Masaryk University
Faculty of Informatics

Edu-hoc: Experimental and educational platform for wireless ad-hoc networking

Master's Thesis

Lukáš Němec

Brno, Fall 2016

Replace this page with a copy of the official signed thesis assignment and the copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Lukáš Němec

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

TODO thanks

Abstract

TODO abstract

Keywords

keyword1, keyword2, …

Contents

1 Introduction

Wireless sensor networks[1] are typically highly distributed networks with many individual devices. We consider networks with tens up to thousands of individual nodes[2] that communicate with each other over radio. The most common use case for these devices is monitoring near environment around them. Thus they are equipped with various sensors. Applications of these networks are very broad; they are being used to control forest fires [1], research whale migration patterns [2], or monitoring of the structural health of bridges and tunnels [3].

One of the goals of this thesis was to create multi-purpose WSN platform for Arduino devices [4]. That includes an initial design of such platform, implementation of all the required software and the actual build of the network itself. As it is today, in WSN world does not exist one unifying platform, that everyone would use. Therefore we opted to create our solution that would be the best fit for our use cases. Decisions related to design choices are discussed in chapter 2. Details about the build itself as well as a description of created software are included in chapter 3.

The other goal of this thesis was implementation and execution of two use cases for our newly created WSN platform. The first use case was a transformation of work done by Mathur et.al. [5] to the WSN environment. Their results show interesting potential in randomness extraction from radio fading channel. However, these were produced on devices capable of much more than what we used for the construction of our network (Mathur et.al. used both FPGA-based development platforms for 802.11 standard and laptops with Atheros WiFi cards compared to our Arduino devices). The principles of bit extraction from radio channel fading are described in chapter 4 and results are discussed in section 4.5.

The second use case of our network is related to the usually difficult first experience with WSN. As a remedy to this problem, the second use case is an educational one. The goal was to create an ad-hoc network, that would be easy to use and would help to introduce various

1. WSN
2. from high-level view each device can be considered a vertex or node in a network graph

concepts used in WSNs. This use case was demonstrated during seminars of PA197 course (Secure Network Design), and slightly modified versions of particular scenarios were used as marked assignments. Basic principles of this use case, detailed descriptions of scenarios and specifics of use during seminars are described in chapter 5.

2 Creating WSN network - Problem analysis

To create functioning WSN network, one must ask and answer many questions which define the shape and parameters of such network. These issues include following points:

1. What is the purpose of this network?
2. Which devices will be used?
3. Will the network be homogenous or heterogenous (only one device type or multiple ones)?
4. Where will the network be placed?
5. What, is the expected lifetime of a network?
6. What will be the power source for devices?
7. How shall we upload new code to devices?
8. How the results will be collected from the nodes?

Some of these are independent decisions; some are quite closely related one to another. Now we can elaborate on these questions a bit more and provide possible reasoning for each one. Numbers in parentheses refer to the point in the previous enumeration.

2.0.1 Network purpose (1)

The biggest decision and it dictates the options for all the others. General purpose or educational network will be entirely different from production network with just one goal only. There is either one sole purpose that is given from the beginning. Thus the network is designed with this goal in mind; let it be forest fire control, air pollution measurements, whales migration monitoring, or reporting free parking places [6]. Such network is perfectly fit for its goal and not so much fo the others.

On the other hand, a network for development purposes or one with education as one of its purposes cannot be application specific. It

has to be general as much as it can be. There might be many different requirements for the network through its lifetime, and it should satisfy all of them. Single purpose network will have devices for the specific job only (maybe even manufactured by required specifications). However, a multipurpose network might require different resources for different applications. One does not know what the requirements might be in the future.

2.0.2 Device selection (2) and (3)

Device selection is very much dependent on the network purpose, but there are some general options and directions one can prefer in certain situations.

First and foremost, is there going to be just one type of device or are we going to use multiple types of devices? It will be much easier to create and develop applications for a homogenous network. All devices share the same properties; all devices behave the same. We can use the same set of libraries and radio drivers; everything is almost guaranteed to be compatible.

Heterogeneous network, on the other hand, will be much more complex from the beginning. Just because the selection of devices that are capable of communication with each other is an issue of its own. The best option might be using some popular platform (e.g. ZigBee [7]) which is supported across many devices (unfortunately, in many cases this is not an option due to application needs). The most basic requirement is the same radio frequency, but sometimes one might be forced to write everything from scratch, just because the supported libraries are not cross platform compatible.

The other aspect of the device selection is its price, therefore for single purpose network, we are going to select the cheapest device that fulfils our needs without much of redundancy. There is no need for it. However is we do not know the purpose in advance, or if we expect it might change over time, then we would like to have devices which are a reasonable mixture of resources compared to their price. There is no point in a device that has the maximum amount of memory but lacks computational power. We might even consider devices which are a bit expensive right now, but we expect them to typically within

few years time. It might be interesting to have applications ready for such devices when they come down to the reasonable price.

2.0.3 Network placement and lifespan (4) (5)

Expected life of the network is closely related to the next issue regarding the power supply for the devices. However, we are also interested in the durability of such devices and possibly resistibility to the weather conditions, such as water, low or high temperatures or even high pressure. There are going to be extreme differences in devices selected for long term outdoor usage and ones placed inside.

Placement of these devices can even give us more choices regarding the power supply because certain locations might have an option of power harvesting. Let it be solar sources or geothermal ones; the options very much depend on expected surroundings of these devices.

On of the factors that would determine the nature of our network is allowing movement of particular nodes. Devices and options are inherently going to be different in the case of the static network compared to a dynamic one. A dynamic network will have much greater needs on redundancy and durability of each device. Also, the design of applications and protocols will have to be much more resilient to the possible events which might occur during the movement (neighbour devices will change, the mortality rate will be naturally much larger, the environment around the network will change).

In the static topology we can predict and assume many possible events, even prepare for them in advance. Therefore once a node is not active for given period, we assume it never will be active again (which in a dynamic network could just be a temporal change of topology and two former neighbours might be able to communicate again after some time). Thus once the static network is deployed (deployment can be randomised) we can save the current state and derive everything else from it. In the dynamic network, we cannot make any assumptions at all from the initial positions, and we have to adapt to the current situation during the whole lifetime of such network.

2.0.4 Power source (6)

Two options for the power supply are available, either supply from a battery or by wire. Let the wire be only power cable or combined with data (e.g. USB cable). Options are mostly determined by the device itself because most of the devices usually have some requirements regarding the power supply. For example, not all devices can be powered by five volts which are provided by a USB cable.

The least convenient method is direct power from the outlet, which unfortunately might sometimes be the only option. Usually, this is the case when the device demands an extraordinary amount of power which cannot be supplied in any other way or only for a short time. Such device, for example, might be required as a base station which provides a connection to the network to the internet. Usually, it will be a common personal computer or device of similar nature.

For a network with short life span, the ideal solution is battery powered devices (e.g. few hours to test some application in real conditions or throw off devices for short-term use). This way it is possible to keep the invested amount of resources low. Also, battery power is ideal for out of the grid networks, where any other solution would not be applicable. Here the main goal is to optimise the power usage to the absolute minimum, therefore to maximise the network lifetime.

For any other network, the optimal solution usually is low power supply via cable The cable usually also provides a data connection.

Alternative options for outdoor networks usually include some forms of energy harvesting; this might include solar power, geothermal energy or any other kind of power harvesting. This option is typically associated with a presence of a battery; that will supply power continually, and energy harvesting is used to recharge this battery. This is the usual case because many of these energy sources are not continuous and the received amount of power might vary through the day.

2.0.5 Code and data upload and download (7) and (8)

Code upload and download is usually only a problem for a long-term network. Primarily for the network with development or research purposes, because this goal usually requires frequent changes of source code, thus a convenient way of doing so. In any other kind of net-

work, software is typically uploaded only once and before the network deployment. However, we also should have a solution for doing so (it does not have to be convenient), because a device that cannot be updated is dangerous.

When the solution for code upload is required for already placed devices, there are two main options, either updated over the connected cable or some FOTA[1] solution. The latter one would be preferred for battery powered networks since it does not require any additional cables.

In a case of power provided through a cable, it is usually the best option to use this cable to provide software uploads and updates as well. The cable is already present, and it is the most reliable way of doing so. The same applies to data transfers; cable connection is the most stable and straightforward solution, when the cable is not present, such functions have to be provided via the radio.

2.1 Testbed design

For the construction of our testbed[2], questions from the previous part have been answered respectively:

1. What is the purpose of this network? There is no single specific application. It is generic type network with research and education purpose.
2. Which devices will be used? Arduino based devices with built-in radio 3.1.
3. Will the network be homogenous or heterogenous (only one device type or multiple ones)? Only one type of device, with the educational network we have to keep things simple.
4. Where will the network be placed? The network will be located inside a building, across multiple rooms.
5. What is the expected lifetime of a network? There is no set lifetime goal. The network will be operational as long as it will be

1. firmware over the air
2. WSN platform for application testing and experiments

required or until it will be replaced by another type of network in the future.

6. What will be the power source for devices? All devices are going to be connected via USB cable. Therefore this cable will provide the power.

