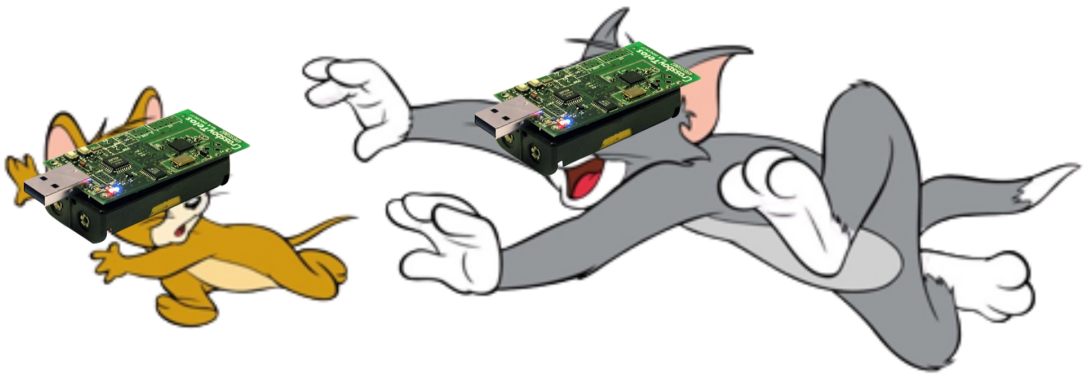
Jamming & Anti-Jamming In WSNs

by

Daniel Christopher Biørrith, 201909298

Lukas Koch Vindbjerg, 201906015

Malthe Faurschou Tøttrup, 201907882



Aarhus University

Department of Electrical and Computer Engineering

December 19, 2022

Contents

1 Introduction

Wireless Sensor Networks (WSNs) are a critical topic, as they play an important role in numerous safety-critical applications. As WSNs are both low-cost and low-power with the ability to collect, process and transmit data, their uses span various fields such as military, healthcare, agriculture, and environmental monitoring.
Despite the numerous advantages of WSNs, they are vulnerable to various security threats, one of which is jamming attacks. Jamming is a denial of service (DoS) attack referring to disruption or interference of transmission between the sensor nodes in order to negatively affect the performance or reliability of the network.
Jamming attacks come in many different forms and can generally be classified into two categories; passive and active jamming with many variations of each.
As a countermeasure, it is essential to design and develop efficient anti-jamming techniques to increase the robustness against malicious attacks on the network. Depending on the jamming attack there are a variety of optimal anti-jamming techniques both as proactive and reactive methods. It is therefore crucial to develop a great understanding of both jamming and anti-jamming in order to successfully develop implementations in the field.
In this project, we will focus on the impact of jamming attacks in WSNs. We will set up a network and explore different jamming attacks and anti-jamming defenses and discuss the effectiveness of these. We will monitor the successful packages sent throughout the network as the criterion of success for both attack and defense with considerations for resource use.

2 Theory

A jamming attack is a DoS attack that uses a transceiver to interrupt the communication between nodes in the network. It is not interested in collecting the data, and will therefore generally consist of emitting radio frequency (RF) noise to the shared medium, in order to deaf out the nearby nodes of the network. Therefore, jamming typically works on the physical layer in the open system interconnection (OSI) model, which is the lowest level layer.
An important parameter for any jamming effect measurement is the received signal strength indicator (RSSI). RSSI provides information about the signal that can be used the determine whether the frequency is under attack. In general, any RSSI measurement above $-80dBm$ is considered a poor signal strength [1], as that would mean much noise is present on the channel.

A jamming attack that disrupts the signal strength can negatively affect the network on multiple parameters. These include reducing network coverage, disrupting communication, or leading to potential loss of data. This project will focus on the disruption of communication and data loss, and not on how jamming will affect the network topology and message routing.

2.1 Types of jamming

There exists a large variety of jamming attacks at many different complexity levels used for different complexities of networks[2]. In this project, we will be focusing on the following types:

- **Constant jammer**: A constant jammer selects a channel and starts sending out a continuous stream of radio signals. The jammer disregards the rules of the MAC protocol and aims to have its messages collide with those of other nodes trying to send messages on the channel.

- **Random jammer**: Similar to the constant jammer, a random jammer selects a channel to jam. The difference is that the random jammer will intermittently send a continuous stream of radio signals. This generally makes it harder to detect compared to the constant

jammer, since packages between nodes in the network will occasionally come through, though it does not block the network as effectively.

- **Channel hopping jammer**: A channel hopping jammer listens for traffic on a channel for a period of time before it decides to take action. If it detects activity on a channel, it will start jamming. If there is no activity it will hop to another channel and repeat the process.

The random and constant jammers are more concerned with *how* the jamming takes place, whereas the channel hopping jammer focuses on *when* to jam. Hence, we will be combining the channel hopping jammer with both the random and constant jamming methods.

2.2 Anti-Jamming

Anti-jamming can be composed of two disciplines: jamming detection and jamming countermeasures. First, the WSN needs to detect that a jamming attack is happening and potentially what kind of jamming attack. Then, the WSN needs to make a decision on how to react to the attack [3]. In this project, we will be focusing on the following detection techniques and countermeasure:

- **Message shortage**: Here, the number of received, non-corrupt messages is compared to the number of expected messages. If the number of lost messages exceeds a certain threshold, it could be classified as jamming.

- **Signal strength monitoring**: One way to detect jamming is to monitor the strength of the received signal. If the signal strength suddenly drops or becomes distorted, it could indicate that the channel is being jammed.

