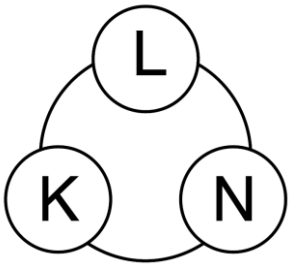


Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. Wolfgang Kellerer

Wireless Sensor Networks Laboratory Report

Museum Alarm System

Author:
Léo Dumas - leo.dumas@tum.de
Lukas Obkircher - lukas.obkircher@tum.de
Filippo Fuochi - filippo.fuochi@tum.de

Supervisors:
Vilgelm Mikhail: mikhail.vilgelm@tum.de
Samuele Zoppi: samuele.zoppi@tum.de

Begin: 14. April 2015
End: 14. July 2015

Kurzfassung

Im Rahmen des Projektpraktikums Wireless Sensor Networks wurde ein kabelloses Alarmnetzwerk für ein Museum entwickelt. Ziel des Projekts war es, wertvolle Objekte mittels verschiedener Sensorik vor Diebstahl und Vandalismus zu schützen. Hierfür wurde ein zentral gesteuertes Multihop-Netzwerk, bestehend aus Z1-Motes, realisiert, welches von einem PC aus überwacht werden kann. Gezeigt wird außerdem, wie eine Kontrolle des Netzwerks auf Funktionsfähigkeit im laufenden Betrieb durchgeführt werden kann und warum für die Datenübertragung eine sowohl Controlled-Flooding und auch Unicast verwendet wird. Ebenso wird erklärt wie die Routingtabelle nach den Kriterien Zuverlässigkeit sowie geringer Verzögerung erstellt wird und wie eine geringe Reaktionszeit im Alarmfall erreicht wird. Das umgesetzte System lässt die einzelnen Module mit deren jeweiliger Sensorik frei platzieren und durch das periodische Scannen der einzelnen Pfade, lässt sich ein Ausfall rasch feststellen. Bei sämtlichen durchgeführten Tests wurde ein Alarm stets rasch erkannt und umgehend am PC signalisiert.

Abstract

The main idea of the project was to protect valuable objects inside a museum from theft and vandalism. Therefore a centralized wireless multihop-network was created. The network consists of Z1-Motes and the state of the network can be checked with a visual interface on a computer. Depending on the network state, two different datalinks are used (controlled flooding and unicast) and there is also shown how the routing table is generated and why the whole network is reliable and fast. The developed system is able to automatically detect all motes nearby, regardless if the mote is directly in the central-motes range or is there are three hops in between and also it is able to check if a mote failed. The final tests showed that no fired alarm got lost and the system is as fast as desired.

Contents

Chapter 1

Introduction

1.1. Idea

The main idea was the development of a wireless museum alarm system to protect valuable objects from theft and detect vandalism. For the detection of illegal actions, different distance sensors are used and the system should also be able to detect movement during the closing hours. To protect the system itself and to improve maintenance, a system-failure-detection for each mote should be implemented. The user-interface is a PC-Application, which should display all-important information about the network and show which note failed or fired an alarm.