7. How shall we upload new code to devices? Same as previous, through the USB cable.

8. How the results will be collected from the nodes? Either through the USB cable or using radio communication.

2.1.1 Possible challenges

Since the network is going to be permanent and without battery power, there are not that many challenges to solve. The main issue is the placement of the individual nodes together with all the supporting infrastructure. Which should be done in such manner, that everything would work properly, and the nodes should be distributed evenly across all the covered area.

3 TESTBED deployment

The most important part of the deployment of our network and creation of testbed was a selection of devices 3.1. Once we decided which devices to use, next part was the placement of these devices 3.2. Placement is quite important because nodes should be distributed evenly across the whole are of a network.

The last part connected with the deployment is the management of these devices. In our case, the whole network is managed by a central server. All nodes are connected to this machine via USB cables, and the server is equipped with our custom software 3.3.

3.1 Hardware selection

New network was designed with both research 4 and educational 5 use in mind. Therefore the selection of hardware for individual nodes was limited by these intended use cases. Since we already had an experience with the research type network [8] and its pitfalls related to the complicated use of specialised hardware, we opted for devices based on Arduino platform 3.1. These are designed with simplicity in mind (Arduino focus group are not only programmers but also artists and people without any advanced IT skills).

JeeNode USB and JeeLink Devices selected for the network are JeeLink nodes [9] (figure 3.1 shows the device). In addition to that, also several JeeNode devices [10] (shown on figure 3.2) were obtained. These two devices are very similar in many ways and can interact with each other; JeeNode USB is only a little bit more versatile in its use options.

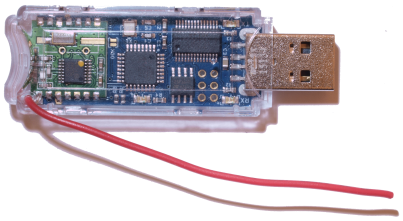


Figure 3.1: JeeLink

Both of these devices are based on Arduino Mini [11] (version with ATmega328P processor).

Wireless communication is enabled by the RF12B chip [12] (unmounted radio chip shown on figure 3.3) which is permanently mounted to the device. Both devices can be connected directly to a computer via USB[1]. The visual difference between these two devices is caused by the fact that JeeLink has a plastic cover. Therefore there is no additional connectivity. JeeNode is not covered, and there are pins available for additional sensors or any other devices. JeeLink has additional 16 Mbit of flash memory [9] available. Therefore this can be used for convenient storage.

For both of these devices, there is accompanying library JeeLib [13] that contains all necessary functions for controlling the radio. It also deals with any other features that are different from standard Arduino Mini.

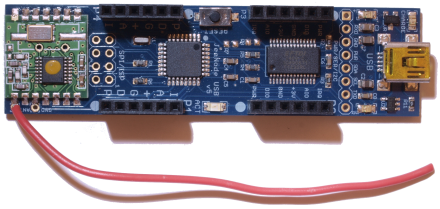


Figure 3.2: JeeNode USB

RF12B radio module Simple and cheap radio module RF12B from HopeRF [12] is present on all JeeMote and JeeLink devices. The version sold in Europe is using 868 MHz frequency for the radio[2]. Like many other common radio modules, this radio is half-duplex. Therefore it is able either to receive or to transmit messages but cannot do both at one time.

The radio module is equipped with omnidirectional whip antenna (≈ 8 cm long wire) by default. It is possible to modify the properties of such antenna by shaping it in various ways, e.g. straight wire compared to coiled one. There might also be many other versions and shapes of an-

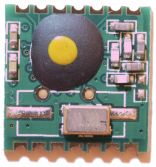


Figure 3.3: RF12B radio module

1. the USB port type one of the few differences
2. other version can use 433 MHz or 915 MHz

tennae, which might have an effect on radio properties, but this would be subject to whole another research in a different field.

3.2 Node placement

25 JeeLink nodes were deployed to achieve even coverage of the whole area. All devices are placed near the ceiling, where they are visible, but do not cause any obstructions. Due to a wire placement constraints, we decided to set these devices close to the walls. Plan with identified locations of each node can Alternatively, in appendix A.1 (each node is represented by a red triangle). The map shows only relative placement, exact coordinates together with IDs of each node can can in the appendix in table A.1.

3.3 JeeTool

Motivation Few nodes can be managed manually without any major problems, most frequently using Arduino IDE [4] directly. This approach is working without problems for up to three devices, and it would be possible to manage more devices this way, but the usability is lowered with each addition.

Either intentionally or not, there is no doubt that the official Arduino IDE is designed with simplicity in mind. Therefore it is usable for a person without a solid background in IT [14]. On the other hand, this simplicity comes at the cost of many features; that might have been useful for more advanced use (e.g. already mentioned management of multiple devices).

MoteTool inspiration JeeTool [15] was created to remedy this problem. A deployment tool that uses Arduino Makefile project [16] to manage our testbed. This tool was inspired by motetool project [17] which was developed by Dusan Klinec to manage the similar task, only for the different type of devices. The basic functionality is the same; it allows users to detect all connected devices and upload applications to the selected ones. Also, jeeTool includes support for bidirectional communication with individual devices via a serial port.

Implementation details As has been stated, Arduino Makefile is used for the actual application upload. To make Makefile call for each of the connected devices, jeeTool creates temporary bash script file, which is used to set the environment for the specific node and execute the upload via Makefile. The temporary script itself is then executed with system call directly from jeeTool. If multiple nodes are selected for upload, jeeTool offers either serial deployment (mostly for debugging purposes) and parallel deployment, where all Makefile calls are made within discrete threads, thus all of them are executed simultaneously.

Communication over serial port In addition to the upload features, jeeTool also enables communication with devices over a serial port. This is done via jSerialComm library [18] at the moment. In the past RxTx library [19] was used, unfortunately, this library is currently not being developed[3]. Therefore we decided to change the library to more up to date one, instead of using problematic old code.

Individual nodes are matched with files, which are named after the nodes (name of the device in the Linux file system). JeeTool is capable of writing all inbound communication in such file or read such file and write it over to the serial port. For the inbound communication, there is an option of delayed termination of the program. Thus jeeTool can be left running for specified amount of time before the ports to devices are closed (e.g. collect results after the experiment). For the outbound communication, jeeTool can send the whole content of the file at once, or there is an option of delay between individual lines of the file (can be used for periodic sending of messages).

3. last update is from 2011

4 Research use

This chapter describes our case study for key generation mechanism without any pre-shared information between individual nodes.

In today's application, this would be quite a common problem since numbers of devices equipped with radio module are constantly growing. Let it be Wi-Fi, Bluetooth, or any other frequency and specification. However, a problem for these devices is a source of randomness. Many of these devices are too small, or their price is too low. Therefore they do not come with a true random number generator. Thus key derivation is a bit challenge on these devices.

Results of Mathur et.al. [5] show a scheme for key derivation without any shared secret, only using properties of radio channel shared between these two devices and the nature of radio wave propagation through such shared channel. We show that this algorithm can be adopted in WSN environment.

4.1 Keys from radio signal

Due to a nature of a wireless channel which is not perfect, we can observe fluctuations and disturbances in the wave propagation. It might seem difficult at first to measure these since we are not able to observe every such event. Almost every radio module can measure RSSI[1], which can be described as an amount of power in the received signal. However, there are many arbitrary scales, and absolute values do not match on devices from different manufacturers since RSSI is relative index [20].

Still, our algorithm can work regardless of the differences in devices. Even two devices with the same configuration can produce different absolute values due to antenna position and overall device manufacturing differences. (many of these devices are partially assembled by hand [2].)

1. received signal strength indication
2. e.g. JeeNode motes come as a ready to assemble kit

4.1.1 Quantization principle

Quantization enables extraction of bits from individual values of signal strength. There are many different solutions to this problem [21], two main approaches here are: lossless quantization [22] and contrary to that we have lossy quantization [5].

Quantization is always lossy by design; we transform analog measurements of RSSI to single bits. However, the difference between two mentioned approaches is a number of generated bits per single RSSI measurement. Lossless quantization produces bit value from every measurement of signal strength, which is useful for high-performance demands, but it requires guaranteed variance in the radio channel (e.g. the nodes are constantly moving, or the environment is changing) Otherwise, the resulting keys could be weak.

Lossy quantization, on the other hand, does not have guaranteed output length per number of measured values, which can lead to a very limited length of output sequences. However, this kind of quantization is expected to have better results in static environments [21] because of its nature to drop bits which fail to differ from others.

Since our network is static and without any moving nodes, we implemented lossy quantizer algorithm designed by Mathur et.al. [5], which showed promising results for the off-the-shelf wireless devices. Authors also provided experimental results from scenarios comparable to our conditions.

4.1.2 Quantization algorithm

Nodes A and B exchange messages between each other, preferably at a high rate of exchanged messages. Each node records RSSI measurement for every received message. Thus both nodes record time series of RSSI values. These values are processed, and a threshold is computed. It depends on the algorithm whether one or multiple thresholds are used. Finally, we process RSSI measurement once again, all measurements above the threshold are treated as bit value 1 and all measurements below the threshold as bit value 0. In a case of multiple thresholds, we drop measurements in between. Example with two thresholds is shown on figure 4.1.