- **Channel hopping**: When a jamming attack has been identified the sender changes the channel of the transmitted signal, making it difficult for a jammer to follow. In this method, it is paramount that the changing of channels follows an algorithm known to both the sender and receiver nodes, in order to stay on the same channel.

3 Methods

For implementing and testing appropriate jamming and anti-jamming methods in WSNs, we have used the Crossbow's TelosB mote (TPR2420) as the hardware for both our network and jammers. It offers features such as programming and data collection via a USB connection and IEEE 802.15.4 CC2420 RF Transceiver. The implementation was also run in a simulation using Cooja. Tests were performed both virtually and physically since differences in the behavior provided different results for analyzing the implementation. The physical world is more relevant for the methodologies, however, as the environment is not consistent and difficult to replicate exactly, it becomes less reliable when comparing different implementations. The Cooja simulation was used to ensure a consistent environment. Unfortunately, this had its own limitations as there would be no background RF noise in the simulation, meaning the robustness of the setup would not be challenged. It was therefore decided that both tests would be used for a more thorough analysis of the implementation.

To implement both we have used Contiki-NG operating system as this is a useful tool for IoT and networking[4]. Contiki-NG has a lot of integrated networking functionalities. These functionalities controlled the fundamental protocols which enabled us to develop at a higher abstraction level. This allowed us to focus more on the relevant theories and methodologies of jamming and anti-jamming.

To set up the network we used Contiki-NG's `NETSTACK` module, which is a low-layer network stack library for network nodes. `NETSTACK` was used as it allowed for simple implementation, alongside lower-level modifications for formatting the jammer.

3.1 Network structure

The network consists of three transceiver motes and one jammer mote. Each mote is connected to a predefined channel. The transceiver motes interact by being initialized with knowledge of the linked layer addresses of all other transceiver motes in the network. Each transceiver mote will then send an incriminating counter value to all other motes in the network with an interval of 2 seconds, functioning as a heartbeat. This uses `NETSTACK.output()` which handles the networking protocols. Each mote keeps track of the messages received from all other motes in the network. The jammer is initialized without any knowledge of the network.

3.2 Jamming

In this project, we implement a constant jammer and a random jammer, both of which are channel hopping. Since we are using Contiki-NG we need to manipulate some of the standard networking functionally. While jamming, the jammer turns off the CCA (channel clear assessment) protocol, which is a MAC protocol that tries to counteract multiple nodes sending packets at the same time in a WSN. Turning this off will make the jammer ignore the fact that other nodes are trying to transmit packets. However, before the jammer commences the attack, the jammer uses the CCA to listen for activity on a channel. To determine whether a packet is being transmitted, the CCA threshold is set to a minimum of $70dBm$ RSSI level. To check the CCA value, the jammer uses the `NETSTACK_RADIO.channel_clear()` function, which returns 0 if the RSSI was above the threshold, or 1 if it was below while sampling, indicating whether the channel is busy or not. If the channel is busy at any point during the listening period, as shown in Fig. 1, the jammer will assume that the channel is being used and start a jamming attack. The constant jamming method will send 10000 packets as fast as possible on the channel, before going back to listening for activity. The random jamming method will send a random amount of packets between $0 - 512$ and then wait for a random amount of time between $0 - 3$ seconds. This is repeated ten times before going back to listening, as shown in Fig. 2. The jammer will cycle through and listen to channels $11 - 25$.

3.3 Anti-jamming

The approach for countering a jamming attack can be divided into two critical tasks. Detecting that a jamming attack is happening and then the corresponding countermeasures.

3.4 Jamming detection

We have implemented two functionalities for detecting a jamming attack as we wanted it to be able to be robust against both the constant jammer and the random jammer.

The first way of detecting a jamming attack is by looking at the package loss from each neighbour in the network. When a mote A sends out a package every two seconds, it increments a `loss_counter` for each neighbouring node. When A receives a message from one of its neighbours it resets the counter for the corresponding node. If at any point mote A hits a threshold of 10 packages lost in a row in total from its two neighbours it identifies this as a jamming attack.

The second detection functionality looks at the noise on the channel and evaluates if there are any signs of jamming attacks. If the signal strength drops, this might be an indication

```
bool check_channel_activity(){
    int channel_activity = 0;
    NETSTACK_RADIO.set_value(RADIO_PARAM_CCA_THRESHOLD, 70);
    for (int i = 0; i < 10000; i++){
        //returns 0 if channel is busy and 1 if its clear
        channel_activity += NETSTACK_RADIO.channel_clear();
    }
    printf("channel activity result: %d \n", channel_activity);

    if(channel_activity != 10000){
        printf("Activity found on channel %d! \n", current_channel);
        return true; //start jamming attack
    }
    else {
        printf("No activity found on channel %d. \n", current_channel);
        return false; //keep listening for activity
    }
}
```

Figure 1: Check channel activity function for jammers. The jammer samples the CCA of the channel for 10000 iterations. If there is a packet being transmitted at any point this function will return true, which results in the jammer launching an attack.

```
while(k < 10){
            PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

            send_iterations = abs(rand() % (int)CLOCK_SECOND*4);
            wait_time = abs(rand() % (int)CLOCK_SECOND*3);

            while(random_interval <= send_iterations){

                cc2420_driver.send((void*)&jpacket, JAMMER_PACKET_LEN);

                random_interval++;
            }
            random_interval = 0;
            etimer_set(&et, wait_time);
            etimer_reset(&et);
            k++;
        }
```

Figure 2: The procedure for launching a random jamming attack when activity is found. The jammer bursts packets with a random amount of iterations and then proceeds to wait for a random amount of time.

of an attack. However, the RSSI of a channel will almost never be consistent and sometimes an object might pass between two nodes temporarily decreasing the signal strength. These measures should not be recognized as jamming attacks. Furthermore, changing conditions such as the humidity or other environmental factors might steadily decrease the signal strength so setting a hard limit for the required signal strength is not a sufficient measurement for jamming detection.