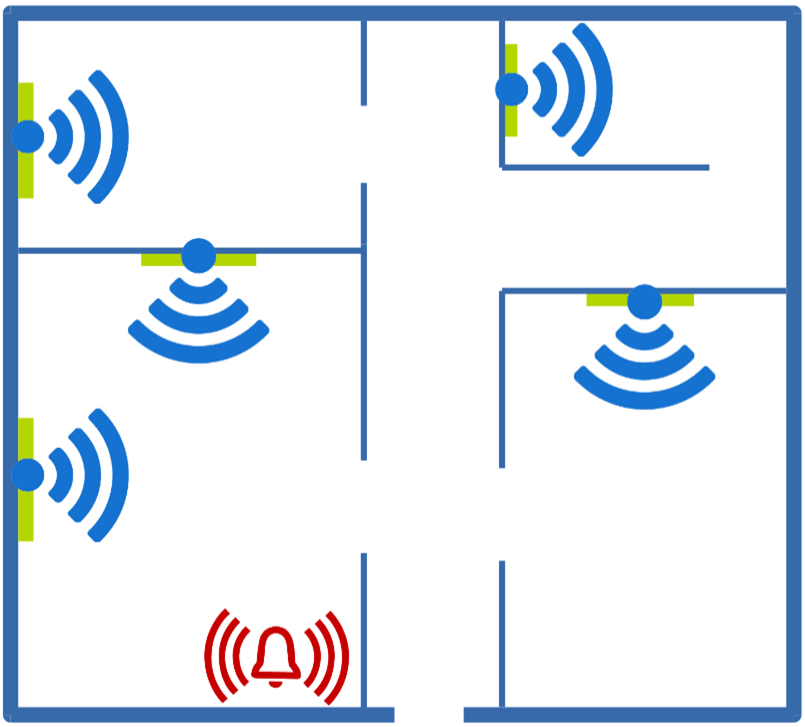


Figure 1: museum ground plan with monitored objects

1.2. Hardware

1.2.1 Motes

The wireless network consists of several Zolertia™ Z1-Motes, which are based on the Texas Instruments Zigbee RF-Transceiver CC2420 and a MSP430 microcontroller (1). USB is used to communicate with a PC and to flash a new firmware. The RF-part has already a chip-Antenna but there is also a possibility

to connect an external antenna. The two black connectors are analog inputs and for our project the distance sensors are connected to it.

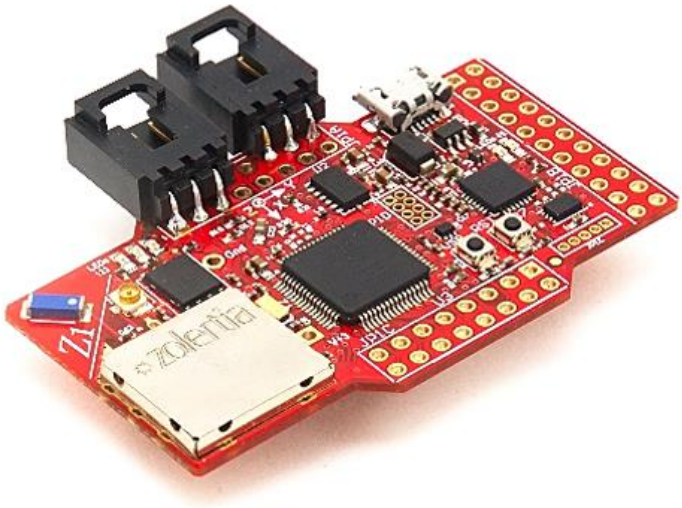


Figure 2: Z1-Mote (Source: wiki.lkn.ei.tum.de)

To power the board two alkaline AA-Batteries or USB can be used. There is also a temperature sensor, an accelerometer and a push-button on the board, but none of these are part of this project.

1.2.2 Sensors

To protect the valuable object, two different types of sensors are used. To detect if a person is getting to close to an object, IR-Distance sensors are used. Ultrasonic Distance sensors have a much larger measurement area than the IR-Distance sensors and that's why they are used for motion detection during the night hours.

IR-Distance Sensor

The Sharp distance sensor can measure distances between 10 and 80cm and it's supply power should be between 4.5 V and 5.5 V (2). To connect it to our Z1-Mote, an adapter has to be used.

Figure 3: Sharp IR-Distance Sensor (Source: www.phidgets.com)

The output of the infrared-sensor is analog and the voltage should be linear to the distance. But according to the selling webpage, the sensor output can vary from unit to unit and is also depending on the target characteristics. That's why we calibrated the sensor to accive a good accuracy:

