A reliable multi-channel RSS measurement tool for Wireless Sensor Networks

Introduction/Motivation

Wireless Sensor Networks (WSN) are composed of tiny devices, that are often called sensor nodes and have a broad range of application areas. These applications require various types of information from their environment. Such as the location information of the sensor nodes is often desired. In some cases, the potential positions in which the nodes can be found are limited. The geographical positions of sensor nodes relative to each other can be detected by leveraging Received Signal Strength (RSS) information that contains enough diversity through sampling at multiple frequencies.

The goal of this project is to develop a tool to collect RSS information by each node from each other node at a given set of frequencies. The system this tool implements assumes that all nodes are equipped with identical radio chips and can communicate only through their IEEE 802.15.4 supported radio interface.

Requirements/Features

The basic requirement for a given network is having a sink-node that connects to a PC (for control and data collection) and we want all the nodes to send and receive measurement packets (in turns to avoid interference and bad measurements). When a node transmits a measurement packet all the others that are within its communication range store the received signal strength (RSS) value as well as the sender of that packet. Each node must later deliver all its measurements to the sink node, which will relay it to the PC.
Apart from this basic functionality there is a need for several other features:

- Do more than one measurement for each transmitting node
- Do measurement on different channels, rather than just on the default channel
- Detect nodes (in network range) dynamically and involve them in the measurement process
- Configurability of the system via PC
    - Number of measurements
    - Channels to do measurements on
    - Wait Timings
        - How long to search for nodes for an experiment
        - How long to wait for nodes when switching channels
- Reach nodes not in range of sink/control node, via multi hop

Challenges

- Reach nodes not in range only reachable indirectly (multi-hop)

- Robust against loss of connection to nodes
- Channel switching on all nodes without losing them
    - Strategy for guaranteeing that everyone got the signal before switching
- Keep measurement time on the same channel low
    - Channel stability varies over time and affects precision of results
- Collect all measurement data in reasonable time
    - Organized sending is required to prevent interference of sending nodes

Setup

Tmote-sky modules were used as sensor nodes. These were programmed with tinyOS in nesC. All development was done on Linux systems. The communication between PC and sink node is done over the emulated serial port but can be done in any way that is supported by tinyOS.

There are two powerful protocols implemented for tinyOS, namely “Disseminate’” and “Collection Tree Protocol”(CTP), which assure proper delivery of packets in the network supporting multi-hop to reach even nodes that are not in direct range of the sender. There we use the term ”disseminate” for the information injected into the network through Disseminate protocol and “collect” for the data carried by CTP. Disseminate issued to “broadcast” control-messages to all the nodes in the network to ensure they all react as they should. CTP, on the other side is used to collect data reliably to the sink node. This is basically for collecting measurement data to the sink node.

To process and evaluate the measurement data a provided R is used script which implements an algorithm that calculates the sequence given a number of measurements.

Design

To generate the requested measurement data, every node sends on each channel a measurement packet which is received by all other nodes. This data is collected from each node to the sink node which passes it to the connected Host (e.g. your computer).