The approach for detecting a jamming attack will therefore involve evaluating the moving average of the RSSI. More specifically we are interested in a short and a long moving average as shown as a sketch in Fig. 3.

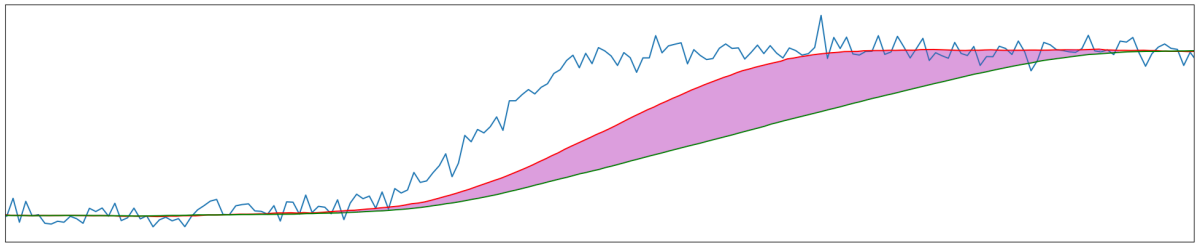


Figure 3: Moving average of signal with increasing noise

A jamming attack would involve a sharp increase in the baseline noise on the channel like what can be seen in Fig. 3. We, therefore, keep track of the moving average of the signal to detect such an increase. We use the two moving averages to calculate how drastic this increase is, a short moving average and a long moving average. If the increasing noise happens quickly the short moving average will follow the true value much tighter than the long moving average as illustrated by the purple shaded area in the figure. If the difference between the two moving averages becomes large enough (determined by a scaling value), it will be classified as jamming. The concept of how all of this is evaluated can be seen in algorithm 1.

Algorithm 1 Detect sudden rise in RSSI baseline

Require: Array X of size n_1, array Y of size n_2, and scaling value p. $n_1 < n_2$
  Fill X and Y with $RSSI$ values
  while True **do**
    Sleep for time t
    Get $RSSI$ value
    Insert $RSSI$ at position of oldest value found in X and Y.
    Calculate average a_X of X.
    Calculate average a_Y of Y.
    if $(p \cdot a_X > a_Y)$ **then**
      Switch channel
    end if
  end while

The short moving average evaluates the previous 20 measurements, while the long moving average evaluates the past 60 measurements. The threshold for the difference was set at 10%. The advantage of this approach is that it covers all the previously mentioned issues in uncertainty. A sudden unexpected spike in interference - such as an object passing in between - would not be detected as jamming. Likewise, a slow and steady increase such as changing environment would not meet the threshold and would therefore not be detected as jamming either. This approach will therefore be effective at detecting a sharp increase in the baseline of the noise.

3.5 Jamming countermeasures

In this project, we implement a variation of the *channel hopping* jamming countermeasure. To do this we create of list of channels to cycle through, known to all of the motes in the network. The channels in the list are 12, 17, 18, and 19. Whenever a jamming attack is discovered by a mote, the mote will hop to the next channel in the list. However, with this method, there is a high chance that the motes will get out of sync if they don't discover the jamming attack simultaneously. To circumvent this issue we extended the packet loss jamming detection described in section 3.4 to work as a catch-up mechanic. Instead of using a 5-packet loss threshold for any of the neighbours, we check if the sum of lost packets for all neighbours

is above a threshold of 10 packets. This means that if two motes are on the same channel while the other is on its own, the two that are together will change the channel less frequently, allowing the sole mote to catch up to them. This catch-up mechanism is illustrated in Fig. 4, where the sole mote jumps every 5 seconds, allowing it to catch up to the other two. However, the downside to this method is that it fails if all motes at some point are scattered on three different channels. Then they will hop with the same time intervals, never meet again, and be stuck in limbo.