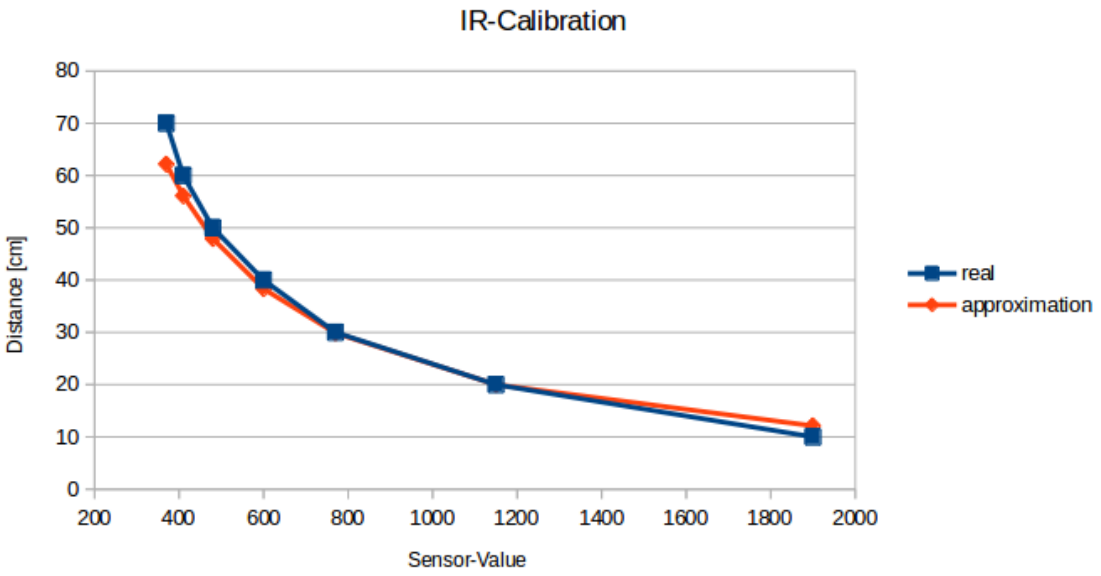


Figure 4: IR-Sensor calibration

The graph shows the real sensor output, measured by the ADC and our simple appriximation, which is basically one division. The appriximation could be better, but for the application the measurement hasn't to be that precise and for performance reasons we doesn't want to use the "Math.Pow"-function.

Ultrasonic-Distance Sensor

For movement detection, an ultrasonic sensor from MaxBotix is used. The sensor can measure distances of up to 6.45m and its supply range goes from 3 V to 5.5 V (3).

Figure 5: MaxBotix EZ-1 Sonar Sensor (Source: www.phidgets.com)

The sensor has three different outputs:

- analog
- digital RS232
- digital pulse width

In this project, only the analog output is used and the sensor itself doesn't need to be calibrated. The analog output is strongly linear to the distance.

1.3. Software

IDEs
For the development, two different IDEs have been used:

- Eclipse IDE for C/C++
- QT Creator

Eclipse is mainly used to develop the firmware of the Z1-Mote and to program it directly with a single click.

Operating System
The Z1-Motes use the open source Contiki OS, as their operating system. Contiki OS a tiny operating system, which is optimized for the "Internet of Things" and supports multiple platforms. [1]
On the PC, which is used for programming the motes, Ubuntu 14.04 LTS is installed.

Version Control
The distributed version control system GIT was used for this project and enabled an independent working flow with all the team members.

Chapter 2

Network Overview

2.1. Centralized

The wireless network is centralized and the main control belongs to a mote, which is connected directly to a computer. This centralized mote manages the whole network including

- Checking Mote-State
- Routing
- Recovery procedures
- Configuration of each mote
- Communication with Visual Interface

We decided to create a centralized network, because in a real museum you know where your motes are and you also know how many do you have. With a centralized version of a network it is easier to monitor all motes, without the need of additional fallback mechanisms. Also the sensor motes doesn't have to store any information about the network itself and that makes them cheaper and faster.

2.2. Datalinks

There are two different Datalinks used for the communication: Controlled flooding and Unicast. The main reason to switch between these two is the idea to keep the network load as low as possible.

[1] https://en.wikipedia.org/wiki/Contiki

2.2.1 Controlled flooding

Three different packages are used for controlled flooding:

- Flooding Search
- Flooding Ack
- Flooding Alarm

The main idea for using controlled flooding is to find each mote, without worrying about network-loops, and to create the routing table. How this is done is explained later at **Fehler! Verweisquelle konnte nicht gefunden werden.**. Another reason for using controlled flooding was to make the alarm as reliable as possible. As long as there is a possibility to get the information about the fired alarm to the central, information should arrive, even if the routing itself isn't completed.

2.2.2 Unicast

Also the unicast uses three different packages:

- Search
- Alive
- Config

The central mote searches periodically all motes and unicast is used to keep the network load as low as possible. Only the destination mote forwards or replies to the received package thus leads to a faster response time, which enables a higher scan frequency and a lower delay for the failure detection.
The third package is a configuration-message, which enables or disabled the sensor of an individual mote, for example disable all movement-detection-motes. Due to time limits, this feature wasn't implemented in the current project, but it could be added within a few working hours.