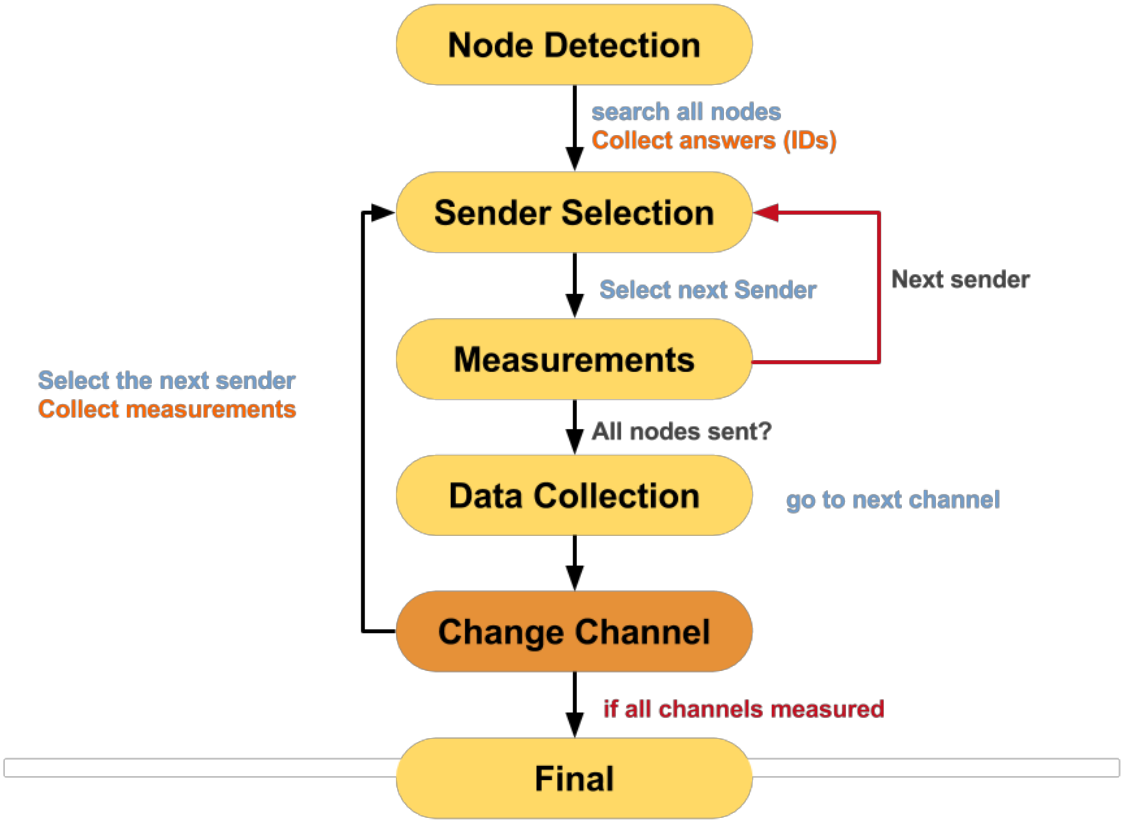
What follows is an in-depth view of the design to achieve the goal of generating and collecting all the measurements.

The nodes assume one of two rules:

- **Sink node:** the node, connected to the laptop. Manages the process
- **Measurement nodes:** transmit measurement packets, read and store the RSS information upon packet reception and send the collected RSS data to the sink node

The Sink node also acts as measurement node.

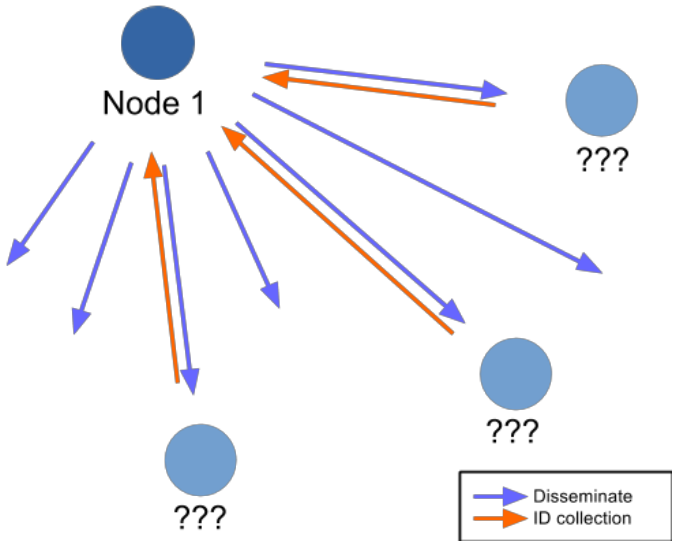
The design of the tool follows a state machine, which is executed on the sink node. The other nodes run by reacting on the sink node.



<statemachine.png>

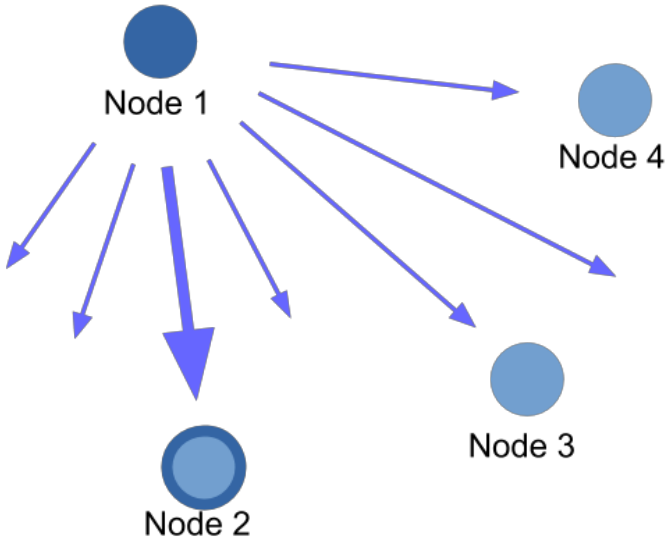
Node Detection

The system starts with the node detection. Here the Sink disseminates a request to all nodes in the network (also to himself), after requesting the Sink collects the IDs from all nodes in the network and stores them.
<Node_Detection.png>



Sender Selection

In the next state selects the sink node one node in the network, while disseminating his ID. Every node checks the disseminate value and compares its ID, when it is the same ID the node starts making measurements.