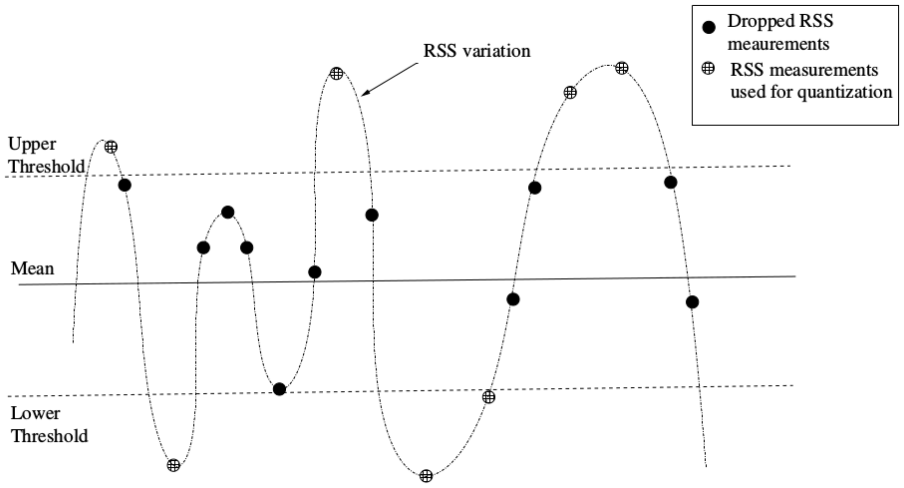


Figure 4.1: Sample quantizer. Values between tresholds are dropped. Values greater than the upper treshold are encoded as 1, values below lower treshold are encoded as 0. Therefore the output in this case will be 1010011 [21].

Quantization princeple designed by Mathur et.al. [5] works as follows:

1. Both nodes send n messages to each other in alternating pattern. The counter value is placed inside these messages, which is used to synchronise messages on nodes. For every one of these messages, signal strength is measured upon reception.

2. When n messages have been successfully exchanged, both nodes can proceed to the computational part.

3. Both nodes calculate mean m and standard deviation sd for signal strength values of all received messages.

4. We set $0 < \alpha < 1$ as a parameter to thresholds proportions. The larger value of α, fewer bits will be produced.

5. Both nodes calculate q^+ and q^- values, which are upper and lower quantizer bin boundaries, as follows:

$$q^+ = m + \alpha \cdot sd$$

$$q^- = m - \alpha \cdot sd$$

6. Every signal strength measurement is then processed, and it is rejected, if it lies within q^+ and q^- boundaries, values above this range are assigned a bit value of one, values below are assigned a bit value of zero.

7. Nodes then synchronise their measurement by exchanging list of counter values, where signal strength measurements were assigned either one or zero bit values.

8. Counter values that match on both nodes are expected to be excursions in the same direction and are used to produce the resulting bit sequence.

9. Final bit sequence can also be created in a different way. Consecutive measurements on one side of a threshold can be treated as one bit in the outcome instead of multiple ones.

As a result, we obtain bit sequence on both participating nodes. We can use the output directly as a key or process the bit sequence additionally. The preferred option would be entropy extraction, which can be done in many ways, most easily with cryptographic hash function.

We also have to be able to detect results with a little entropy. Moreover, since these occur rather frequently, we would like to create strategies to improve our situation.

4.2 Improvement strategies

As we can see from data collected in experiments during undisturbed network 4.5.3, there are no bits generated, and that is the case most of the time. To deal with this problem, we have to introduce some form of artificial noise or disturbances.

4.2.1 Cooperative jamming

The solution that offers itself is the help of other nodes within the network. All of them are equipped with radio, and we only have to work around the media access control mechanism to make two or more of them broadcast at the same time. Such idea has already been researched, e.g. in work of Goel et.al. [23] or by He et.al. [24].

This approach is primarily used to increase secrecy of a wiretap channel [25] but can also be utilised in other ways. We presume one device with multiple antennas or two or more cooperating devices involved in this process. While one device is broadcasting the actual information, other device or antennae are used to create artificial noise in the radio channel.

This noise can be of various sources [24]. We can use random noise, which has the same impact on every receiver in the network. Or we can use more advanced techniques, like using random codebook or structured codebook [24]. With these techniques, the receiver can distinguish the jamming and extract it from received signal. Thus the receiving party, who has knowledge of used codebook, can decode the jamming signal and remove this interference.

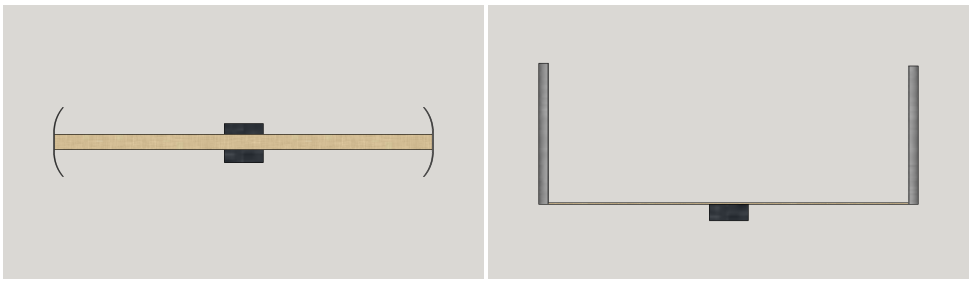
In our situation, we are only interested in the addition of artificial noise to the radio channel. Such noise should preferably be unpredictable in transmission power and time.

4.2.2 Other means of disturbances

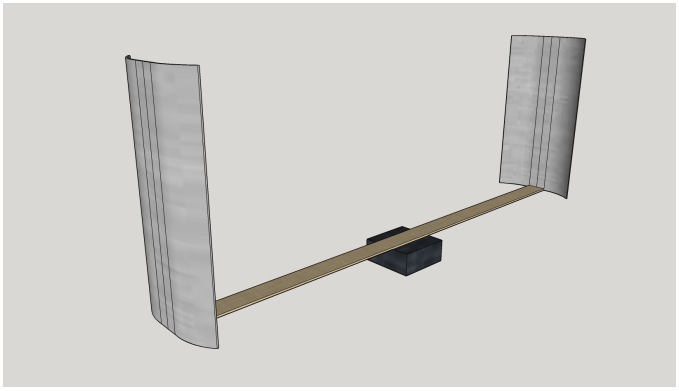
Artificial noise is not the only source of disturbances we can utilise to our advantage. There are many different sources, some might be more suitable for our purposes than others. During our experiments we examined the following two:

Movement of people Initially, we tried to avoid this kind of disturbance as much as was possible, because the movement of people is unpredictable (to certain extend) and such experiments are very hard (nearly impossible) to replicate.

After we acknowledged the need for artificial disturbances, movement of people was the most easy one to utilise. Unless we place our network in isolated environment, we can usually expect movement



(a) Top view (b) Front view



(c) 3D view

Figure 4.2: Aperance of our device

of people within the area of our network. Although this movement can be arranged, in most situations we can expect it to be somehow randomised. Therefore, the way it affects our RSSI measurements is also unpredictable to certain extend.

Mechanical obstructions Inspired by the results we obtained from experiments with movement of people, we tried to produce similar disturbances with a use of a mechanical device. This device was constructed from cardboard (structure itself) and aluminium foil (side shields), and its movement was enabled by a single stepper motor. Appearance of our device can be seen on figure 4.2.

Device rotates around the central axis and in 15 seconds accomplishes one complete turn. One of the nodes is placed inside of a

perimeter of our device. Therefore, the transmitted radio signal is obstructed by the moving shielding from aluminium foil.

4.3 Experiment setting

To collect data from a real-world network we created a custom application. Such data then were processed by quantization algorithm 4.1.2 and produced bit sequences were analysed regarding the entropy and usability for cryptographic keys 4.5.

By default, JeeNode platform is not able to directly obtain analog value of RSSI from radio module. The addition of short jumper wire between corresponding radio module pin and Arduino analog input pin is required [26].

In the experiment itself, two motes exchange messages with a counter value (counter is updated only on one of these motes). Both motes save measured RSSI values for each message. The counter value is used to guarantee delivery of these messages, as mote A sends a message with value n, mote B upon reception responds with the same value n. Mote A can increment counter value only when respond message with current counter value is received. Otherwise the same counter value n has to be sent again and both motes will keep the RSSI measurement only from the most recent message with counter n.

The application is programmed in such way, that 1000 counter values are exchanged (the number of messages is limited by the available memory of our devices). Subsequently, all RSSI measurements are sent out via the serial port and application is terminated. Intervals between individual messages should be as short as possible so that the disturbances in the radio channel can be captured on both nodes. Even though RSSI measurement is not symmetrical, using this approach we can expect them to be highly correlated [21].

Since the bottleneck of our experiment was in the communication via the serial port, we opted to exchange all messages and gather data first and only after that to transfer them out to the server. The application was then uploaded to the nodes again and run from original state several times as the experiment settings required.