Chapter 3

Custom Protocols

3.1. Unicast Packages

Byte	**0**	**1**	**2**	**3**	**4**	**5**	6
Description	Source-ID	Dest-ID	0 = Search	Hopcount	Path	…	…
	Source-ID	Dest-ID	1 = Alive	Hopcount	Path	…	…
	Status	Dest-ID	2 = Config	Hopcount	Path	…	…

Table 1: Unicast Packages

All the packages contain in their third byte the type of the package. From now on, every package will be referred as the type of the package itself.
In this network, a unicast data link is used in order to develop two different functions:

- Periodical control of the presence of a mote
- Change of the status of the mote

The first function is achieved thanks to two different packages.
The first one is the "Search" package, it is sent from the central station towards the mote, which has to be checked. Inside this package, as can be seen in Tab. 4.1, there are the source and the destination ID, the type, the hopcount and the path, an array of mote IDs that define the path from the mote to the central station. The hopcount and the path are necessary in order to define the next hop at every mote, because of the centralized routing table.

The second one is the "Alive" package; this one is sent from the checked mote to the central station after the reception of the "Search" package. Also in this package there are hopcount and path in order to let every mote know the next hop. The meaning of this package is to let the central station know that the mote is still alive.

The second function is achieved by the "Config" package. With this package, it is possible to send a status in the first byte to the mote in order to activate or deactivate the sensor. This is useful during the opening hours, to switch off the sonar sensors, and if a painting has to be removed or replaced in order to not trig unwanted alarms.

3.2. Flooding Packages

Byte	**0**	**1**	**2**	**3**	**4**	**5**	6
Description	Source-ID	Dest-ID	3 = FAlarm	MSG-ID	Sensor Type	Distance-H	Distance-L
	Source-ID	Dest-ID	4 = FSearch	MSG-ID	-	-	-
	Source-ID	Dest-ID	5 = FAck	SUM-RSSI	Path	…	…

Table 2: Flooding Packages

Also the flooding data link is used to develop two different functions:

- Alarm detection
- Construction of the centralized routing table

After the sensor detect a value under a determined threshold, the mote flood the network with a "FAlarm" package. Inside this package there are, as can be seen in Tab. 4.1, the message ID, which is used in order to make the flooding controlled and prevent loop inside the network, the sensor type, necessary to know if the alarm is triggered by a movement detection or a close detection, and two bytes specifying the measured distance.

The routing table is important in order to use the unicast packages; in this way, it is possible to not overload the network. To construct and update the routing table, two package are used: The "FSearch" and the "FAck". The first one has the message ID in order to prevent loop in the network. The second one is more interesting, as can be seen in Tab. 4.1 it contains one byte called "SUM-RSSI", which is updated after every hop, this is the sum of the RSSIs over a define threshold, and the path, this is an array and every mote append its ID at the tail of the package before flooding the network. The prevention of loop inside the network is determined by controlling this array.

Chapter 4

Routing

4.1. Principle

There is only one routing table in the whole network, this is used to show that there is at least one path from the motes to the central station to guarantee that the alarm part can work, and it is used to send the unicast messages to every mote.

The routing table is built according our standards of reliability and fastness. The reliability is measured with the SUM-RSSI in the uplink section, and the fastness is related with the number of hops in the path.