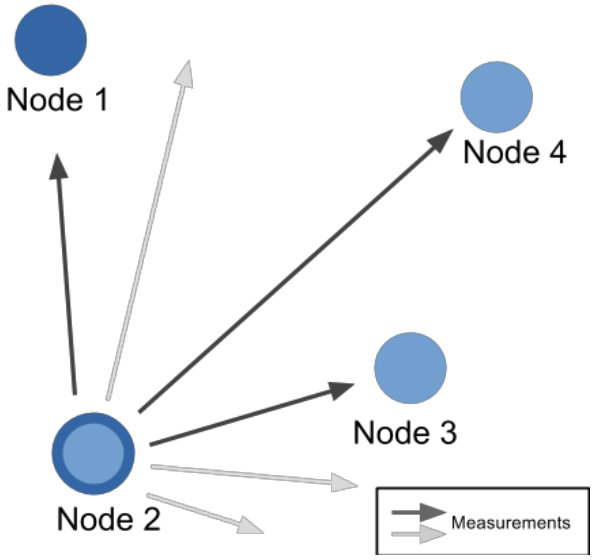
<Sender_Selection.png>

Measurement

The selected node starts making measurements on the current channel. After measuring through all nodes in range the node gives a signal to the sink whereupon the sink selects the next node.

We could configure the counts of measurements from the nodes in the *measurement* state (default number of measurements is 20) with our python script.

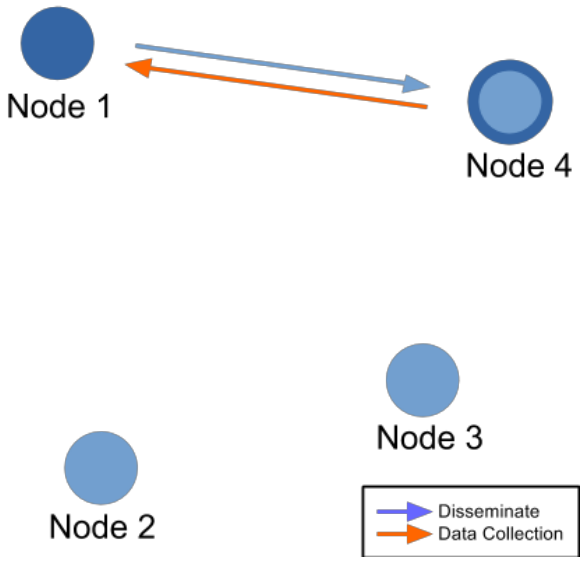
<Measurement.png>



Data Collection

When all nodes made their measurements the sink starts the Data Collection state. Here the sink disseminates the ID from the node equal to the *Sender Selection*.

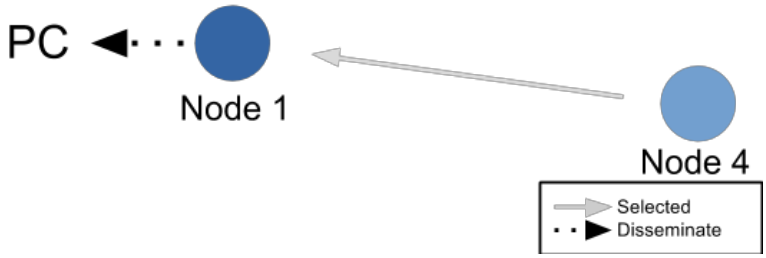
The selected node will send back data over CTP to the sink.



<Data_Collection.png>

Serial Transmit

Since the sink does not hold enough RAM for collecting all data packets from each node at once, but instead only has room for one packet, the sink has to transmit the information instantly via serial to the PC, as soon as the transmission took place.
There is one exception regarding the first step: Before the first data collection the sink has to transmit its own data to the PC.



<serial_transmit.png>

Channel Switching

After the collection of all nodes the sink node disseminates the new channel, on which the new measurements will come. We could also differentiate on which channel we measure. The disseminate will come on the “old” channel.