In the optimal conditions, the expected time for a single run is about 30 seconds; runtime over two minutes is usually a result of

some error. Such error can be in the application itself, placement of devices, or disturbances in the radio channel.

4.3.1 Attacker model

In our experiments we consider passive attacker, which can be present anywhere in the network, even at several places at once. This attacker is represented by sniffer node. We used up to two sniffer devices at the time of application runtime. These were uploaded with modified version of an application, so that they would only receive messages, but would not send any messages to other devices.

We have done several trials with a placement of these sniffer motes, to see any impact of position in the network on the results. In a case of multiple sniffer nodes we expected them to share every piece of information. Therefore, we assumed the worst case scenario when we processed all gathered data from multiple sniffers.

4.4 Node placement

We have made several runs of our application with different settings. Nodes A and B were positioned always on the same places (sniffer and jammer positions were altered). Positions can be seen on figure 4.3.

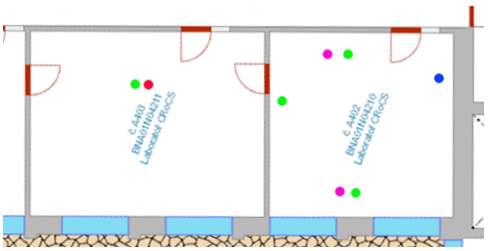


Figure 4.3: Placement of nodes. Red - node A, Blue - node B, Green - sniffer, Purple - jammer

4.5 Performance Evaluation

Quatization algorithm 4.1.2 is parametrised by α value. In evaluation of our experiments we had set $\alpha = 1/2$. This setting favours higher output rate, because smaller excursions exceed the treshold. Therefore we obtain more data for analysis. If we were to use the output directly as a key, then the setting should be different, preferably calibrated to the environment of the network.

To analyse the gathered data we used set of R scripts and all data postprocessing was done outside of the nodes. In the real settings, devices would have to do all of the computations on their own. In order to evaluate the produced bit sequences, we used min-entropy 4.5.1 and frequency test 4.5.2. Both were executed on eight and four-bit chunks of data. The goal was to obtain several measurements of entropy within the data and then use the most conservative estimate of contained randomness. Since we had collected only small amounts of data for individual sources, we decided not to use any of statistical test batteries [27].

4.5.1 Min entropy

Min-entropy [28] is considered the most conservative estimate of unpredictability in information theory. To make a min-entropy estimate of IID[3] source we can use simple calculation as follows, given the dataset with N samples:

1. Find the most frequent value and let C_{MAX} equal number of occurences of this value in dataset.
2. $pmax = \frac{C_{MAX}}{N}$
3. $C_{BOUND} = C_{MAX} + 2.3 \cdot \sqrt{N \cdot pmax \cdot (1 - pmax)}$
4. $H = -\log_2(\frac{C_{BOUND}}{N})$
5. W is the number of bits in each sample of dataset.
6. Min entropy is then equal to $\min(W, H)$

It is important to note, that this calculation of min-entropy estimate is only meaningful for IID sources [28], for non-IID sources we have to use other means. Testing sources with dependencies with this min-entropy calculation may result in overestimates and generally output without any meaningful use. Please note, our source is most likely non-IID source.

3. Independend and identically distributed

There are many tests for conservative estimates of entropy contained within the output of non-IID sources, most notably collision test, partial collection test or the compression test [28]. To us the most meaningful one was the frequency test[28], because of its ability to work on small amounts of data. A larger amount of data only increases the accuracy, but the test can provide a meaningful estimate on any amount of data provided.

4.5.2 Frequency test

On first sight, the calculation is similar to the original min-entropy 4.5.1. The calculation is following, using 0.005 as confidence level α:

1. Count occurences of each value in the dataset as $count_i$ for all $i = 1 \ldots n$ where n means all possible outputs.
2. Calculate ϵ from confidence level α and number of observations in dataset.
$$\epsilon = \sqrt{\frac{\log(\frac{1}{1-\alpha})}{2 \cdot N}}$$
3. $pmax$ is the probability of most likely observation value.
$$pmax = max_{i=1..n}(\frac{count_i}{N})$$
4. Resulting estimate of entropy is
$$min - entropy = -\log_2(pmax + \epsilon)$$

Using these two approaches we can get the most conservative estimate of entropy contained within our gathered data.

Results from all experiments are summarised in the table 4.1 and then results for each source are discussed in following subsections.

4.5.3 Plain source

As we can see from the table 4.1, there were no shared secret bits produced from this experiment. The reason behind is simple, there were

Sources:	Plain	Jammer	Moving people 4.2.2	Mechanical obstructions 4.2.2
Total runs:	40	10	424	26
Usable runs I. ($sd \neq 0$)	0	0	256	25
Avg. bits produced I.	0	0	189.41	238.48
Avg. errors produced	-	-	0.41	0.00
Avg. errors on sniffer	-	-	51.00	59.88
Usable runs II. (Both 1s and 0s in output)	-	-	97	10
Avg. bits produced II.	-	-	409.06	470.00
Min entropy (8bit chunks)	-	-	25.96	34.21
Min entropy (4bit chunks)	-	-	50.64	68.39
Frequency test (8bit chunks)	-	-	42.45	52.62
Frequency test (4bit chunks)	-	-	72.40	96.01

Table 4.1: table of sources evaluation

no disturbances in the radio channel, therefore all measurements of RSSI resulted in the same value. Therefore the quantization algorithm 4.1.2 computes sd equal to 0, thus fails to establish bounds q^+ and q^- and there are no bits produced on the output. Sample visualisation can be seen in figure B.1.

Since there are no bits produced, there is no point in measuring the entropy of the output. This result can be considered as the pitfall of static networks. Without any moving parts or changes in the environment, there is not enough entropy in the radio channel as such. Therefore no entropy can be extracted because there is none. Of course,

the greater the area covered by network, disturbances are more probable. We have to be aware of this problem and be prepared to identify such environment. The main reason being any bits generated in such environment should not be used [21].

4.5.4 Jammer

Friendly jamming can be used as a possible remedy to the problem of static network 4.2.1, creating artificial noise in the network and thus possibly adding some entropy (given that the effects of jamming are not predictable).

As the results in the table 4.1 show, this approach was not successful, mostly because of limitations of used radio modules 3.1. While this method would work with more advanced radios that are capable of subtractions on the signal level and allow for more devices to broadcast at one time. RF12 radio module and its accompanying library, however, does not permit this. Mostly because of primitive collision avoidance algorithm used (based on simple principle, not to send messages when there is any other signal present in the wireless medium).

Therefore our jammer, although jammer itself was able to broadcast anytime, failed. The legitimate nodes A and B could not receive nor broadcast their messages, because of the collision avoidance algorithm. Thus the only effect of jammer was prolonged time needed to exchange 1000 messages between devices.

To counter this behaviour, we tried to limit the transmission power used by sniffer, even using it without the antenna. Once the signal was limited to the level, when nodes A and B would not be interrupted by it. We observed same results as if the jammer was not present at all.

Although there might be other options how to make this approach work, we limited ourselves to using stock radio modules and available software for radio management. Therefore, we were not able to produce any secret shared bits using this method.

Thus we leave out this option for more advanced users (who might be able to control the radio on the most elemental level, defining own collision avoidance rules, etc.) or users with more specialised hardware (e.g. software defined radio modules).

4.5.5 Moving people

The data from an experiment with the movement of people clearly show some patterns, mostly we can observe timeframes which produce bit sequences with high entropy and also timeframes where no bits were generated. Since our experiments were conducted during lectures, we can expect, that the movement was not during the entire time frame of our experiments. Rather we expect time periods with movement and periods without it, which corresponds to our data.

Most of the time people remained stationary, therefore with almost no effect on the network. Therefore no secret shared bits were produced during these runs. Once somebody decided to move, we can clearly see excursions in the measured RSSI values. Although movement of people is not random, it can be considered not predictable. Therefore it might be sufficient source of randomness for our purposes.

One sample run from this experiment is shown on figure 4.4. This run generated 161 bits secret shared bits, with zero errors between nodes A and B. Sniffer node made 90 errors, thus about half of the bit sequence was compromised.

The results show promising values and more research should be done in this direction. We can see a clear trend in data, once we filter out unusable runs (either because no bits were produced, or because produced bit sequence contained only zeroes). About one-quarter of runs is suitable for bit extraction. Of course, these runs are not evenly distributed in our sample; they are rather clustered. On average, we can generate about 400 bits from one successful run. As we can see from the results of min-entropy 4.5.1 and frequency test 4.5.2, the most conservative estimate is about 26 bits of entropy contained within these data. On average we would like to utilise five consecutive runs, to generate a good key.

4.5.6 Moving shielding

In the last experiment, motivated by the promising data from previous one, we tried to introduce disturbances in the radio channel without the presence of any person 4.2.2. In the table we can see, that this approach achieved almost the same results as the previous one 4.5.5.