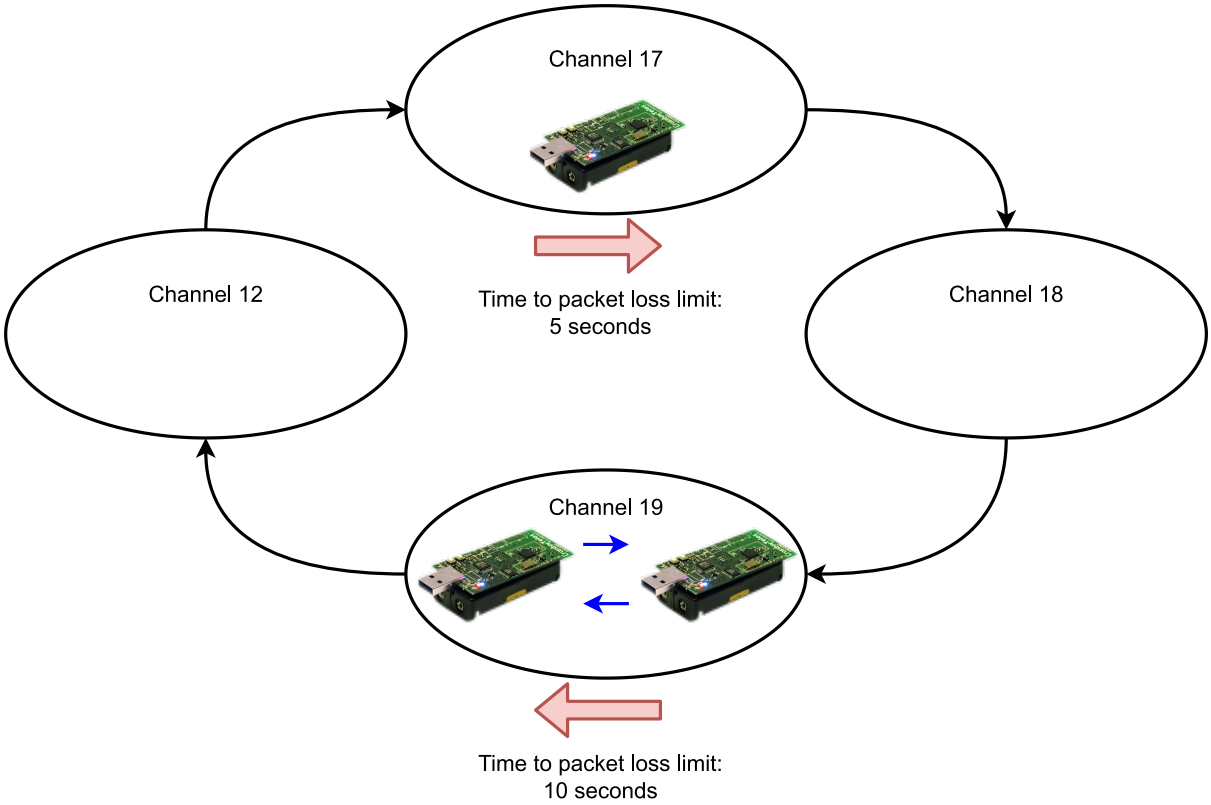


Figure 4: Channel cycle of transceiver motes, with time to hop indications.

3.6 Experimental setup

In this project, we conduct two experiments: one in simulation and one in the real world. In both experiments, the scenario will be largely the same, in order to compare the results of the two experiments afterward. In both scenarios, we have three motes who are communicating and one jammer as described in section 3.1. In the experiments, we decided to use the combination of channel hopping random jammer, as this should be the hardest type to detect for our network. With the two experiments, we seek to answer the following questions:

- Can the jammer reliably detect the communicating motes on a channel?
- Is the network able to reliably detect the presence of the jammer?
- Are the motes able to consistently meet again on a common channel? Or will they get stuck in limbo?
- Does the method work in noisy environments, or is it only applicable in low or noise-free settings?

Simulation setup

The simulation experiment setup is shown in Fig. 5. In this setup Node 1 and 3 are in close proximity to the jammer, compared to Node 2.

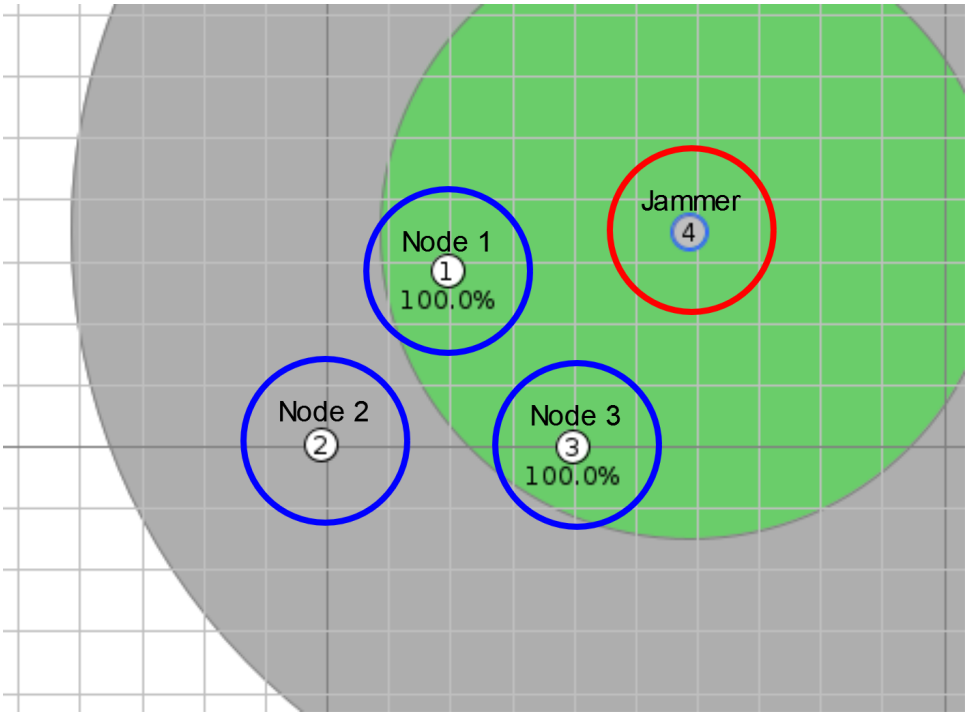


Figure 5: Simulation setup in Cooja.

Real world setup

The real-world experiment setup is shown in Fig. 6. The setup is very similar to the simulation setup, however, in this case, all of the motes have approximately the same distance to the jammer.