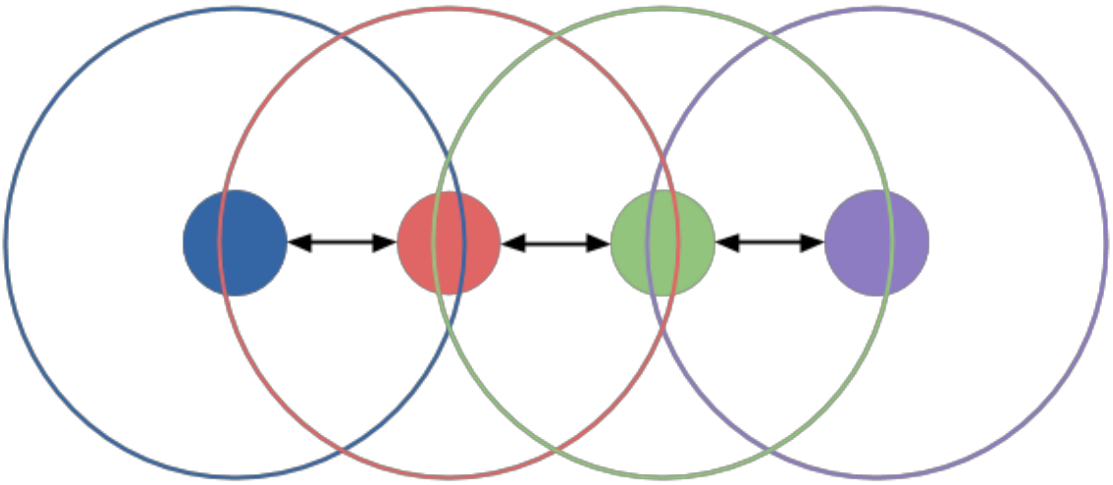
Multihop validation

Multihop is a major requirement of the system, because the sink node has to coordinate the nodes that are not in its direct range. Thus it was important to test its reliability in a dedicated experiment:

Configuring all nodes with the lowest power and setting them up with a constant distance between each pair of them, we get a setup where

1. The blue node cannot reach the purple one
2. Every nodes reaches its neighbors

The tests show, that signals from blue to purple only work if the red and the green nodes are active. This validates the multihop capability of the tool.



<multihop.png>

Recovery Methods

It may happen under certain circumstances that nodes don't receive the control signals they need to continue the measurement process of the system. We designed and implemented various system to recover these losses:

First of all, Disseminate and CTP try on their own to deliver messages even if they get lost. If they don't succeed we have a ReassignTimer.

The ReassignTimer waits after having assigned a node to do measurements. If the node didn't respond its success in doing measurements we retry assigning it by resetting the disseminate first to a null-value and than to the control signal to assign that node again.

If this procedure fails three times, we interpret the node being

- out of range
- turned off/broken
- or on another channel

which means that we won't be able to do further measurements with him. To not wait for him all the time in the ongoing experiment we delete him from the sinks' nodeList.
The lost node is maybe stuck on a channel and doesn't get any control signals to go on. This means that on further experiments it wouldn't be found and used, as it has to be on the base channel. To solve this issue we implemented a ResetTimer, that brings a node back to the base channel after it didn't receive any control messages in 20 seconds (configurable in source code).

Time

The most time during an experiment is spent on collecting the data. As the transfer rate highly depends on the connection quality(e.g. are nodes close to each other) this varies.
For good connections an experiment with 12 nodes, 4 measurements per channel on 16 channels took about 3 minutes while the same measurement with bad connections (i.e. higher distances between nodes) took about 9 minutes.
Time depends on the following factors:

- number of nodes in experiment
- number of channels
- number of measurements per channel
- distances between nodes
- transmission power

Limitations

The system is limited to start measurements on channel 11 always. The next channels can be chosen freely. You can choose no more than 16 channels.
As the memory of the nodes is limited and we have to cache measurements before transmitting them to the sink node(to keep channel-time low for better channel stability), the number of measurements we can collect in one channel is limited to 1000. This means that the number of nodes multiplied by the number of measurements per channel is limited to this number.
Please use parameters that won't exceed this limit (e.g. with 50 nodes in the system, do not more than 20 measurements per channel).
As the sink node can't store the measurement data, it has to be connected to a PC during the process to pass the data via Serial to the PC.
Due to data collection and multi-hop support the system might be relatively slow, depending on your setup (distance of the nodes, number of nodes, number of measurements).
We needed for 8 nodes 4.30 min with 10 measurements per node and channel on all channels(11-26).