4.2. Flooding-Search

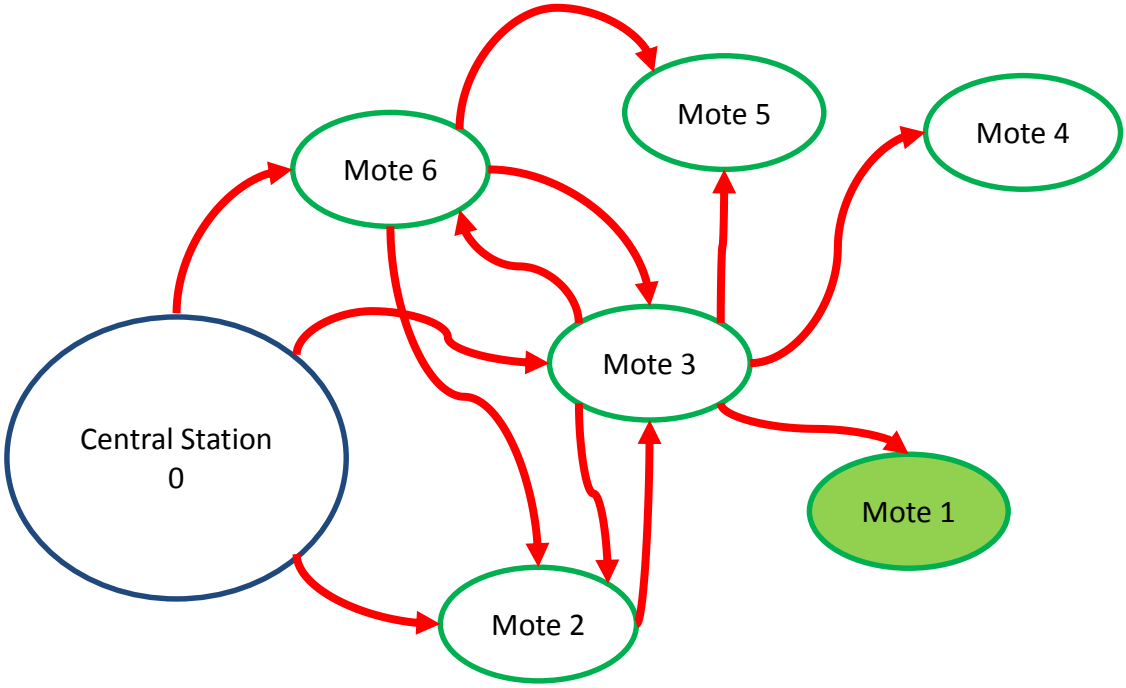


Figure 6: Flooding Search Procedure

The building process of the routing table starts from the central mote. It floods the network with a message "FSearch" setting the destination ID in order to find a single mote. As can be seen in Fig. 6, the network could be overload. To avoid that, the MSG-ID increases for every package sent from the central station. Thus every mote check the MSG-ID of a received packet: if it has already received that specific packet, then it won't forward the message. After the "Fsearch" message arrival at the searched mote, it will build and send a "FAck" package.

4.3. Flooding-Ack

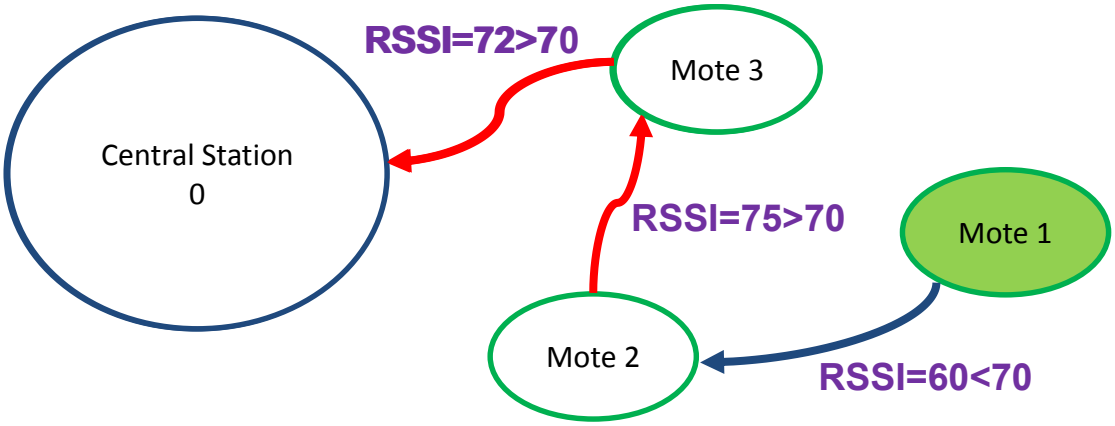


Figure 7: Focus on a single path of the Flooding Ack Procedure

The "FAck" message tries to reach the central station. Every mote inside the path change the received package in two different ways before forwarding it. First, it measures the RSSI of the previous hop and it compares that value with a threshold. If the measured value is higher than the threshold, then it adds the difference of the two values to the "SUM-RSSI" byte of the package. For example, looking at the path in Fig. 7, Mote 3 will add 75-70=5 at the "SUM-RSSI" before forwarding the message. The ID of the mote is also added at the tail of the package. In order to do this it starts a loop to find the first empty position of the array containing the path. Inside this loop, if it finds its ID, that signifies that it has already forwarded the package and it will not do that again.