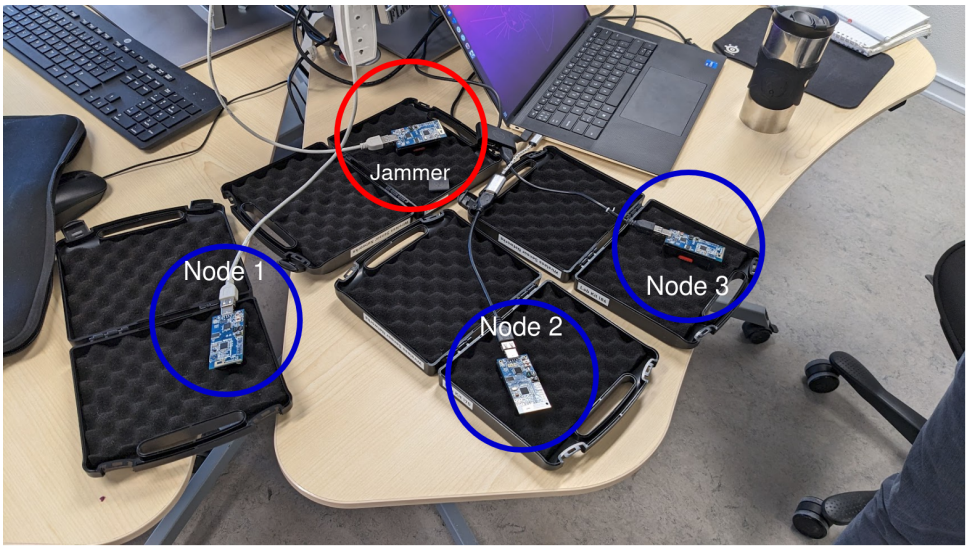


Figure 6: Real world setup.

4 Results

The following section will present the results obtained when testing the setups described in section 3.4. The results will be split up into two sections, results from the simulation and results from the test carried out on the motes in real life. The results will be presented in the form of figures and tables.
As a baseline for the results, if there are no anti-jamming methods deployed, the jammers will successfully block all messages in testing while jamming is active.

4.1 Simulation results

A snapshot of the result is shown in Fig. 7. It shows the jammer finding activity on channel 12, prompting it to start jamming. Node 1 and 3 detect said jamming by their RSSI value, prompting them to switch channels, while Node 2, not in close proximity to the jammer, stays on the channel until it changes because of message shortage.

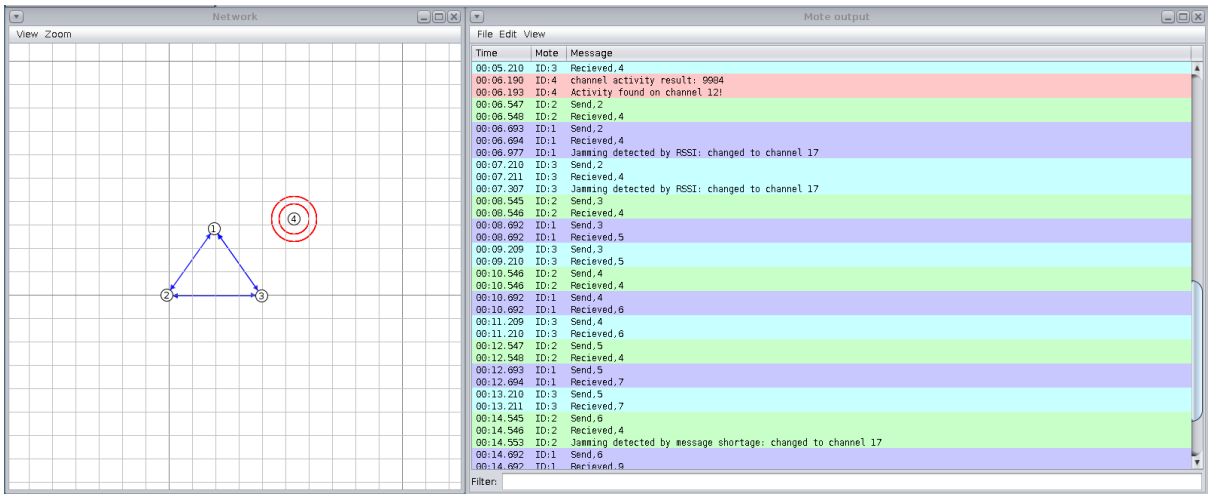


Figure 7: Snapshot of a result in the Cooja simulation.

Fig. 8, 9, and 10 shows the RSSI and moving averages RSSI of Node 1, Node 2, and Node 3, respectively. On all figures, it's clear to see that no noise is present.

From Fig. 8 it's clear to see the spikes in RSSI values when the jammer turns on. For example, there are big spikes in the RSSI value around sample 75 (multiple spikes as the jammer is a random jammer). This prompts an increase in the moving average values, where the difference between the two is greater than the 10% scaling value p. Said difference triggers a change in channel, by which the RSSI value goes back to -100. The second time the jammer finds the nodes, around sample 400, the immediate difference is not significant enough to trigger a channel switch. However, in time, around sample 450, it does trigger a change in channel.

The fact that Node 2 is not in close proximity to the jammer also shows in Fig. 9, as the RSSI value increases much less from the jammer. This means that Node 2 never switches channels due to the RSSI value. Instead, employs the catch-up mechanic and switches due to message shortage, which would be around sample 200 and at the very end of the measurement.