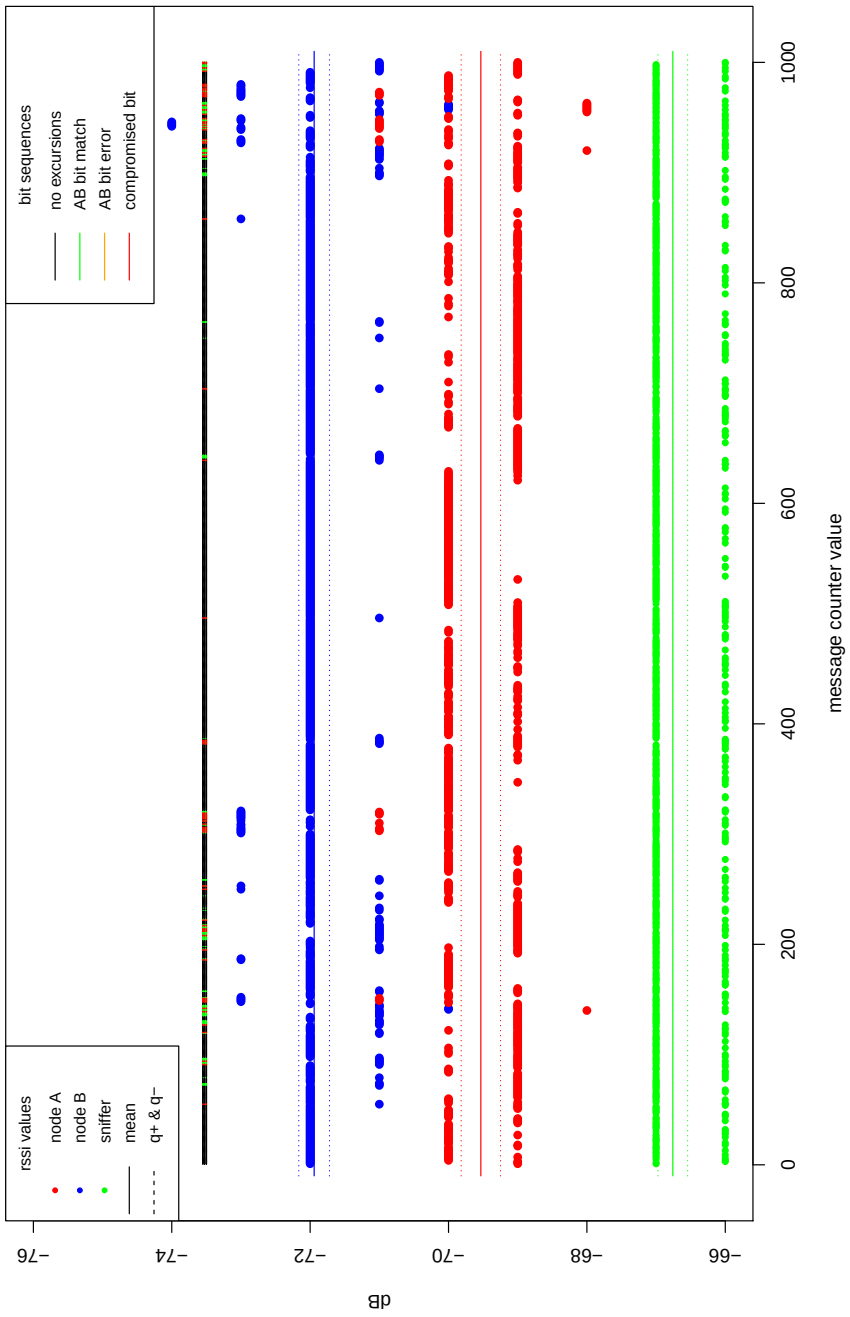


Figure 4.4: Graph with RSSI measurements for nodes A, B and sniffer. 161 excursions did match on nodes AB (161 secret shared bits produced) with 0 errors. 90 bits were compromised to the sniffer node.

Unfortunately, due to an error[4], we did not manage to collect any larger amount of data. Therefore there is not much we can conclude from these results. However, the results are promising, averages of generated bits and entropy estimates are slightly better than in other experiments. Values from this experiment, compared to the others, can be seen in table 4.1. Values might decrease with more experiments, but given the fact, that these results were produced on a network without any interference from outside, our approach with moving shielding is worth looking into.

The major issue regarding this source is the predictability of such data, because as we can see on graph of sample run B.2 there are clearly visible excursions that do repeat in somewhat regular fashion.

4.5.7 Speed (shared secret bit rate)

One run (exchange of 1000 messages) takes approximately 30 seconds to complete. This is under a condition that these two nodes are only one to use the wireless medium. Otherwise, the necessary time would be different.

Approximately 30 seconds to exchange values and then some minor time for calculations and exchange of two messages with identified excursions (which are negligible regarding the time). From these 1000 values we can obtain about 25 bits of entropy on average 4.5.5, therefore we can gather about 50 bits per minute.

This result is comparable to the original 1 bit/sec of Mathur et.al. [5]. However, we were capable of exchanging a smaller number of messages per minute than the original proposal (our rate 1000 msgs/30 sec is not comparable to 4000 msgs/sec achieved by Mathur et.al.), probably due to the nature of used devices 3.1.

4. Malfunction of automated application upload to the device, thus presence of a person was required to upload the application. However, a presence of a person was prohibited during the experiment itself, because it would affect the data. With experiment duration of 30 seconds, this was nearly impossible to manage.

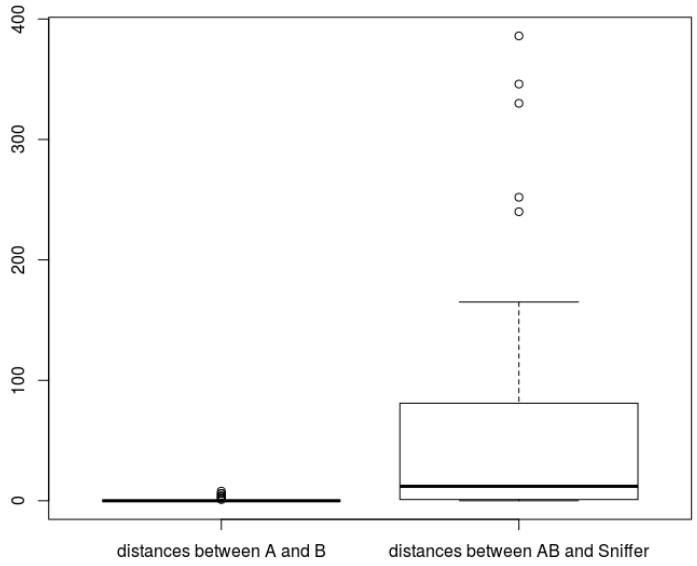


Figure 4.5: Boxplot of Hamming distances between generated bit sequences from nodes A, B, and sniffer node during movement of people 4.5.5

4.5.8 Possible errors

Question about reliability surely comes to mind. We have no direct guarantee, values and measurement on individual nodes are clearly not symmetrical, we only expect them to be highly correlated.

As we can see on the boxplots 4.5 4.6, the errors between nodes A and B are indeed scarce. Therefore there is a reasonable probability, that the nodes A and B will be able to establish the common key, using the obtained bit sequence from quantization algorithm 4.1.2.

4.6 Attacker results

One of the most important parts is how many bits were compromised to the sniffer node and potential attacker. As we can see 4.5 4.6, on average there is an hamming distance of 50 bits between output bit sequence for nodes A and B and bit sequence derived at sniffer node.

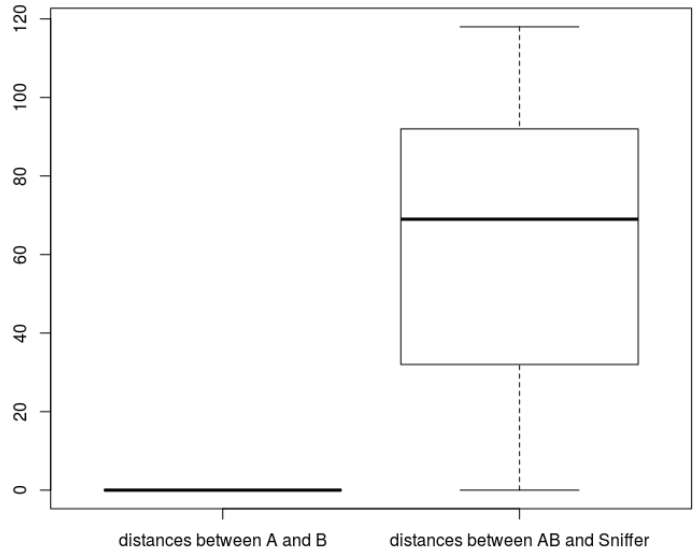


Figure 4.6: Boxplot of Hamming distances between generated bit sequences from nodes A, B, and sniffer node during movement of shielding 4.5.6

Of course, there are exceptions in both directions. Therefore we cannot guarantee that the derived bit sequence is secret in every case.

On the attacker side, guessing is also not an option. Although individual bit positions in a sequence are known, there is no way to estimate whether the identified excursion is over or below the threshold. Guessing which bits are not correct would be very time-consuming. While the one-bit change would mean 400 trials on average, two bits would mean about 400^2 trials. Complexity grows exponentially and the sniffer gets 50 bits wrong on average without any reasonable clue which bits are correct and which are not.