4.4. Routing Table Construction

Byte	**0**	**1**	**2**	**3**	**4**	**5**	n
Description	Mote-ID	Found	SUM-RSSI	Hops	Path[0]	Path[1]	Path[n]

Table 3: Routing Table

When the "FAck" package reach the Central station then it starts to build the routing table. First of all the central station add the RSSI over the threshold at the value of the received "SUM-RSSI", for example in Fig. 7, it will add 2 at the previous 5 stored in the package. Then the received path can be stored if it fulfills one of the following three conditions:

1. There are no other path stored
2. The stored path has a greater SUM-RSSI
3. The stored path has equal SUM-RSSI but lower hops

By following these three criteria it is possible to save a path that is guaranteed to be the most reliable for the uplink and, probably, the fastest. Indeed the time spent in processing the package has usually a strong impact on the total time that a package needs to follow the entire path.

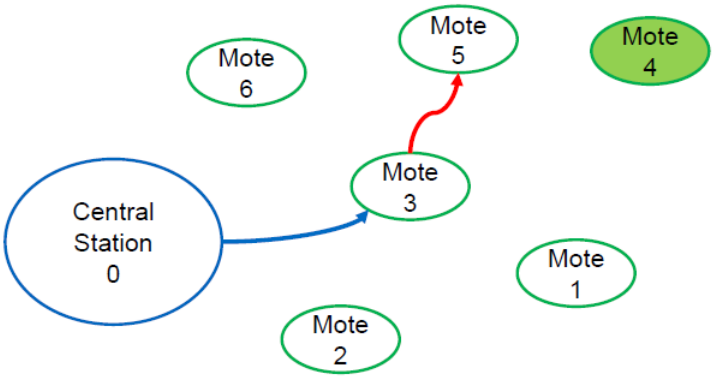
The routing table is built, as can be seen in Table 3, by setting the second byte to "1" when a path is found, copying the SUM-RSSI and the path from the received "FAck" message, and calculating the number of hops from the path array. As the path array is copied directly from the received message, the fifth byte represent the ID of the first forwarding mote in the uplink.

Chapter 5

System Behavior

5.1. Search

The routing table has been built according to the previous chapter. The path for each mote has been stored. We are going to use the Search package of the Unicast protocol in an infinite loop to reach each mote and check if it is alive. The Search loop continue to the next mote when the central station receives the alive acknowledgment, so at least until our unicast timeout expires (3 seconds).



Byte	0	1	2	3	4	5	6
Description	Source-ID	Dest-ID	0 = Search	Hopcount	Path		
Content	0	4	0	1	5	3	

Figure 8: Search example

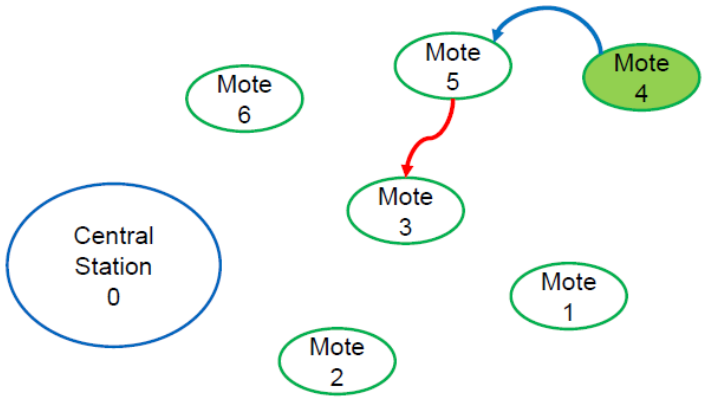
As seen in the Chapter 3 and shown in Fig. 8, the Search package has the path stored from the mote to the central station. Each time the message reach a mote, the hopcount is decreased and allows us to select the next mode ID. For a direct connection, the hopcount is zero so we never count the central station neither the target mote. To select the right ID, we choose the Byte corresponding to the beginning of the string path added with the hopcount minus one. In Fig. 8 example, we take the Byte = 4+1-1=4 => ID of mote 5. If the hopcount is zero we take the Destination ID.