The results of Node 3, seen in figure 10, looks much like the ones one Node 1, with big spikes around sample 75 and 400, triggering a change in channel. However, it's also evident that sudden spikes occur outside of the jamming periods. An explanation of this could be that the

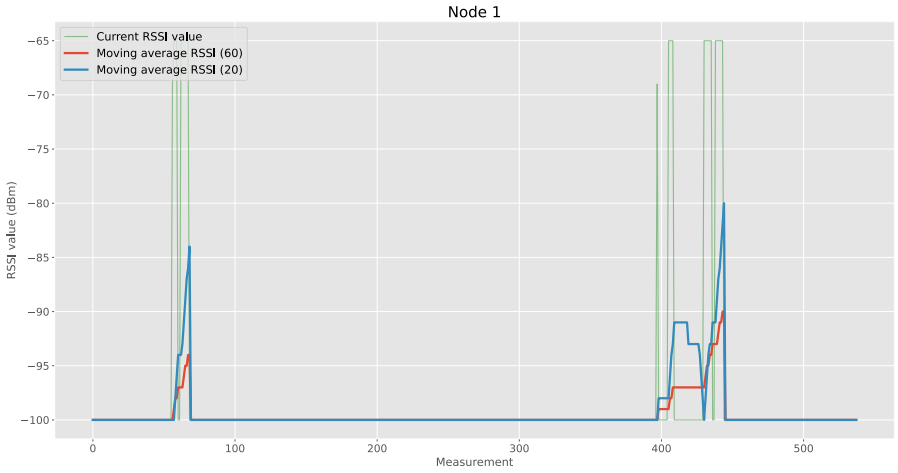


Figure 8: Results of simulation test of Node 1. The jammer is active around samples 75 and 400, which shows in the spike in RSSI.

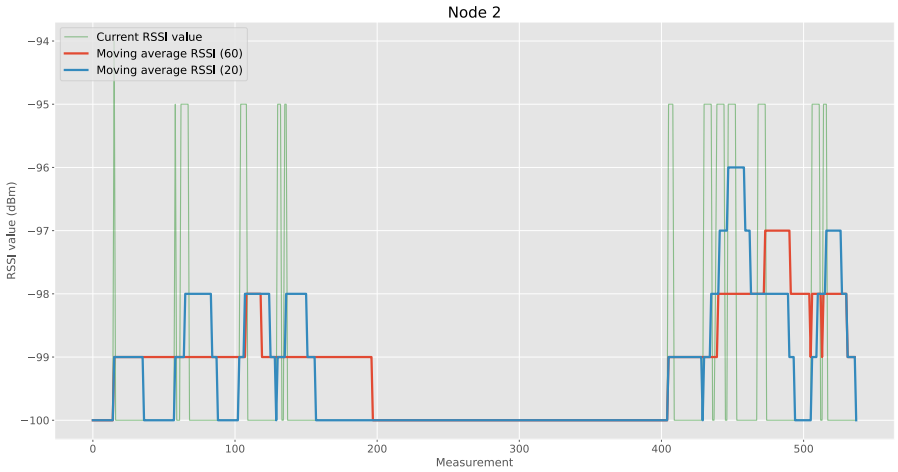


Figure 9: Results of simulation test of Node 2. Clear to see that it is far from the jammer by looking at the RSSI values, resulting in RSSI not being a factor in switching channels.

measure is taken at the exact same time as messages are sent, which results in an RSSI value different from -100. What further implies this, is that the spikes seem periodic. A measurement every 0.1 seconds and a message every two seconds, if in sync, will meet every time a message is sent. Finally, the reason such a sync might occur is because of the difference in the exact time the nodes change channels, which is why they aren't in sync before or after the switch.

Table 1 shows the expected number of messages received (total number of messages sent to the node), compared to the number of messages received. It shows that Node 1 and Node 3 received over 90% of messages, while Node 2 received around 70% of messages. This is to be expected, as Node 2 switches channel slower, resulting in a larger drop in messages.

4.2 Real world results

For the test carried out on the motes in real life, the figures for all three nodes look alike. Because of this, we only included Node 2's figure, Fig. 11. Results of Node 1 and Node 3 can be found in Appendix A.

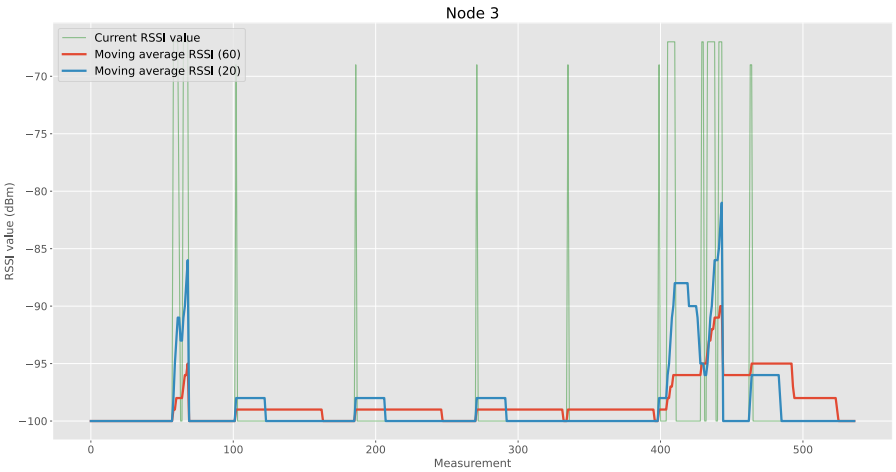


Figure 10: Results of simulation test of Node 3. The jammer is active around samples 75 and 400, which shows in the spike in RSSI.

Node 1		**Node 2**		**Node 3**	
Expected	Received	Expected	Received	Expected	Receeved
56	51	56	40	56	51
91% received		71% received		91% received	

Table 1: Results of simulation testing, the expected number of messages received, compared to the actual number of messages received.

Fig. 11 shows that the RSSI, unlike in the simulation test, has noise. This shows in the mean average values in the form of constant, small changes, though none great enough to prompt a channel switch. When the jammer starts around sample 75, the difference becomes great enough to trigger a channel change. Once again, at around sample 230, the jammer starts, triggering a channel change. The result shows that the anti-jamming works as intended, as it does not get affected by the noise at the level present in testing, while they still detect the jamming.
Table 2 shows the expected number of messages received (total number of messages sent to the node), compared to the number of messages received. It shows all three nodes received over 90% of messages, which is a very positive result, showing great resistance against our jamming.