If we use generated bit sequence directly as a key, then there might be a risk of such sequence being compromised. However, there are much more use cases 4.7 and for some of them we do not care if the sequence has been compromised or not, only the majority of sequences has to remain secret.

4.7 Discussion and use cases

Our bit sequence still needs some post-processing to extract the randomness; the most simple one would be a hash function. Therefore, if we use a correct function, we can expect, that one bit would change cause a change in half of the bits in the resulting key thanks to the avalanche effect [29].

There are many use cases for this source of entropy. We can use this algorithm directly when two devices without any preshared data would like to communicate over a secured channel. Then we use our bit sequences directly (or after some postprocessing) as a key for encryption of communication.

Other different use could be improving secrecy of already secured network. Such algorithm usually utilises other neighbouring nodes to transfer additional key material [30]. With the use of our algorithm, we can generate additional key material directly between these two nodes, even if currently used key is compromised. This extra material then could be used to secure the direct connection again.

5 Education use

5.1 motivation for educational WSN network

The current state of the art WSN devices usually uses specialised hardware and software in order to achieve the best performance available. This, unfortunately, is not the ideal prerequisite. In fact, most of WSN devices have rather complicated setup and are quite challenging for novices [31].

Because of such discouragement, it is difficult to teach how to work with WSN's; few hours (at least) are usually required to explain the basics. It is reasonable for a research project or something similar. Hoewver, for a class exercise, this would turn out to be not the most effective use of time, if it would be achievable at all. And we have not yet mentioned more advanced topics in this area, such as common techniques for encryption or message authentication.

Issue of this nature can be solved in various ways, in the case of Edu-hoc, we decided to sacrifice performance 3.1; which is not that much important for a network with an educational purpose. On the other hand, using hardware that is really easy to comprehend and use is of a great benefit here. Also having less powerful, but relatively cheap devices (in the range of $30 rather than $100 or more) gives the opportunity to lend each the students one of the devices. They can try the basics on their own, and also use this device for interactions with the network.

5.2 Scenario approach (attack and repair)

In order to make learning more enjoyable experience and also to add some challenging part to the learning process, we decided to make Edu-hoc scenario based. Each scenario is composed of two distinct parts: first part in the role of an attacker (both enjoyable and educational) and the second part as a code reviewer or developer (primarily educational).

Solutions to individual scenarios are not part of this thesis. They should not be publicaly available, since Edu-hoc is being actively used in the lectures.

Attacker part of a scenario Students are presented with network application which has known, or easily detectable, vulnerability. Task is to take advantage of such vulnerability and exploit it using own nodes and carefully executed interactions with the network. How successful these efforts were can be easily evaluated by percentage difference from the expected traffic of the network or by presenting learned secret in the submission of the solution. Exact methods of evaluation are described in 5.3.

Reviewer part of a scenario An important part of the educational process is not only to find mistakes, but also to be able to correct them. The task in this part of the scenario is then to take the current source code of an application, which has been deployed on the network during the attacker part 5.2, correctly identify which mistakes were made and propose changes in the code, which would make the application secure against such kind of an attack. Exact evaluation methods are again described in 5.3.

5.2.1 1st scenario - Eavesdropping

The first scenario can be considered really easy and it is by design. Since this scenario is the first encounter with Edu-hoc, we opted for passive attacker approach, without any actual interaction with the network, just listening for any traffic.

Eavesdropping is also the first thing one would do if one would be about to attack some network because it does not compromise the presence of an attacker and intercepted messages may provide many useful pieces of information. This is another important reason why this task was selected as a first one, not only it is rather easy to do, but it provides vital information for the rest of the exercises (e.g. which nodes are present in the network, which frequency is used and what are the settings of the radio)

Scenario setting Each node in the network sends messages, after each message, there is a delay of 1000 ms. Therefore any receiving node has time to process the transmitted messages. All messages are sent as a broadcast. Nodes within the network do not receive nor

process the messages since it is not needed for successful scenario run. If this would be the case, transmission rates would have been updated accordingly and the scenario would be much more likely to fail on its own, because of single node malfunction. With current settings, the network can operate without any problems even is several of the nodes would fail.

5.2.2 2nd scenario - Black hole attack

This scenario presents more advanced concept - dynamic routing. The task is to attack the routing algorithm and diverse all traffic so that the central node does not receive any messages. This scenario requires active attacker, thus some basic interaction with the network.

Scenario setting The scenario is divided into two parts, first parts are shorter and are used to establish routes for this particular run of scenario. Without any attack or interference outside from the network, these routes should be always the same.

In the following part of scenario each node periodically sends messages to the parent node, therefore all messages are routed to the central node. This node then counts all received messages and the final outcome is the number of expected messages compared to the number of actual messages received.

5.2.3 3rd scenario - Sinkhole attack

Although very similar to the second scenario, this poses far greater challenge than the previous one. The goal is not only to divert the traffic but also modify it on the fly and send it to the original recipient.

The greatest issue here is related to the performance, since all messages from the entire network will be routed through single attacker node, therefore most of the messages are very likely to be dropped somewhere in the process, because single node is not capable of receiving and processing messages from every other node in the network, unless they do transmit on very slow rate.

Task for this scenario is to deliver a modified message to the central node before the original message is delivered; messages with the same

identifier but different content are discarded and only the one which was received the earliest is kept at the central node.

The reason why two very similar scenarios exist, is to present scenarios in a manner of slowly increasing difficulty. If we were to skip the second scenario 5.2.2, the task might be too complicated to some. This way, the required task in the third scenario 5.2.3 is only a minor extension to the second one. However, the achieved result is of a far greater impact.

scenario setting Runtime of this scenario is the same as for the previous one 5.2.2; initial phase of route establishment and after that messages are routed from nodes to the central node. The expected outcome is to divert ad modify as many messages as possible.

5.2.4 4th scenario - Jamming

Evey once a while attacker comes around a network, application or protocol which is designed with security in mind. There are no backdoors or weak spots in the design or implementation. Idealistic scenario, however, it is very likely that the attacker does not have resources to find or use existing vulnerabilities. For such attacker there is no real way how to exploit such application.

Nevertheless, it is very likely that such attacker would like to cause any kind of damage he is capable of. Most likely to make the application, protocol, or network unavailable; most likely by denial of service attack. For wireless communication it is quite easy to do, since the wireless medium, air to be specific, can be used by only one device at a time. If an attacker is able to utilise all the available slots for transmissions, then the legitimate user would not be able to transmit anything. Such technique is called jamming and it is quite easy to deploy.

scenario setting The setting of this scenario is simple, all nodes send a message via the fixed routing tree to the central node. the number of delivered messages is then counted. There is no phase of route establishment, only the message delivery phase. The assignment is to prevent delivery of as many messages as possible. Since there is no option to modify the routes etc. the only reasonable way is jamming.

5.3 Evaluation principle

Measuring the success can be done in various different ways, Eduhoc utilises few different methods to decide, whether the attack was successful and how big was the impact of an attack.

Evaluation for each scenario is a little bit different because there are different objectives. The task of first scenario 5.2.1 is to capture as many messages as possible, therefore the evaluation is done as a simple comparison between the set of captured messages and set of pre-computed messages that are known to be transmitted in the time period of scenario run. Comparing these two inputs line by line we get intersect of these two sets, that contains all successfully captured messages and is stripped of all the duplicates and possible fraud messages. A number of messages in the resulting is then compared to the expected number of messages in order to receive the final percentage evaluation, the higher the number, the better the result.

The second and the fourth scenario 5.2.2 5.2.4 have the same goal, only differ in the scenario settings, which can be omitted for the evaluation purposes. The common goal is to prevent as many messages as possible from delivery to the central node. The way it is done is rather simple, the central node counts every delivered message and the resulting number is compared with the expected amount of messages.

Unfortunately, this design of scenario lacks in the evaluation one important thing, the credit for the attack cannot be given to any participant, because we have no identifier involved. Messages only vanish, but we do not know, whether it was by joint work of multiple attackers, one really successful one or by a fault in the network itself. Therefore there is not much of use for such scenario in the courses, and because of this, modified versions of scenarios have been used 5.5.

The third scenario 5.2.3 however utilises message modification, which can be used for identification purposes. Therefore messages on the central node are not only counted, but alco compared to the expected ones and differences are noted[1].

1. the amount of computation on this level would limit the usability of the node, thus the node only sends received messages over the serial port to the server, where the actual evaluation is done

Since students are assigned a unique identifier to modify the messages with, we can easily distinguish between participants and evaluate how successful they were in this task.

The last option which can be utilised in evaluation is a submission of secret, which was received from interaction with the network, node either responds to received message with such secret or the secret has to be extracted from the communication in the network. Such secret can be a short word, number or something of similar nature and again such secret can be unique for each participant, which makes te evaluation an easy task.

5.4 Web interface and auto run

Edu-hoc was designed as a long-term exercise, thus single scenarios are expected to run for days, possibly week or longer. However, for some, it would be extremely impractical to repeat only once during the whole time because we would like to present fresh scenario to everyone involved, and also we would like to reset the scenario once a while. This will ensure correct initial settings and if we set the intervals right, then everyone will be able to test out several solutions within a reasonably short timeframe.