Output

The measurements are saved in 7 columns. There is stored the *RSS* value from each *sender* to every *receiver*, on which *channel*, with which transmit *power*, *time* and the number of the measurement *packet*.

Note, that the transmit power is always set to 31, independent from the actual power used. It is only there for compatibility reasons.
The Time value is recorded by the python application and not by the sink node which means that the time values show when a package was sent to the PC and not when it was received on a node.

Snd	Rcv	Ch	RSS	PWR	Time	PacketNr
15	13	24	-77	31	1422996540	18
15	13	24	-77	31	1422996540	19
0	15	24	-65	31	1422996544	0
0	15	24	-65	31	1422996544	1
0	15	24	-65	31	1422996544	2
0	15	24	-65	31	1422996544	3
0	15	24	-65	31	1422996544	4
0	15	24	-65	31	1422996544	5
0	15	24	-65	31	1422996544	6
0	15	24	-65	31	1422996544	7
0	15	24	-65	31	1422996544	8
0	15	24	-65	31	1422996544	9
0	15	24	-65	31	1422996544	10
0	15	24	-65	31	1422996544	11
0	15	24	-65	31	1422996544	12
0	15	24	-65	31	1422996544	13

Usage

The nodes have 3 LEDs that show some information about ongoing measurements. The sink node shows different information, than the other ones:
For the sink node we use the 3 LEDs as a binary number to represent 7 stati. The red LED is bit 0, green is bit 1 and blue is bit 2. The stati are:

0. idle state
1. node detection
2. node selected
3. serial sink data state
4. data collection
5. changed channel
6. receive measurements
7. reserved state/not used

For the other nodes each LED has a meaning:
Blue: Measuring (sending measurement packets to be measured by all (other) nodes.
Red: collecting data (sending measured data to sink node)
Green: Is on base channel(11)
Please note that Blue and Red are not turned off after the action is finished, but first when another node is starting the corresponding action(measuring/collecting).

To control the measurements we wrote a python script which is best described by its "--help"-output:
<style: corrier new>

```
usage: host_controller.py [-h] [--measurements MEASUREMENTS]
                          [--channels CHANNELS] [--channelWait CHANNELWAIT]
                          [--senderChannelWait SENDERCHANNELWAIT]
                          [--idWait IDWAIT] [--outfile OUTFILE]
                          [--nodePath NODEPATH]

optional arguments:
  -h, --help            show this help message and exit
  --measurements MEASUREMENTS
                        How many measurements per node and channel (default:
                        20)
  --channels CHANNELS   The channel to measure on. comma separated (e.g.
                        11,12,13,14). Limited to 16 values. has to start with
                        11 (default: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
                        21, 22, 23, 24, 25, 26])
  --channelWait CHANNELWAIT
                        How much time to wait after a channel switch (default:
                        100)
  --senderChannelWait SENDERCHANNELWAIT
                        How much time to wait after a channel switch (sink
                        node) (default: 200)
  --idWait IDWAIT       How much time to wait for node ids? (default: 2000)
  --outfile OUTFILE     Write to stdout or to a filename (default: None)
  --nodePath NODEPATH   the path of the sink node (default:
                        serial@/dev/ttyUSB0:115200)
```

</style corrier new>

The program configures the sink node and tells it to start a new measurement. Then the sink node organizes the measurement process and passes the collected data back to the python script which either prints it to stdout or to a file (--outfile). The python script terminates when the sink node indicates the end of the measurement process, providing the possibility to do a bunch of different measurements automatically one after another.
The help messages from the script describe quite well how to use it.
As the script terminates after the experiment completed, to do multiple experiments just rerun the python script with your parameters.

Conclusion

The requirement for a reliable system for measuring and collecting RSS values was fulfilled. A tinyOS program was developed for the sensor nodes and a python script was developed to control the nodes.

The Python script sends control packets via serial to the sink node which does a measurement experiment based on the configuration in the control packet and returns the measured data.
The developed tool uses several techniques to ensure that measurement can be done on different channels and that there can be made a configurable number of measurements on each channel. All configuration can be made dynamically by calling the python script with well documented command line options.
The tool uses protocols and techniques to support sensor node networks where not all nodes can “hear” each other and so packages have to be forwarded by intermediate nodes to enable communication between all nodes.
Reliability of packet transfer comes mainly from the use of CTP and Disseminate protocols.