Node 1		**Node 2**		**Node 3**	
Expected	Received	Expected	Received	Expected	Received
48	46	48	46	48	48
95% received		95% received		100% received	

Table 2: Results of real-life testing, expected messages to receive, compared to the actual number of received messages.

Something to notice regarding the jammer is that it might find activity on other channels than the one the motes are operating on, due to other devices possibly operating on the channel. For example, our jammer always found activity on channel 26 during testing.

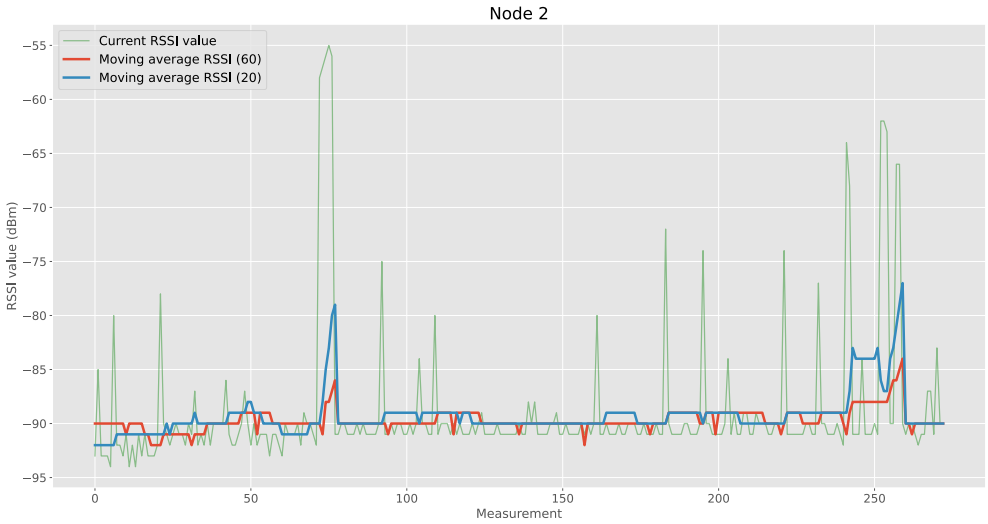


Figure 11: Results of real-life test of Node 2

5 Discussion

From the results, we can confidently say that our anti-jamming, against the specific jamming type we implemented, works as intended. Both the physical and simulated tests showed that the network was successful at both detecting and counteracting the jamming attacks. The catch-up functionality of tracking the packages during channel hops ensured that the motes of the network were eventually able to re-synchronize on the same channel.
The anti-jamming method designed would be easily scalable, as the catch-up mechanic would be scalable unless the case where two equal-size groups would be on each channel, where they would never catch each other. To avoid this, we recommend the number of motes to be a prime number, as this will ensure the catch-up will always work, as it would then be impossible to have all groups of the same size. Additionally, the RSSI anti-jamming does not depend on the number of motes in the network, therefore it does not affect the scalability.

It is worth considering, however, that since both the jamming and anti-jamming were implemented by us, we knew exactly what the advantages and flaws of each were. The promising results of testing might therefore be somewhat biased where using e.g. an unknown jammer might not prove as effective.

Likewise, a point to consider is if the jammer knew of the specific anti-jamming methods deployed. The jammer could then gradually increase the jamming, as that would also gradually increase the mean average, resulting in no change of channel from the RSSI value. Introducing a threshold value for the mean average RSSI might be a way to counteract this, though the jammer could keep the value just above the threshold. Additionally, the jammer could monitor the channel hopping pattern and simply follow the motes constantly.

Another point to consider is the energy consumption of the anti-jamming. A WSN depends greatly on being frugal with energy, measuring the RSSI, and calculating the average of 60 and 20 values every 0.1 seconds resulting in over 800 operations every second. Though we did not carry out testing of energy consumption, it is highly possible that the energy consumption is too great to deploy in a real WSN. Attempts to improve the algorithm have been made, where instead of summing up all 60 values in each iteration, we keep one, great summation to add

and subtract the oldest and newest RSSI value. However, we encountered some weird Contiki addition error we could not overcome, by which we summed up all values each iteration.
Our anti-jamming is a reactive approach, yet a possible anti-jamming method we did not explore could be a more proactive approach. If we instead opted to switch channels after every message sent, it could both be harder for the jammer to find the given channel, as well as possibly more energy efficient, depending on the cost of switching channels, compared to the cost of constant RSSI measures and calculations. However, this would introduce more problems with synchronization. There is no guarantee that the motes run on the exact same clock cycle, the motes could possibly go out of sync, which is not fixable without a hard reset.

6 Conclusion

WSNs rely on wireless communication to transmit data between nodes in the network. However, this communication can be disrupted by various forms of interference, such as jamming, which can prevent the successful transmission of data. To address this issue, various techniques have been developed to provide WSNs with a way to resist or mitigate the effects of jamming. These techniques, known as anti-jamming techniques, are designed to allow the network to detect and counteract jamming attacks. In this project, we have designed such a technique using a combination of *signal strength monitoring* and *channel hopping* and deployed it in a simulated and real WSN. Furthermore, we have developed a jammer that combines *random* and *channel hopping* jamming techniques, to test the effectiveness of the designed anti-jamming technique. From the results after testing it was found that the WSN was successfully able to mitigate the jamming attack with a minor loss of packets. However, the proposed anti-jamming technique does contain some vulnerabilities that will need to be addressed in the future.

