Suggested reason for faillure
1	sending raw data-values	Sentilla motes can only send and receive Message-objects that implement a serializable interface. Also the same reasons why the 2nd approach fails can be named here.
2	sending a byte-sequence representing a serialized Message-object	The byte-sequence probably should be sent over a 802.15.4 MAC layer. Also the same reasons why the 3rd approach fails can be named here.
3	sending a byte-sequence over the MAC-layer representing a serialized Message-object	Sentilla motes use encryption when sending a Message. The sending SunSPOT does not use encryption. Also the same reasons why the 4th approach fails can be named here.
4	sending a byte-sequence over the MAC-layer representing an encrypted serialized Message-object	PAN-ID that is used by Sentilla-motes is unknown or some undocumented things happen when Sentilla motes send data.

Table 2: approaches for communication between Sentilla motes and SunSPOTs and the suggested reasons for their failure

Appendix B: Test-report for fixed-position experiment

Position-number	Room-number	Success-rate
1	B	80
2	B	**50**
3	B	**50**
4	B	80
5	B	90
6	B	70
7	B	80
8	B	80
9	B	60
10	B	100
11	A	90
12	A	90
13	A	90
14	A	**40**
15	A	100
16	A	90
17	A	**40**
18	A	80
19	A	70
20	A	80

Average success-rate room A	**77**
Average success-rate room B	**74**
Average total success-rate	**75,5**

Appendix C: Test-report for single fixed mote per room experiments

Experiment: First in room A then in room B		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,666667	0,823529
2	0,75	0,823529
3	**0,4375**	0,666667
4	0,875	0,6875
5	0,6875	0,705882
6	**0,533333**	0,866667
7	0,8	0,933333
8	**0,5**	0,785714
9	1	0,933333
10	0,875	0,733333
Average success-rates:	0,7125	0,7959487

Experiment: First in room B then in room A		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,833333	0,75
2	**0,470588**	0,666667
3	0,6875	0,8125
4	**0,352941**	0,923077
5	**0,4**	0,818182
6	**0,5**	0,75
7	**0,4**	0,933333
8	0,823529	0,8
9	0,8125	0,733333
10	0,666667	0,9375
Average success-rates:	0,5947058	0,8124592

Worst-case performances:	
Room A:	0,352941
Room B:	0,666667

Appendix D: Test-report for 2 fixed motes per room - NE, SE / NW, SW

Experiment: First in room A then in room B		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,647059	0,875
2	0,8125	0,8125
3	0,9375	**0,529412**
4	0,882353	**0,411765**
5	0,846154	**0,529412**
6	0,875	**0,588235**
7	0,875	0,625
8	0,888889	0,695652
9	0,75	**0,555556**
10	1	**0,588235**
Average success-rates:	0,8514455	0,6210767

Experiment: First in room B then in room A		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,647059	0,882353
2	0,818182	0,733333
3	0,647059	0,928571
4	0,733333	0,769231
5	0,777778	**0,5**
6	0,875	0,625
7	0,75	**0,529412**
8	0,823529	0,785714
9	0,705882	0,642857
10	0,764706	0,866667
Average success-rates:	0,7542528	0,7263138

Worst-case performances:	
Room A:	0,647059
Room B:	0,411765

Appendix E: Test-report for 2 fixed motes per room - NE, SW / NE, SW

Experiment: First in room A then in room B		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,888889	0,809524
2	0,8	0,823529
3	0,6875	0,904762
4	0,705882	0,882353
5	0,9375	0,764706
6	0,875	0,823529
7	0,8125	0,705882
8	0,75	0,823529
9	0,8	0,8125
10	0,8125	0,75
Average success-rates:	0,8069771	0,8100314

Experiment: First in room B then in room A		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,785714	0,941176
2	**0,588235**	0,933333
3	**0,529412**	0,9375
4	0,705882	1
5	**0,588235**	1
6	0,764706	0,875
7	**0,5**	0,866667
8	0,823529	0,866667
9	0,705882	0,928571
10	0,705882	0,785714
Average success-rates:	0,6697477	0,9134628

Worst-case performances:	
Room A:	0,5
Room B:	0,705882

Appendix F: Test-report for two fixed motes per room - NE, S / NE, SW

Experiment: First in room A then in room B		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	1	0,8125
2	0,666667	0,75
3	0,736842	0,789474
4	0,9375	0,833333
5	0,764706	0,823529
6	0,666667	0,882353
7	0,833333	0,941176
8	0,625	0,823529
9	0,85	0,842105
10	0,823529	0,9375
Average success-rates:	0,7904244	0,8435499

Experiment: First in room B then in room A		
Test-number	**Success-rate Room A**	**Success-rate Room B**
1	0,764706	0,8125
2	0,823529	0,75
3	0,777778	0,75
4	0,647059	0,75
5	0,823529	0,9375
6	0,8	0,764706
7	0,823529	0,866667
8	0,9375	0,8125
9	0,888889	0,833333
10	0,882353	0,875
Average success-rates:	0,8168872	0,8152206

Worst-case performances:	
Room A:	0,625
Room B:	0,75

Appendix G: Binomial pdf for small-sample hypothesis-test

k successes	Px(k)
<= 56	0,0000
57	0,0001
58	0,0002
59	0,0004
60	0,0010
61	0,0023
62	0,0050
63	0,0100
64	0,0186
65	0,0321
66	0,0510
67	0,0746
68	0,0999
69	0,1216
70	0,1337
71	0,1318
72	0,1154
73	0,0885
74	0,0586
75	0,0328
76	0,0151
77	0,0055
78	0,0015
79	0,0003
80	0,0000

Critical region for two-sided test with $\alpha = 0.05$:

k <=	63
k >=	76