5.2. Check if alive

When the target mode receives a Search package, it answers with the "Alive"-package to the central station. It restores the hopcount according to the length of the path and proceeds in a really close way than before. The difference is the way to select the right ID. Because the path is still stored from the mote to the central station, we choose the Byte corresponding to the end of the package, so the string path, minus the hopcount added with one. In Fig. 9 example, we take the Byte = 5-1+1=4 => ID of mote 3. If the hopcount is zero, we take again the Destination ID.

When the Alive package is received by the central station, the timer is directly set to zero and the Search loop is continued. If the Alive package is not received before the timeout expiration (3 seconds), a recovery procedure is launched by doing a Dynamic Source Routing to find the target mote as shown in

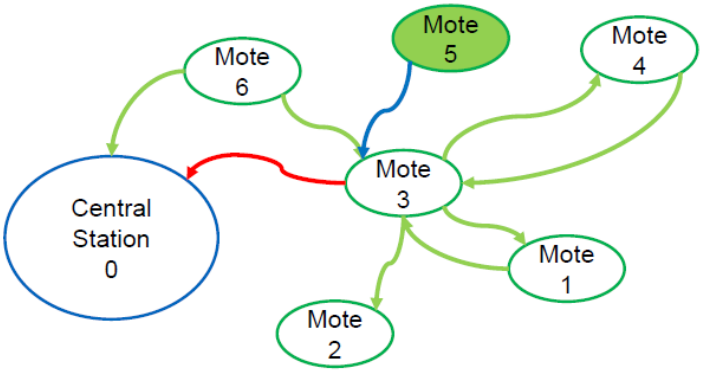
Chapter 4. We then adapt the routing table according to the best path received. If the flooding Ack is not received by the central station, we then indicate in the Visual Interface that the mote has a problem and we can proceed to a technical check of the network.



Byte	0	1	2	3	4	5	6
Description	Source-ID	Dest-ID	1 = Alive	Hopcount	Path		
Content	4	0	1	1	5	3	

Figure 9: Check if Alive example

5.3. Alarm



Byte	0	1	2	3	4	5	6
Description	Source-ID	Dest-ID	3 = FAlarm	MSG-ID	Sensor Type	Distance-H	Distance-L
Content	5	0	3	1	0	2	28

Figure 10: Alarm example

If a mote has detected an intrusion, it sends the Alarm package by controlled flooding. A mote can only flood to central station if its sensor is activated and that activation depends on the museum and on the role of the sensor.

As shown in Fig. 10, the ID of the mote, the type of the sensor and the distance of the intruder is stored in the package. We have here a movement detected at 5.4m by the sonar sensor of mote 5. As for other flooding packages, the network's overload is avoided by checking the message ID. When the central station receives the Alarm package, it fires the alarm and display indications about it on the Visual Interface.