Overall, the use of anti-jamming techniques in WSNs is crucial for maintaining reliable communication and ensuring a successful transmission of data. The project has explored the importance of a deep understanding of both the attacking and defending aspects of the field to implement successful designs. While no single technique is perfect, a combination of different methods can be used to combat jamming and maintain the integrity of the WSN effectively.

6.1 Future work

For future work on this project, we consider the following to be of the highest priority:

- **Energy efficiency**: In WSNs it is very important to be as energy efficient as possible to increase the up-time of the WSN. In this project, we have no measure of the energy cost of the proposed anti-jamming method, although one could infer that it is expensive to continuously measure the RSSI. For future work, it would be valuable to measure the energy consumption of the network motes and make an effort to optimize the anti-jamming technique with respect to energy and required computations.
- **Test implementation with unknown adversary**: As this project has developed both the jammer and the anti-jammer there are surely some functionalities where we unknowingly have taken advantage of knowledge from their counterparts. This might have skewed the results so an interesting task would be to test the implementation using external tools as the adversary. This would help to observe the performance in a real-world setting.
- **Reliable channel hopping**: With the current channel hoping mechanism of the WSN there is a chance that the motes will be scattered across different channels. Currently,

there is no way for the motes to synchronize on a common channel if this happens, as they will all be hopping with the same frequency. The current strategy will need to be extended to deal with this particular scenario.

Bibliography

[1] "RSSI values for good/bad signal strength," Mist. (), [Online]. Available: https://www.mist.com/documentation/rssi-values-good-bad-signal-strength/ (visited on 12/19/2022).

[2] J. Kanwar, "Detecting and identifying radio jamming attacks in low-power wireless sensor networks," M.S. thesis, Luleå University of Technology, Department of Computer Science, Electrical and Space Engineering, 2021, p. 46.

[3] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "A survey on jamming attacks and countermeasures in wsns," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, pp. 42–56, 2009. DOI: 10.1109/SURV.2009.090404.

[4] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 101 089, 2022, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101089.

Appendix A

Results of mote test node 1 and node 3

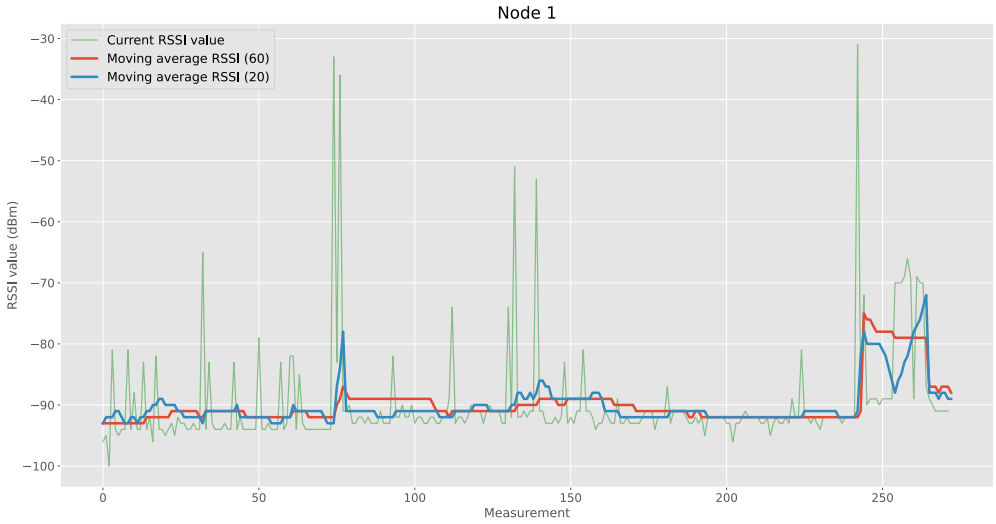


Figure 12: Results of real-life test of node 1

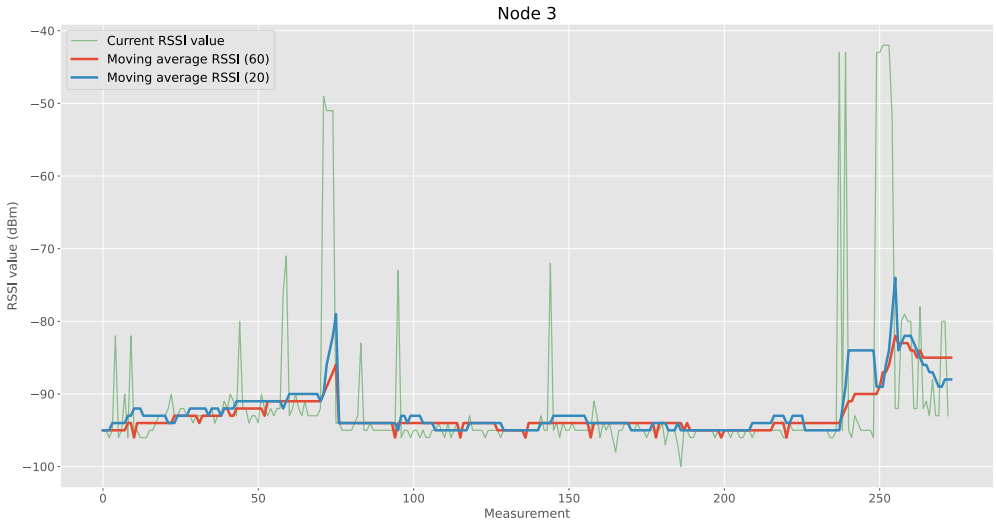


Figure 13: Results of real-life test of node 3