To achieve such kind of behaviour, we can utilise various approaches, the easiest one being set of bash scripts and setting the cron service accordingly. The other option might be modified JeeTool, which would be running continuously and in set intervals would trigger the corresponding action.

The last option is the most complicated one but offers the most

5.5 PA197 use and results

6 Future and related work

6.1 RSSI

6.2 Edu-hoc

7 Summary

Bibliography

[1] Yanjun Li, Zhi Wang, and Yeqiong Song. "Wireless sensor network design for wildfire monitoring". In: *6th World Congress on Intelligent Control and Automation (WCICA 2006), Dalian (China)*. 2006.

[2] Zygmunt J Haas and Tara Small. "A new networking model for biological applications of ad hoc sensor networks". In: *IEEE/ACM Transactions on Networking* 14.1 (2006), pp. 27–40.

[3] Frank Stajano, Dan Cvrcek, and Matt Lewis. "Steel, cast iron and concrete: Security engineering for real world wireless sensor networks". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2008, pp. 460–478.

[4] *Arduino*. https://www.arduino.cc/. 2016.

[5] Suhas Mathur et al. "Radio-telepathy: Extracting a Secret Key from an Unauthenticated Wireless Channel". In: *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*. MobiCom '08. San Francisco, California, USA: ACM, 2008, pp. 128–139. ISBN: 978-1-60558-096-8. DOI: 10.1145/1409944.1409960. URL: http://doi.acm.org/10.1145/1409944.1409960.

[6] Luis Sanchez et al. "SmartSantander: IoT experimentation over a smart city testbed". In: *Computer Networks* 61 (2014), pp. 217–238.

[7] ZigBee Alliance et al. *Zigbee specification*. 2006.

[8] Vashek Matyáš et al. "WSNProtectLayer: Security Middleware for Wireless Sensor Networks". In: *Securing Cyber-Physical Systems*. CRC Press, Sept. 2015, pp. 119–162. ISBN: 978-1-4987-0098-6. DOI: doi:10.1201/b19311-6. URL: http://dx.doi.org/10.1201/b19311-6.

[9] JeeLabs. *JeeLink product page*. http://jeelabs.net/projects/hardware/wiki/JeeLink. 2016.

[10] JeeLabs. *JeeNode product page*. http://jeelabs.net/projects/hardware/wiki/JeeNode_USB. 2016.

[11] *Arduino Mini*. https://www.arduino.cc/en/Main/ArduinoBoardMini. 2016.

[12] HOPE RF. *RF12B Datasheet*. https://www.sparkfun.com/datasheets/Wireless/General/RF12B-IC.pdf. 2016.

[13] JeeLabs. *JeeLib library*. http://jeelabs.net/projects/jeelib/wiki. 2016.

[14] *Arduino Introduction*. https://www.arduino.cc/en/guide/introduction. 2016.

[15] L. Němec. *JeeTool*. https://github.com/crocs-muni/JeeTool. 2016.

[16] *Arduino-Makefile*. https://github.com/weareleka/Arduino-Makefile. 2016.

[17] D. Klinec. *WSNmotelist*. https://github.com/ph4r05/WSNmotelist. 2014.

[18] W. Hedgecock. *jSerialComm library*. http://fazecast.github.io/jSerialComm/. 2016.

[19] *RXTX library*. http://rxtx.qbang.org. 2011.

[20] *Understanding RSSI*. http://www.metageek.com/training/resources/understanding-rssi.html. 2016.

[21] Suman Jana et al. "On the Effectiveness of Secret Key Extraction from Wireless Signal Strength in Real Environments". In: *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*. MobiCom '09. Beijing, China: ACM, 2009, pp. 321–332. ISBN: 978-1-60558-702-8. DOI: 10.1145/1614320.1614356. URL: http://doi.acm.org/10.1145/1614320.1614356.

[22] B. Azimi-Sadjadi et al. "Secret Communication over Fading Channels". In: *Securing Wireless Communications at the Physical Layer*. Ed. by Ruoheng Liu and Wade Trappe. Boston, MA: Springer US, 2010, pp. 281–309. ISBN: 978-1-4419-1385-2. DOI: 10.1007/978-1-4419-1385-2_12. URL: http://dx.doi.org/10.1007/978-1-4419-1385-2_12.

[23] S. Goel and R. Negi. "Guaranteeing Secrecy using Artificial Noise". In: *IEEE Transactions on Wireless Communications* 7.6 (June 2008), pp. 2180–2189. ISSN: 1536-1276. DOI: 10.1109/TWC.2008.060848.

[24] Xiang He and Aylin Yener. "Cooperative Jamming: The Tale of Friendly Interference for Secrecy". In: *Securing Wireless Communications at the Physical Layer*. Ed. by Ruoheng Liu and Wade Trappe. Boston, MA: Springer US, 2010, pp. 65–88. ISBN: 978-1-4419-1385-2. DOI: 10.1007/978-1-4419-1385-2_4. URL: http://dx.doi.org/10.1007/978-1-4419-1385-2_4.

[25] Aaron D Wyner. "The wire-tap channel". In: *The bell system technical journal* 54.8 (1975), pp. 1355–1387.

[26] Charles-Henri Hallard. *Improved RFM12B with accurate RSSI reading Library*. https://hallard.me/rfm12b_arssi-library/. 2014.

[27] Andrew Rukhin et al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Tech. rep. DTIC Document, 2001.

[28] Elaine Barker and John Kelsey. "Recommendation for the entropy sources used for random bit generation". In: *Draft NIST Special Publication* (2012).

[29] A. F. Webster and S. E. Tavares. "On the Design of S-Boxes". In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 523–534. ISBN: 978-3-540-39799-1. DOI: 10.1007/3-540-39799-X_41. URL: http://dx.doi.org/10.1007/3-540-39799-X_41.

[30] Radim Ostadal, Petr Svenda, and Vashek Matyas. "On Secrecy Amplification Protocols". In: *The 9th WISTP International Conference on Information Security Theory and Practice (WISTP–2015), LNCS 9311*. Springer, 2015, pp. 3–19. DOI: 10.1007/978-3-319-24018-31.

[31] Fei Hu and Xiaojun Cao. *Wireless sensor networks: principles and practice*. CRC Press, 2010, pp. 225–226.

A Testbed map and node placement

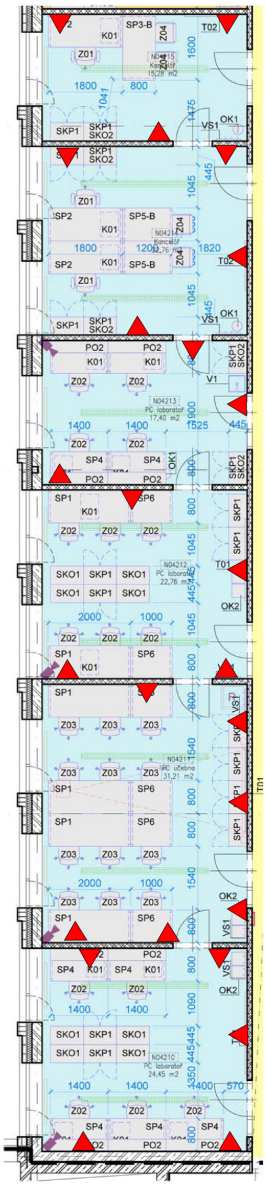


Figure A.1: Map of testbed with placement of individual nodes

id	x	y
1	1498	0
2	1977	207
3	1147	90
4	1147	447
5	1134	0
6	1134	310
7	949	0
8	506	65
9	506	475
10	1718	0
11	1964	242
12	1616	462
13	1603	490
14	2434	0
15	2434	500
16	2214	0
17	2447	295
18	2753	70
19	2753	320
20	333	0
21	493	390
22	493	70
23	155	0
24	0	210

Table A.1: Node IDs and placement coordinates
(Coordinates begin from top right corner of map)