Chapter 6

Visual Interface

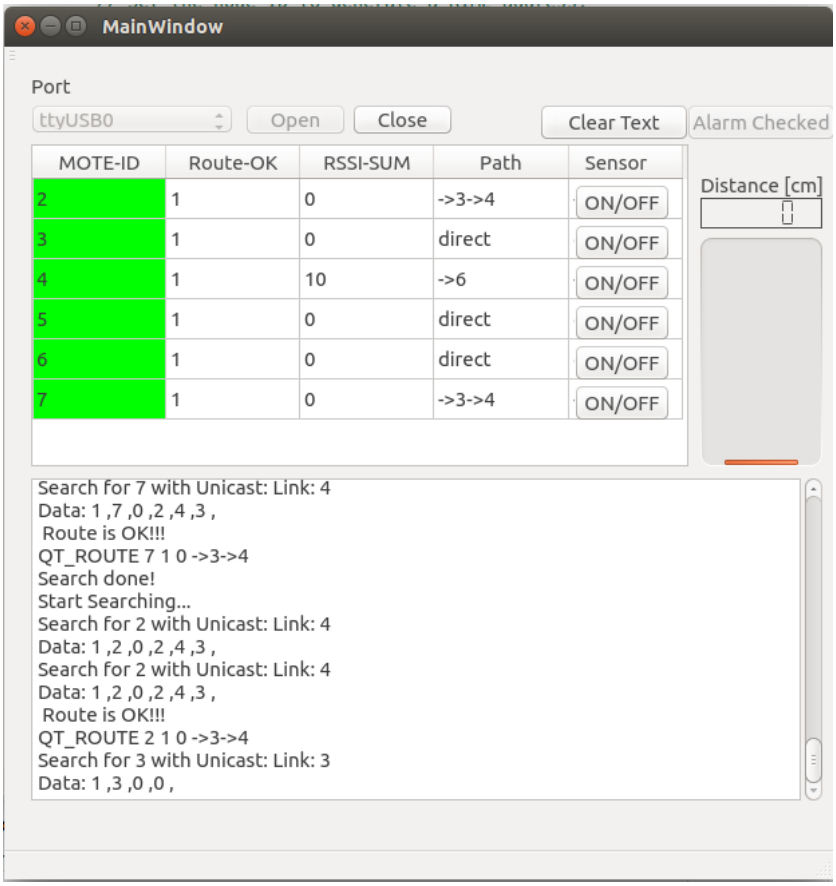


Figure 11: Visual Interface

The visual interface was realized with QT creator. First of all it allows to see the routing table, how it is built, and the reliability of the connection with the RSSI-SUM. The color of the mote ID indicate if the path was found, that is to say if the Flooding Search and Ack, the unicast Search and Check if Alive (or at least the recovery procedure) have worked. Green is the color assigned to a stored path and yellow indicates a problem to reach the mote. Red is be the color of the alarm.
For more information for debugging or taking a deeper look at the system, we display the messages received by the central station and what it is doing.

Several buttons were integrated to open/close the USB connection with the central station, clear the text of the bottom display. The alarm checked button remove the red color signal and restore the real color according to the network situation. If there is a path stored it will change to green, and yellow for the contrary. Thus, even with an intrusion detected, we are able to manage the network.
For a fired alarm, we also provide a display of the distance in cm that could be somehow useful.
Finally we have incorporated a sensor on/off button to use the Config package in order to deactivate the embedded sensors. We have not implemented it yet but it would be an improvement for a real test in real conditions.

Chapter 7

Conclusions and Outlook

In this project we have followed a centralized routing protocol approach using Dynamic Source Routing for a unicast network monitoring, and it was combined with controlled flooding for triggering alarms.
The system handle multi-hop network, is reliable and fast.
The visual interface is clear, enables to interact with the network, in real time and easy to use.
The weakness of the network is that it would be difficult to extend it with the controlled flooding.
Several improvements are possible. Firstly we could build another centralized routing table for the ride from the central station to the mote. Even if the unicast works pretty well, it could increase the reliability of this connection. We could also add other type of sensors to enhance the detection.
For a greater network, we could add a mesh of sub central mote that manage each one a multi-hop subnetwork.

References

1. **ZOLERTIA.** wiki.lkn.ei.tum.de. *Zolertia™ Z1 - Datasheet.* [Online] March 11, 2011. [Cited: 07 25, 2015.] https://wiki.lkn.ei.tum.de/_media/intern:lkn:all:lectures:ss2015:wsnlab:hardware:zolertia-z1-datasheet.pdf.
2. **MaxBotix Inc.** http://www.phidgets.com/. *LV-MaxSonar High Performance Sonar Range Finder.* [Online] 07 2007. [Cited: 07 25, 2015.] http://www.phidgets.com/documentation/Phidgets/1128_0_EZ1-Datasheet.pdf.
3. **Sharp.** http://www.phidgets.com/. *GP2Y0A21YK0F.* [Online] 12 1, 2006. [Cited: 07 25, 2015.] http://www.phidgets.com/documentation/Phidgets/3521_0_Datasheet.pdf.

Notation and Abbreviations

Ack	Acknowledge
ADC	Analog to Digital Converter
Config	Configuration
GUI	Graphical User Interface
ID	Identifier
IDE	Integrated Development Environment
IR	Infrared
OS	Operating System
RF	Radio Frequency
RSSI	Received Signal Strength Indication