B RSSI graphs

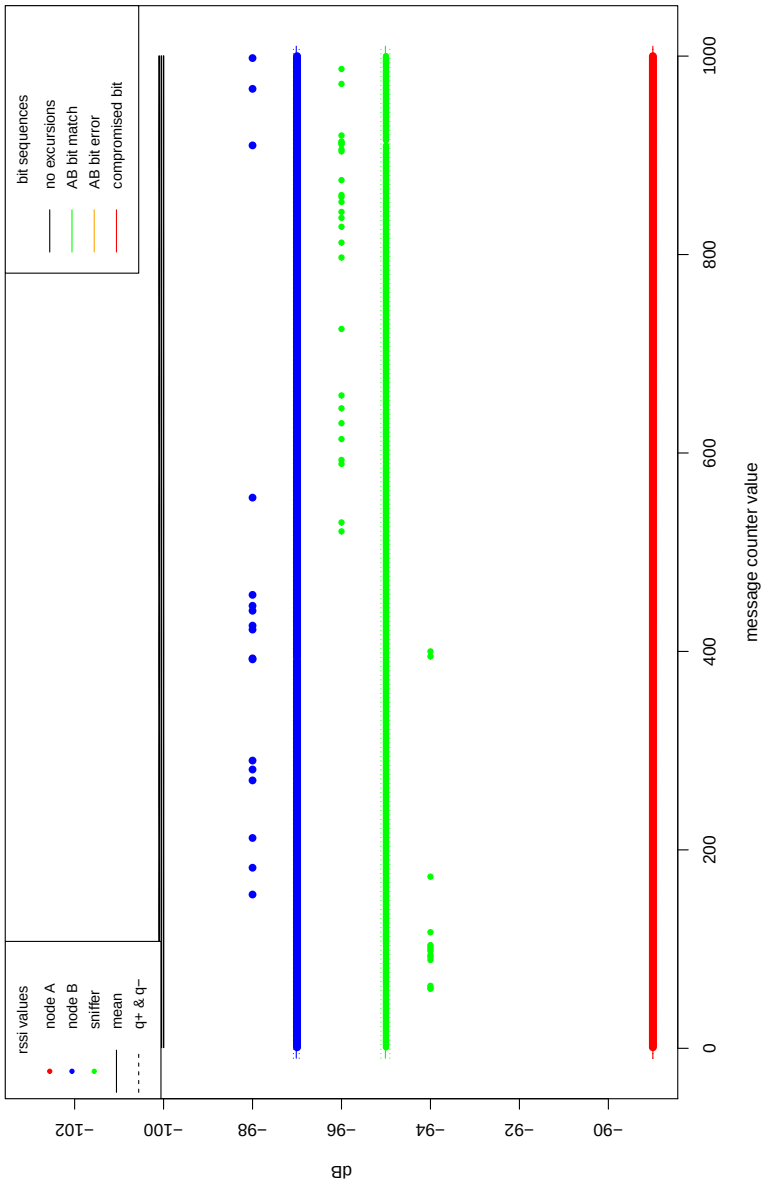


Figure B.1: sd = 0

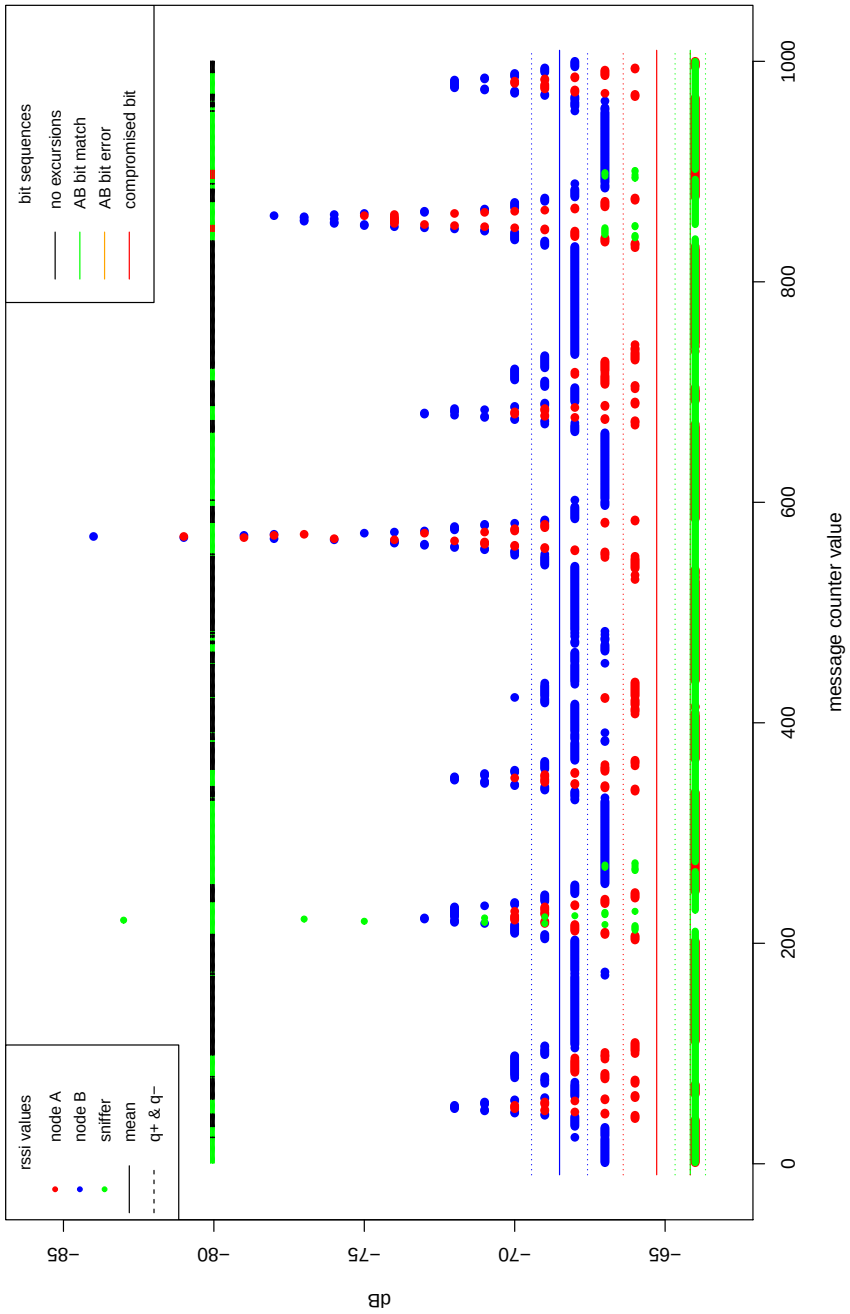


Figure B.2: spikes

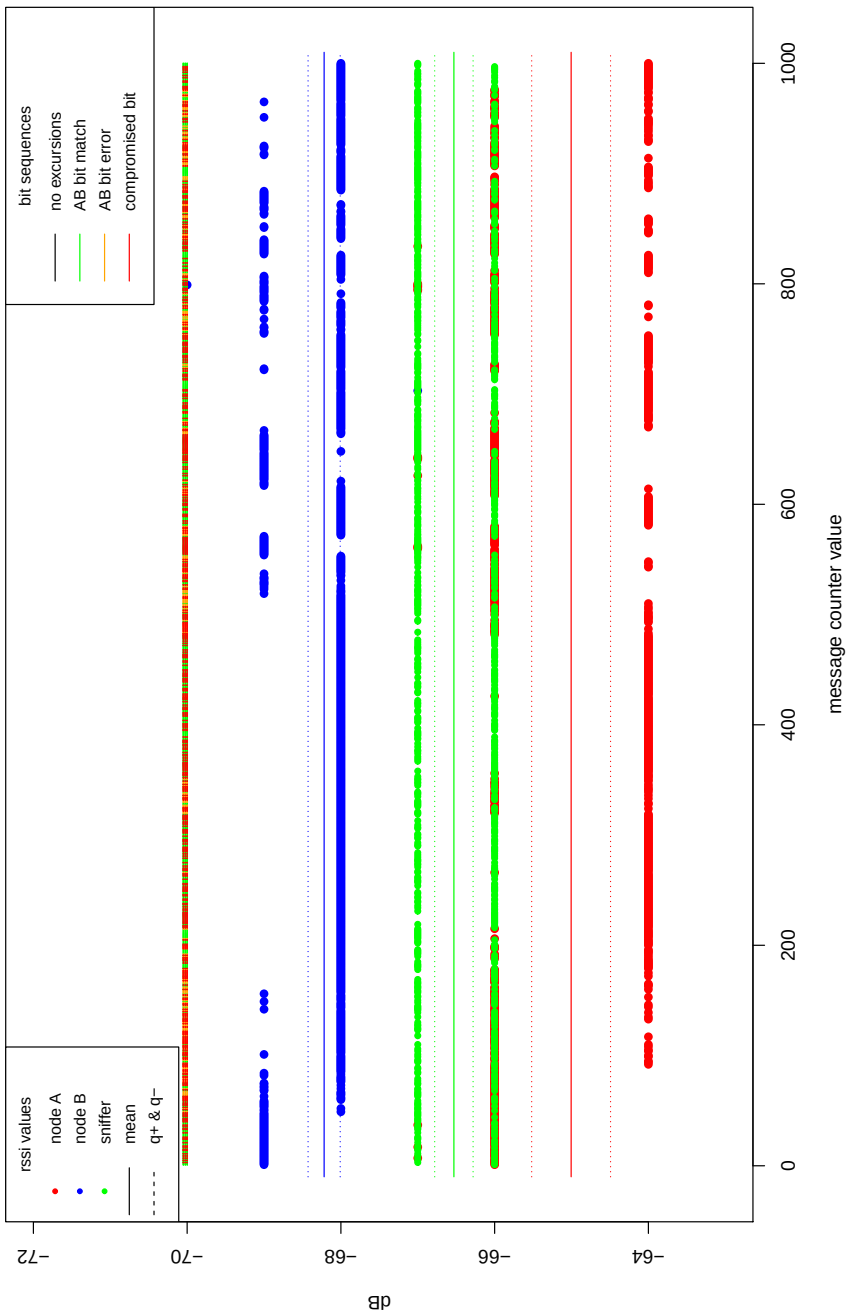


Figure B.3: 1000 bits output